The Journal of Systems and Software 208 (2024) 111908

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Harmonizing DevOps taxonomies — A grounded theory study ☆

Jessica Díaz [a,*], Jorge Pérez [a], Isaque Alves [b], Fabio Kon [b], Leonardo Leite [b], Paulo Meirelles [b], Carla Rocha [c]

[a] ETSI Sistemas Informáticos, Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, Spain
[b] Department of Computer Science, University of São Paulo, Brazil
[c] Faculty of Gama, University of Brasilia, Brazil

ARTICLE INFO

Dataset link: https://github.com/jdiazfernandez/devops_taxonomies.github.io/tree/V1.0.0#readme, https://doi.org/10.5281/zenodo.6948397

Keywords:
DevOps team structures
DevOps taxonomies
Grounded theory
Inter-coder agreement

ABSTRACT

Context: DevOps responds to the growing need of companies to streamline the software development process and thus has experienced widespread adoption in the past few years. However, the successful adoption of DevOps requires companies to address important cultural and organizational changes. Nevertheless, it is crucial to recognize that various DevOps taxonomies exist, both from academic and practitioner perspectives, which may lead to misleading or failed adoption of DevOps.
Objective: This paper presents empirical research on the structure of DevOps teams in software-producing organizations. The goal is to better understand the organizational structure and characteristics of teams adopting DevOps by harmonizing the existing knowledge.
Methods: To achieve this, we employed a grounded theory approach with collaborative coding, involving two research groups. Inter-Coder Agreement (ICA) was utilized to guide the discussion rounds. We conducted a comprehensive analysis of existing studies on DevOps teams and taxonomies to gain a deeper understanding of the subject.
Results: From the analysis, we built a substantive and analytic theory of DevOps taxonomies. The theory is *substantive* in that the scope of validity refers to the ten secondary studies processed and *analytic* in that it analyzes "what is" rather than explaining causality or attempting predictive generalizations. A public repository with all the data related to the products resulting from the analysis and generation of the theory is available.
Conclusions: We built a theory on DevOps taxonomies and tested whether it harmonizes the existing taxonomies, i.e., whether our theory can instantiate the others. This is the first step to define which taxonomies are best suited to approach DevOps culture and practices according to the companies' objectives and capabilities.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

DevOps is an organizational transformation originating at the 2008 Agile Conference in Toronto, where P. Debois highlighted the need to resolve the conflict between the development and operation teams when they had to collaborate to provide quick response time to customer demands (Debois, 2008). In today's highly fast-paced and ever-changing workplace, DevOps culture and practices have been consolidating as an approach to respond to the growing need among companies to streamline and automate continuous delivery of new software versions while guaranteeing their correctness and reliability (Díaz et al., 2021).

Investment for DevOps implementation and adoption has grown in the last years, and most surveys show DevOps on the rise. In fact, the global DevOps market has an annual growth rate (CAGR) of almost 20%–25% until 2030 (2020 DevOps Trends Survey, 2020). However, successful DevOps adoption requires a deep cultural and organizational change in IT departments of companies. Organizations have difficulty defining a DevOps adoption strategy and establishing successful team structures, which brings negative consequences: Firstly, organizations may face difficulties in designing their team structures to achieve continuous delivery, which leads to confusion and uncertainty about the appropriate steps to take (Leite et al., 2021). Secondly, organizations

☆ Editor: Alexander Serebrenik.

* Corresponding author.
E-mail addresses: yesica.diaz@upm.es (J. Díaz), jorgeenrique.perez@upm.es (J. Pérez), isaque.alves@ime.usp.br (I. Alves), kon@ime.usp.br (F. Kon), leofl@ime.usp.br (L. Leite), paulormm@ime.usp.br (P. Meirelles), caguiar@unb.br (C. Rocha).

https://doi.org/10.1016/j.jss.2023.111908
Received 5 April 2023; Received in revised form 22 August 2023; Accepted 20 November 2023
Available online 22 November 2023

may not fully comprehend the implications of their chosen structure, which may impact software delivery performance (López-Fernández et al., 2021).

Understanding the organizational structure and characteristics of teams adopting DevOps is key to establish a successful adoption strategy and performance. Thus, comprehending the existing theories and representations of team taxonomies is critical to guiding companies in a more systematic and structured DevOps adoption process.

Several authors have attempted to describe one or several team structures and how their characteristics impact the software delivery performance (Leite et al., 2021; López-Fernández et al., 2021; Shahin et al., 2017; Nybom et al., 2016a; Macarthy and Bass, 2020; Luz et al., 2019; Puppet and CircleCI, 2021). However, these works do not adopt a shared vocabulary or concepts to describe each team structure. Thus, the literature presents a considerable amount of empirical research from data collected from a large number of organizations (of different sizes, businesses, and contexts), but there is no subsequent analysis to extract common understanding in order to (i) synthesize knowledge for practitioners and (ii) update the state-of-the-art for researchers.

This paper introduces innovative empirical research on the configuration of DevOps teams within software-producing organizations, specifically examining how these organizations structure their development and infrastructure/operation teams to embrace DevOps culture and practices. The primary aim is to gain a deeper comprehension of the organizational setup and team attributes within the scope of DevOps adoption, by synthesizing existing knowledge. In pursuit of this objective, the study delves into the core principles and defining characteristics that delineate the structures of DevOps teams. Additionally, it explores the intricate connections among these principles and attributes. To achieve this, the qualitative nature of the research led to the application of Grounded Theory (GT) (Glaser and Strauss, 1967; Charmaz, 2014) as the chosen research methodology. This approach played a pivotal role in harmonizing diverse DevOps taxonomies, a feat accomplished through meticulous curation and analysis of existing literature pertaining to DevOps teams and their respective taxonomies.

GT involves one or more human analysts reading textual data (e.g., interview transcripts, documents, emails, discussion forum posts, field notes), interpreting and labeling these data (*coding*), recording their thoughts in notes called *memos*, organizing the data, labeling them into categories, and constantly comparing and reorganizing these categories until a mature or *saturated* theory emerges (Stol et al., 2016). Although GT has been mainly used to analyze primary or field data, GT has also being widely adopted for systematic analysis of secondary data such as academic papers and grey literature (Wolfswinkel et al., 2013). In fact, Glaser & Strauss is their seminal work (Glaser and Strauss, 1967) state that field data, historical documents, and other library materials lend themselves wonderfully to the comparative method, which is key in GT. Furthermore, analytical techniques pioneered in the GT literature, such as open/initial, selective/focused, axial, and theoretical coding (Stol et al., 2016) may involve multiple researchers to capitalize on the potential benefits of *collaborative data analysis* (Hall et al., 2005; Guest and MacQueen, 2008; Cornish et al., 2014), i.e., "*the processes in which there is joint focus and dialogue among two or more researchers regarding a shared body of data to produce an agreed interpretation (Cornish et al., 2014). A shared understanding of the phenomenon being studied (Saldaña, 2012)*".

In this research, we applied a novel GT process (Díaz et al., 2023) that extends the Charmaz GT variant to allow multiple researchers to participate in the coding process (*collaborative coding*). It is a way of juxtaposing and integrating multiple interdisciplinary and international perspectives (Cornish et al., 2014), so as to to counteract individual biases (Olson et al., 2016) and enhance/increase credibility (Olson et al., 2016), trustworthiness (Patton, 1999), and rigor (Dubé and Paré, 2003). Collaborative coding is also said to enforce systematicity, clarity, and transparency (Hall et al., 2005). The Inter-Coder Agreement (ICA) measurement is utilized within a process to facilitate discussions and identify concepts that raters may have different perspectives on. This method helps to analyze and interpret each component independently, leading to a more comprehensive understanding of the various elements at play, therefore developing a shared interpretation in qualitative research (Armstrong et al., 1997; Weston et al., 2001; Campbell et al., 2013; MacPhail et al., 2016; McDonald et al., 2019; O'Connor and Joffe, 2020).

Therefore, this paper extends the state-of-the-art by introducing a theory on DevOps team structures and taxonomies using collaborative qualitative analysis. This harmonization enables to gather different cultural and organizational nuances of the companies that were analyzed in existing studies. Extensive research data is made available as open data[1] so that other researchers can verify and extend this work in new directions.

The structure of the paper is as follows. Section 2 provides an overview of the rise of the DevOps culture and existing work on team structures. Section 3 is devoted to the research methodology applied in this work. The development and examination of the validity of the created theory on harmonizing DevOps team structures are presented in Sections 4 and 5, respectively. In Section 7, we analyze the threats to the validity of this theory and its limitations. Finally, Section 8 draws the main conclusions of our work.

2. Background and related work

Before presenting our research methodology, let us first discuss some of the key concepts around DevOps, DevOps adoption, and team taxonomies that serve as a foundation for our work.

2.1. DevOps

The study by Iden et al. (2011), although it does not explicitly mention DevOps, is one of the seminal papers that empirically analyzed the conflict between development and operations teams when they have to collaborate. This follows the idea initiated by Debois (2008) and Flickr employees (Allspaw and Hammond, 2009). Iden et al. (2011) used the Delphi method (brainstorming with 42 Norwegian IT experts, reduction, and ranking) to provide a key baseline for analyzing the problems that reveal the lack of cooperation between developers and IT operations personnel. Later, Nybom et al. (2016b) interviewed 14 employees of an organization to analyze the benefits of dev & ops collaboration, such as improved trust and smoother work flow. Then, some studies followed (Erich et al., 2014; Lwakatare and Kuvaja, 2016; Riungu-Kalliosaari et al., 2016; Luz et al., 2019).

In this research, we adopted the DevOps definition of Leite et al. (2019): *"DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability"*.

The goal is to apply practices and cultural values to reduce the barriers between development and operations teams. To understand these practices and values, Leite et al. (2019) presented a DevOps conceptual map with four subfields: People & Process (Management Perspective) and Delivery & Runtime (Engineering Perspective). The *People* subfield focuses on the union of the development and operations team through the culture of collaboration, enabling breaking down silos, sharing of knowledge, and achieving autonomy. The *Process* subfield presents concepts that aim to achieve a business outcome, such as reducing risk and cost, complying with regulations, and improving product quality and customer satisfaction (Shahin et al., 2016). The *Runtime* presents security, scalability, and infrastructure concepts as code. Finally, *Delivery* is related to providing automation, frequent and reliable release process, and support enabling Continuous Delivery and Deployment.

[1] https://github.com/jdiazfernandez/devops_taxonomies.github.io/tree/V1.0.0#readme

2.2. DevOps adoption and team taxonomies

Over the last decades, organizations have become more interested in the DevOps benefits (Díaz et al., 2021) and have looked for ways to adopt and handle all the concepts and techniques around the term. Google, for example, adopted Site Reliability Engineering (SRE), a term coined by Ben Treynor Sloss, which implements part of the DevOps philosophy but has its own principles, such as *Operations Is a Software Problem and Work to Minimize Toil* (Murphy et al., 2018). Amazon became known for the premise *"You build it, you run it"*, presented by Werner Vogels, CTO of the company. They showed that DevOps is an effective model for improving enterprise IT delivery. These companies implemented DevOps, each with its maturity and structure, and in the case of Google and Amazon, they established new methods and principles that suited the company's needs.

As a consequence of DevOps adoption, companies change their team structures and relationships to address organizational silos and conflicts, ownership sharing, and cultural values. One of the first papers on analyzing team structures is the work by Shahin et al. (2017). They conducted a mixed-method empirical study that collected data from 21 interviews in 19 organizations and a survey with 93 practitioners. They identified four common types of team structures: separate Dev and Ops teams with higher collaboration; separate Dev and Ops teams with a facilitator(s) in the middle; small Ops team with more responsibilities for the Dev team; and no visible Ops team.

Since then, other authors have attempted to describe one or several team structures, and how their characteristics impact the software delivery performance (Leite et al., 2021; López-Fernández et al., 2021; Macarthy and Bass, 2020; Luz et al., 2019; Puppet and CircleCI, 2021; Zhou et al., 2022). Hence, Leite et al. (2021) identified four common organizational structures: (1) Siloed Departments, (2) Classical DevOps, (3) Cross-functional Teams, and (4) Platform Teams. Meanwhile, López-Fernández et al. (2021) also identified four team structures: (A) Interdepartmental Dev & Ops collaboration, (B) Interdepartmental DevOps team, (C) Boosted (cross-functional) DevOps team, and (D) Full (cross-functional) DevOps team. These models are not equivalent, but they have much in common. For example, (1) and (A) are practically the same team structure. Both are characterized by a certain collaboration of Dev and Ops teams but well-defined responsibilities and suffer silo-related problems and low performance. Nevertheless, (1) and (2) are not supported by any horizontal team, whereas (A) and (B) are usually supported by a platform team that is usually limited to providing the required DevOps infrastructure.

To illustrate the challenges of having multiple taxonomies, consider the classic Siloed Department. Leite et al. (2021) identified seven core properties that characterize a Siloed Department: well-defined roles, conflicts among silos, limited access to production by developers, neglect of non-functional requirements by developers, limited DevOps initiatives, limited delivery performance, and insufficient embrace of automated tests. Conversely, López-Fernández et al. (2021) characterized the same team structure with the following properties: multiple leadership, shared ownership, occasional collaboration, organizational and cultural silos, and low autonomy. Although both descriptions accurately represent Siloed departments, at first glance, the properties appear dissimilar, which can lead to confusion and difficulties in understanding and adopting a particular DevOps team structure.

Therefore, these empirical studies seek to understand and categorize team structures, also known as team taxonomies.[2] They are representations of groups organized by their characteristics (silos, communication, collaboration, and others). Comprehending these theories and representations of team taxonomies is critical to guide the company in a more systematic and structured DevOps adoption process.

[2] Collections of classes wherein each class is an abstraction that describes a set of properties shared by the instances of the class (Ralph, 2019)

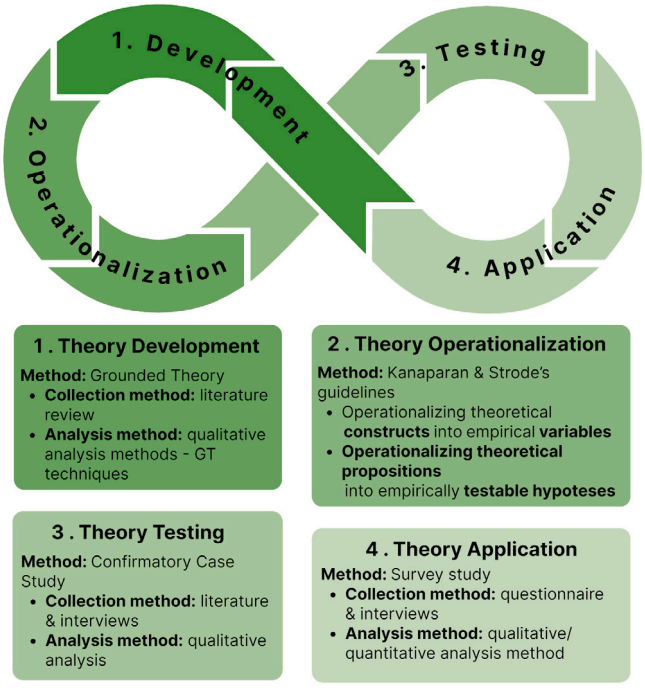

Fig. 1. Methodological framework.

Team structures reflect continuous delivery and the reliability of new software versions. Hence, Leite et al. (2021) and López-Fernández et al. (2021) show that when there is a breakdown of silos, companies tend to increase the performance of delivery (frequency of deployment, time from commit to production, and mean time to recovery). Therefore, according to the companies' objectives and limitations, it is possible to define which structure is best suited to their context.

While related work (e.g. Macarthy and Bass (2020), Nybom et al. (2016b), among others) and previous work (i.e. Leite et al. (2021) and López-Fernández et al. (2021)) focused on identifying and describing DevOps team taxonomies using primary data, our study takes a more comprehensive approach by integrating knowledge from various sources on the same topic. This approach allows us to harmonize the knowledge acquired from different studies and provide a more robust theory. Additionally, our research acknowledges the inherent subjectivity of qualitative research, and we account for potential influences such as researchers' geographical location and researcher's ontological view. By doing so, we aim to ensure a more accurate and unbiased representation of the DevOps team structures and related aspects. Furthermore, our previous works have laid the foundation for this research. Building on those works, we expand the analysis and incorporate new insights to understand better DevOps team taxonomies and their implications on software delivery performance.

3. Research methodology

The contribution described in this paper is part of broader research in theory-building in the context of DevOps Team Taxonomies. To analyze and describe this phenomenon, we adopted a multimethod or mixed approach (Creswell, 2003), in which two different research groups participated. The outcomes of theory-building research are enriched by building theory from multiple research perspectives and methods (Lynham, 2002). The methodological framework depicted in Fig. 1 illustrates the stages of this broader research.

Theory-building is a continuous process of refinement and improvement that consists of four stages ((Lynham, 2002), p. 229):

1. Conceptual development - the conception of pertinent constructs and relationships through inductive and abductive processes

(Sjøberg et al., 2008). The output of this phase is an explicit, informed, conceptual framework that often takes the form of a model ((Lynham, 2002), p. 232).

2. Operationalization - the conceptual elements must be converted into observable entities and measurable units that can be further inquired into and (dis-)confirm (Lynham, 2002; Sjøberg et al., 2008).

3. Testing - the examination of the validity through empirical studies to purposefully inform and intentionally confirm or disconfirm/reject the theoretical framework central to the theory (Lynham, 2002; Sjøberg et al., 2008).

4. Application - the study, inquiry and understanding of the theory in action, i.e., in real situations in practical disciplines (Lynham, 2002).

This paper describes the execution and results of Phase 1, i.e., the development of the theory or a conceptual model. Thus, this is the first paper in this current research on theory building for harmonizing DevOps Team Taxonomies based on existing literature (i.e., secondary study). Before, the authors, who belong to different research groups, researched DevOps Team Taxonomies independently and separately, based on primary data from interviews.

3.1. Theory development methods

This research is based mainly on the constructivist model as an underlying philosophy (epistemological and ontological positions) (Easterbrook et al., 2008). Constructivism or interpretivism states that scientific knowledge cannot be separated from its human context. It also states that a phenomenon can be fully understood by considering the participants' perspectives and context. Therefore, the most suitable methods to support this approach are those that collect rich qualitative data from which theories (tied to the context under study) may emerge.

One of these methods is Grounded Theory (GT), which aims at the iterative development of a theory from qualitative data (Glaser and Strauss, 1967) and encourages deep immersion in the data (Ralph, 2019). *"In grounded theory, initial analysis of the data begins without any preconceived categories. As interesting patterns emerge, the researcher repeatedly compares these with existing data and collects more data to support or refute the emerging theory"* (Easterbrook et al., 2008). GT allows in-depth analysis of the phenomenon to be studied, here, how software-producing organizations structure their development and infrastructure teams and how they interact when they adopt the DevOps culture and practices, i.e., which DevOps taxonomies exist. Thus, GT is adequate for our purposes, and according to our philosophical stance, we used *Constructivist Grounded Theory* (the Charmaz's GT variant (Charmaz, 2014)).

When utilizing GT with secondary data sources, the data are derived from published papers rather than customary open-ended interviews, ethnographic observational notes, or conversational analysis coded transcripts (Wolfswinkel et al., 2013). The researchers are then presented with one or more unstructured sets of published studies, primarily consisting of individual empirical studies. Using (open, selective, theoretical) coding, the researchers conceptualize and articulate the hidden aspects of a set of relevant excerpts identified during their comprehensive review of single studies. This kind of analysis is performed until 'theoretical saturation' has occurred (no new concepts, properties or interesting links arise). This approach enables researchers to create a representative view of data relevant to the specific research question(s) posed (Wolfswinkel et al., 2013).

Specifically, we applied a novel GT process (Díaz et al., 2023) that extends the Charmaz's GT variant to allow multiple researchers to participate in the coding process, known as *collaborative coding*. This GT process ensures consensus on the constructs that support the theory, thus improving the rigor of qualitative research (cf. Díaz et al. (2023)). We used a collaborative coding approach, in which a team of four researchers analyzed qualitative data independently and systematically. This method is particularly crucial in the context of DevOps team taxonomies due to the intricate nature of this domain. DevOps team taxonomies involve a diverse range of non-technical and technical concepts, spanning development, operations, and management. By leveraging the diverse perspectives and backgrounds of multiple researchers, we believe that the analysis is enriched, leading to improved research quality. To evaluate the level of agreement among coders and facilitate the resolution of disagreements, we utilize inter-coder agreement (ICA) analysis. This method quantifies the degree to which different raters assign the same precise value, such as a code or category, to each qualitative data item or quotation (Gisev et al., 2013).

Collaborative coding involving multiple raters often results in consensus. This consensus can be achieved through various means, including group discussion, dialogic intersubjectivity, coder adjudication, and simple group consensus as the desired agreement goal (Saldaña, 2012). However, to improve the coding process, it is crucial to measure consistency and agreement among coders. Here, agreement techniques come into play, allowing researchers to gauge the level of agreement among multiple coders.

By promoting systematicity, communicability, and transparency in the coding process, measuring consistency and agreement among raters encourages reflexivity and dialogue within research teams. This, in turn, helps to enhance the trustworthiness of research for diverse audiences. Therefore, ICA techniques play a critical role in ensuring the reliability and validity of research findings, particularly when multiple coders are involved (González-Prieto et al., 2023).

3.2. Data collection

GT involves iteratively performing interleaved rounds of qualitative data collection and analysis to lead to a theory (e.g., concepts, categories, patterns) (Ralph et al., 2021). In our case, the collected material is secondary data from scientific papers and grey literature. Grey literature is any document not controlled by commercial publishers, such as source code files, wikis, roadmaps, and mailing list messages; it also includes publications not indexed by scientific repositories such as working papers, white papers, annual/technical reports, blogs, videos, and web pages (Rothstein et al., 2005). Alongside commercial and academic publications, grey literature can be used to broaden the understanding of the structure of DevOps teams. Grey literature is a leading source for identifying topics and gaps not yet covered by academic literature. It enables investigating more up-to-date, and emerging information since academic studies entail longer publication delays due to its peer-review process (Paez, 2017).

We initially collected data from scientific databases according to the *purposed sampling strategy*. Then we moved on to *theoretical sampling* and iteratively collected more data — both from scientific studies and grey literature. Our search encompassed prominent academic databases such as IEEE and comprehensive searches on Google Scholar. Each analyzed document described at least one DevOps team structure and its properties based on the concepts and categories relevant to the emerging theory. We collect new data for analysis in each iteration, ensuring that at least one new document is added during each cycle. This iterative data collection process continues until the ICA value exceeded a threshold and the theoretical saturation was reached.

3.3. Qualitative data analysis

To conduct a constructivist GT, we followed the subsequent stages detailed in the next section: initial/open coding, selection of core categories, selective coding, sorting, and theoretical coding (see Fig. 2). The outputs of these stages are the coding, the memos, the categories, and the resulting theory, all managed through the Atlas.ti v9 tool (ATLAS.ti Scientific Software Development GmbH, 2019). To support our findings, we also included some excerpts from the collected data to maintain the chain of evidence.

Initial/open coding and selective coding (see white rectangles in Fig. 2, which are decomposed in Fig. 3) are characterized by (i) being

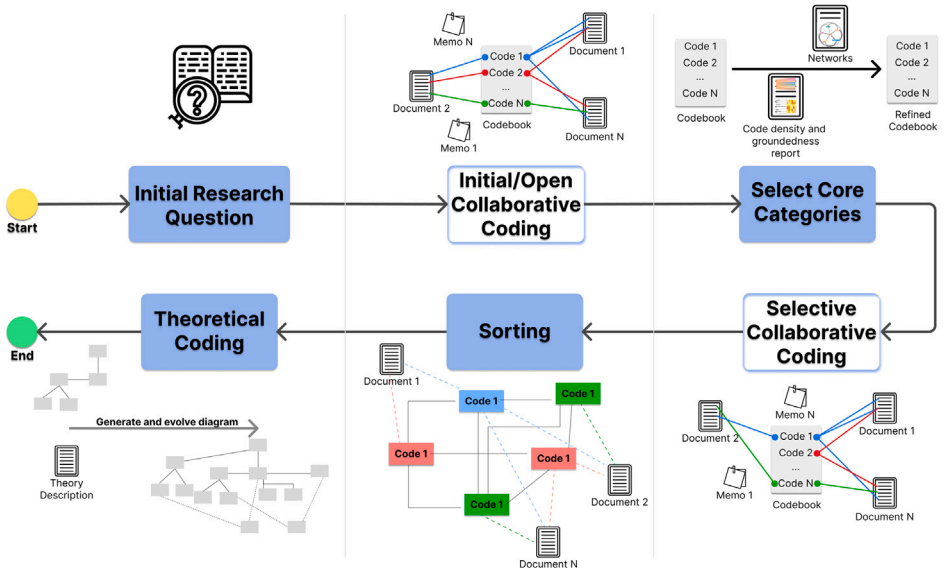

Fig. 2. Using ICA in GT studies (Díaz et al., 2023). Part 2: Key stages and artifacts of Qualitative Data Analysis.

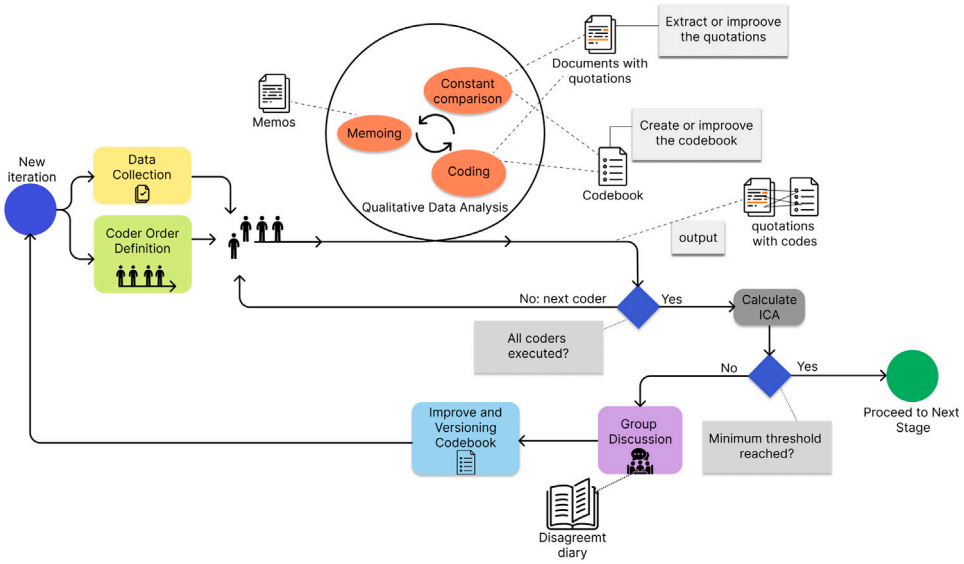

Fig. 3. Using ICA in GT studies (Díaz et al., 2023). Part 1: High-level iteration cycle.

collaborative, involving various coders in these stages, (ii) consisting of multiple rounds of coding, constant comparison, and memoing, specifically one round per coder involved in the collaborative coding, and (iii) being performed in various iterations until Inter-Coder Agreement (ICA) reaches a threshold. Hence, a first coder analyzes the selected documents, line by line, extracting quotations (segments of texts) and assigning codes to these quotations. During this process, the coders also write notes that are relevant to the research. We call these "memos". At the end of this analysis, the coder creates memos, a codebook, quotations, and the relation between the codes and quotations (i.e., the codes assigned to each quotation). The following coders analyze the same documents that the previous coder analyzed. These quotations are visible, but the coders never see the codes assigned to each quotation. They have access to the codebook and the documents with the quotations. Then, subsequent coders can create new quotations, and create or modify codes and memos if necessary. Once all coders have analyzed and coded the documents, the following step is to calculate the ICA, see activity *Calculate ICA* in Fig. 3. If the ICA results exceed a specified threshold, then we *Proceed to Next Stage*. If not, a *Group Discussion* is required. This discussion aids coders in refining their comprehension, aligning their perspectives, and primarily focusing on resolving points of disagreement. The objective behind conducting a new iteration is to validate whether the discussion has resulted in a consensus. Utilizing the improved codebook on new data subsequently ensures that researchers can collectively achieve a consensus on the accumulated information or continue deliberating and analyzing new data if necessary.

ICA helps and guides the group discussion to identify mistakes, coding disagreements, and weaknesses in the categories, improving the codebook and code descriptions. All decisions are documented in the *Disagreement Diary* document. After that, we repeat the process until a previously-defined minimum threshold is reached. This process is fully described in previous work (Díaz et al., 2023).

4. A theory on DevOps team structures

This section describes a theory on how software-producing organizations structure their development and infrastructure/operation teams

Table 1
Description of the documents selected.

ID	Source	Year	Name
D1	Scientific paper	2021	DevOps Team Structures Characterization and Implications (López-Fernández et al., 2021)
D2	Scientific paper	2021	The organization of software teams in the quest for continuous delivery (Leite et al., 2021)
D4	Scientific paper	2020	An Empirical Taxonomy of DevOps in Practice (Macarthy and Bass, 2020)
D5	Scientific paper	2016	On the Impact of Mixing Responsibilities Between Devs and Ops (Nybom et al., 2016a)
D6	Scientific paper	2017	Adopting Continuous Delivery and Deployment: Impacts on Team Structures, Collaboration and Responsibilities (Shahin et al., 2017)
D7	Scientific paper	2019	Adopting DevOps in the real world: A theory, a model, and a case study (Luz et al., 2019)
D8	Scientific paper	2011	Why Enterprises Must Adopt Devops to Enable Continuous Delivery (Humble and Molesky, 2011)
D9	Gray literature	2022	Site Release Engineering book Chapter 1 - How SRE Relates to DevOps (Murphy et al., 2018)
D11	Gray literature	2020	State of DevOps Report (Puppet and CircleCI, 2021)
D12	Scientific paper	2022	A Cross-Company Ethnographic Study on Software Teams for DevOps and Microservices: Organization, Benefits, and Issues (Zhou et al., 2022)

and how they interact when they adopt the DevOps culture and practices. In this process, four raters (coders) were involved: (R1) Jessica Díaz; (R2) Isaque Alves; (R3) Jorge Pérez; and (R4) Carla Rocha. The process results are available in a public repository (see Section Data Availability) , i.e., a replication package to motivate others to provide similar evidence by reproducing this GT study. The steps and iterations to analyze the data and construct the theory are described as follows.

4.1. Initial/open coding

This activity (see Fig. 2) aims to discover the concepts underlying the data and instantiate them in the form of codes. In each open coding iteration, n documents are analyzed and chopped into quotations assigned to a previously discovered code or to a new one that emerges to capture a new concept.

Iteration 1

In the first open coding iteration, researchers R1-R4 analyzed documents D1 and D2 (see Table 1) as follows.

First, R1 selected the quotations and created and assigned codes to these quotations, resulting in an initial codebook. Subsequently, R2 received this initial codebook and quotations, and assigned codes to the quotations as well as created or modified codes when necessary. After completion of the two rounds of coding, Krippendorff's coefficients α (Hayes and Krippendorff, 2007; Krippendorff, 2004) were computed (see González-Prieto et al. (2023) for a detailed description of the use of these techniques in qualitative research). Specifically, $Cu\text{-}\alpha$[3] and $cu\text{-}\alpha$[4] coefficients were computed, and their values are shown in Table 2. As we can observe from this table, the value of the global coefficient $Cu\text{-}\alpha$ did not reach the acceptable threshold of 0.8, as defined in the literature (Krippendorff, 2004). For this reason, a group discussion and review meeting was necessary to examine disagreements and the application criteria of the different codes, and thus, improve the codebook. Then, R3 and R4 replicated the process, and after completion of the two rounds of coding, again the global coefficient $Cu\text{-}\alpha$ did not reach the acceptable threshold (see Table 2). Thus, a new review meeting was held to discuss disagreements and the application criteria of the different codes, in such a way that the codebook was improved.[5]

The by-products of this first iteration were a total of 71 quotations extracted from the documents and a codebook composed of 21 codes categorized into seven semantic domains or categories: (S1) Automation; (S2) Culture; (S3) Management; (S4) Monitoring; (S5) Organizational Structure; (S6) Sharing; and (S7) Skills & Roles.[6]

Iteration 2

Researchers R1-R4 analyzed document D4 (see Table 1). Since the coders agreed on a common codebook in the previous iteration, we expected a more significant agreement that materializes as a higher value of ICA. Table 3 shows the ICA values for this second iteration. From the results of the table, we observe that after this refinement of the codebook, $Cu\text{-}\alpha$ reached a value greater than the required agreement threshold (0.905). Thus, the open coding process stopped: we reached a consensus on interpreting the codes presented in the codebook, and we proceed with selecting core categories and selective coding.

As a by-product of this second iteration, 7 new codes and a new semantic domain emerged, leading to a new version of the codebook with 28 codes and eight semantic domains (see Table 4 and the public repository for a full description of the codebook).

[3] The $Cu\text{-}\alpha$ coefficient measures the degree of agreement in the decision to apply different semantic domains, independent of the chosen code.

[4] The $cu\text{-}\alpha$ coefficient is computed on a specific semantic domain S. It indicates the degree of agreement with which coders identify codes within S.

[5] Meeting results are documented in a *disagreements diary* file, which is available in the public repository.

[6] Atlas.ti projects and codebooks for each iteration are available in the public repository.

Table 2
Values of the different Krippendorff's α coefficients in the open coding iteration 1. Values lower than the threshold (< 0.80) are shown in Red.

Coders	$Cu-\alpha$ per semantic domain							$Cu-\alpha$
	S1	S2	S3	S4	S5	S6	S7	
R1 & R2	1	N/A	0.941	0.096	1	0.906	1	0.789
R3 & R4	0.839	1	N/A	1	0.915	0.976	0.918	0.761

Table 3
Values of the different Krippendorff's α coefficients in the open coding iteration 2. Values lower than the threshold (< 0.80) are shown in Red.

Coders	$Cu-\alpha$ per semantic domain								$Cu-\alpha$
	S1	S2	S3	S4	S5	S6	S7	S8	
R1-R4	1	1	N/A	1	N/A	0.708	1	1	0.905

Table 4
Domains and codes resulting from open coding.

Domain: AUTOMATION (S1)	Domain: CULTURE (S2)
Automated application life-cycle management	collaboration
Automated infrastructure management	communication
Platform builder	cultural silos/conflicts
Platform servicing	values & best practices
Domain: IT INFRASTRUCTURE (S3)	**Domain: MANAGEMENT (S4)**
Cloud	Leadership & management
Containerization	Rotary human resources
Hybrid (on-premises & cloud)	Team self-organization & autonomy
On-premises	Transfer of work between teams
Domain: MONITORING (S5)	**Domain: ORGANIZATIONAL STRUCTURE (S6)**
Delivery performance	Devops (bridge) team
End-to-end product vision	Enabler (platform) team
	Organizational silos/conflicts
	Small size teams (two pizza rule)
Domain: SHARING (S7)	**Domain: SKILLS & ROLES (S8)**
Alignment of dev & ops goals	Cross-functionality/skills
Knowledge sharing	Evangelization and mentoring
Responsibility/ownership sharing	Role definition/attributions

Table 5
Groundedness & density of semantic domains (open coding stage).

ID	Semantic domain	Density
S5	Organizational Structure Gr = 59	112
S2	Culture Gr = 41	84
S6	Sharing Gr = 40	84
S7	Skill & Roles Gr = 36	75
S1	Automation Gr = 34	69
S3	Management Gr = 26	59
S4	Monitoring Gr = 8	21
S8	IT Infrastructure Gr = 7	18

Table 6
Groundedness & density of codes (open coding stage).

Code	Density
responsibility/ownership sharing Gr = 26	61
cross-functionality/skills Gr = 23	54
collaboration Gr = 25	50
enabler (platform) team Gr = 23	50
organizational silos/conflicts Gr = 18	36
platform servicing Gr = 13	34
team self-organization & autonomy Gr = 11	30
devops (bridge) team Gr = 15	29
role definition/attributions Gr = 13	26
knowledge sharing Gr = 10	22
transfer of work between teams Gr = 10	21
communication Gr = 6	15
end-to-end product vision Gr = 4	14
values & best practices Gr = 5	13
automated infrastructure management Gr = 4	11
cultural silos/conflicts Gr = 5	11
delivery performance Gr = 5	11
evangelization and mentoring Gr = 4	11
rotary human resources Gr = 3	9
hybrid (on-premises & cloud) Gr = 3	8
platform builder Gr = 4	8
leadership & management Gr = 3	6
cloud Gr = 2	5
small size teams (two pizza rule) Gr = 3	5
on-premises Gr = 1	3
containerization Gr = 1	2

4.2. Selection of core categories

In this activity (see Fig. 2), R1 and R2 selected the core categories, i.e., the most relevant codes from the 28 codes obtained in the open coding. The selection of core categories is mainly interpretative although we also used the groundedness of the codes and semantic domains (i.e., the number of quotations labeled with a code, and its domain) and the density of the codes and semantic domains (i.e., the number of relationships between codes and between domains, i.e., the co-occurrence of codes in the same quotation). For example, the code "responsibility/ownership sharing" is related to 22 codes (from a total of 28) 61 times, i.e., the density is equal to 61. Tables 5 and 6 show the values for groundedness and density per semantic domain and code, respectively. Fig. 4 shows the density for the code "responsibility/ownership sharing". These are some examples of the information we analyzed in this activity. The detailed analysis is documented in the *selection of core categories* file of the *selection of core categories* folder in the public repository, together with the Excel reports of co-occurrence tables.

As a result of the analysis, we selected six semantic domains (i.e., we discarded S3 and S5) and 19 codes for the next activity. This codebook is available in the codebooks folder in the public repository.

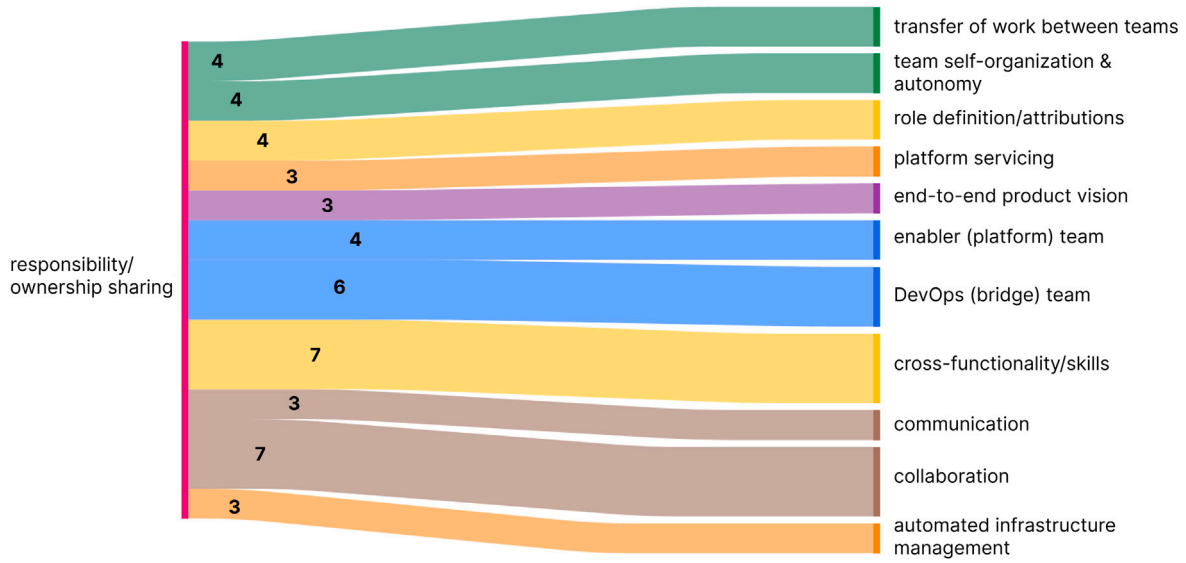

Fig. 4. Illustrative example of density analysis - co-occurrence of the code "responsibility/ownership sharing".

4.3. Selective coding

Selective coding (see Fig. 2) is an inductive-deductive process in which new data are labeled with the codes of selected categories (semantic domains). Coders focus only on core categories, but the number and definition of their inner codes can be modified according to the new data analysis.

Iteration 3

Researchers R1-R4 analyzed documents D5, D6, D7 and D8 (see Table 1). *Cu-α* reached an acceptable agreement threshold (0.864), but we did not reach theoretical saturation and new iterations were required. As a by-product of this iteration, we obtained a new version of the codebook with 23 codes and six semantic domains (see the public repository for a full description of the codebook).

Iteration 4

Researchers R1-R4 analyzed document D9 (see Table 1). *Cu-α* did not reach the acceptable threshold of 0.8. For this reason, a group discussion and review meeting was necessary to discuss disagreements and the application criteria of the different codes. As a by-product of this iteration, we obtained a new version of the codebook with 25 codes and six semantic domains (see the public repository for a full description of the codebook).

Iteration 5

Researchers R1 & R3 analyzed documents D11 and D12 (see Table 1). *Cu-α* reached a value greater than the acceptable agreement threshold (0.956) and we also concluded that we had reached theoretical saturation. In this way, the selective coding process stopped: There was a consensus in interpreting the codes presented in the codebook and we reached theoretical saturation, so we proceeded to the sorting stage. As a by-product of this iteration, we obtained a new version of the codebook with 25 codes and six semantic domains. Table 7 shows an excerpt of the final version of the codebook (full description available in the public repository).

4.4. Sorting procedure

Sorting (see Fig. 2) refers to the conceptual sorting of codes, categories, and memos into an outline or network of an emergent theory, showing relationships between concepts. To this end, we performed a co-occurrence analysis of codes together with the sorting of memos to draw the relationships between the different concepts and outline an emergent theory. All the data we analyzed in this stage is available in the public repository. Fig. 5 shows the result of this analysis: a network composed of concepts (rectangles) and the relationships among them. Next, we describe the most relevant concepts (based on their groundedness), the most relevant relationships (based on their density), and some excerpts as a chain of evidence.[7]

Table 7
Excerpt of the final version of the Codebook.

Domain: AUTOMATION (S1)
Platform servicing: This code refers to the services that an entity (e.g., a team, an external consultant, etc.) offers/provides to product teams (developers and/or operators) to assist them on the DevOps platform, such as: infrastructure management (e.g., containerization, cloud, etc.), infrastructure automation (from low to high levels of automated infrastructure services aka. Infrastructure as Code, IaC), pipeline automation (CI/CD and release tools), IT operation, etc.
Domain: ORGANIZATIONAL STRUCTURE (S6)
Enabler (platform) team: This code refers to specific structures/teams that organizations create to satisfy some needs of product teams. Sometimes it is not necessary to create new structures/teams because the operations team/department takes over these needs and assists product teams. These needs can be platform servicing and tools (mainly for infrastructure and deployment pipelines), consulting, training, evangelization, mentoring, human resources, etc. Thus, they behave as enabler teams by providing these capabilities. These structures/teams are named in different ways, e.g., DevOps Centers of Excellence, chapters, guilds, platform/SRE teams, etc. This code should be used when a quotation explicitly mentions these new structures/teams or when a quotation describes the capabilities (at least three) of these enabler teams.

A keyword in the definition of DevOps is **collaboration** between teams (development, infrastructure, quality assurance, and security teams, among others). This ranges from the lack of collaboration, which may generate conflicts and disagreements on decisions, to daily collaboration (working together regularly). Starting from collaboration, which is the code with the highest groundedness (**grounded = 54**) and density (**density = 89**) at this stage, we analyze the enablers of collaboration between teams as well as the relationships and impact of collaboration on team structures.

A team culture based on **responsibility/ownership sharing *enables* collaboration (density = 14)** - [5:18] *"Mixing the responsibilities brought Devs and Ops closer to each other. Employees mentioned that Devs*

[7] The format for excerpts is [ID_document:quotation_number]*"text"*.

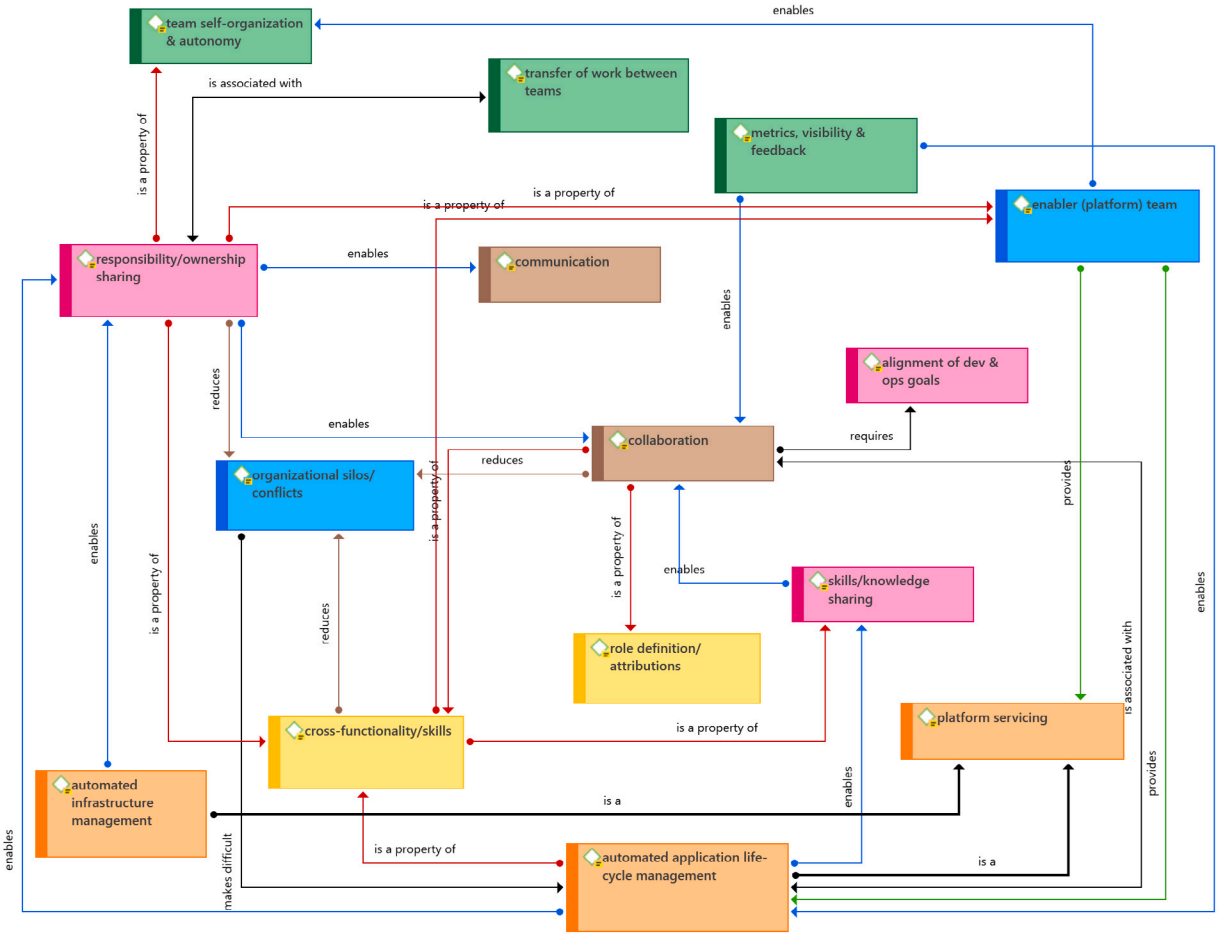

Fig. 5. Relationships among concepts.

and Ops now collaborate on different tasks since they now realize the importance of collaboration".

The second code to be analyzed is **responsibility/ownership sharing (grounded = 49 & density = 80)**. This code ranges from shared responsibility of the products to separate responsibilities and tasks (i.e., each team member has different responsibilities and tasks). Next, we analyze its relationships with others.

Responsibility/ownership sharing *is a property of* cross-functionality /skills (density = 10) teams - [1:25] *"We observed that there exists a relationship between how product teams share the product ownership and how these teams are structured. For example, ID29 shows a high level of sharing of the product ownership within product teams, which are cohesive, small (less than 12 people), and multidisciplinary".*

The complete set of concepts and their relationships, alongside quotation excerpts to illustrate each relationship, is available in the supplementary material indicated in Section Data Availability. The core categories are boxed, while the font size of each category and the thickness of the lines that relate them correspond to the groundedness of semantic domains and the density of codes, respectively.

4.5. Theoretical coding

Theoretical coding (see Fig. 2) is defined as *"the property of coding and constant comparative analysis that yields the conceptual relationship between categories and their properties as they emerge"* (Glaser, 1992). Thus, it is a stage that aims to consolidate the emerging concepts and relationships we obtained in the sorting stage.

According to Gregor's classification (Gregor, 2006), in our case, we developed a theory of the type Analysis: *"Theories of this type include descriptions and conceptualizations of "what is". Also included are taxonomies, classifications, and ontologies in the sense of Gruber (1993)"*. In fact, Gregor points out that *"Some examples of grounded theory can also be examples of Type I theory, where the grounded theory method gives rise to a description of categories of interest"*. Type I theory refers to *"Analytic theories [that] analyze "what is" as opposed to explaining causality or attempting predictive generalizations"*. Such theories are valuable when little is known about the phenomena they describe. This is the case with DevOps team taxonomies, which are relatively new. That is, the theory to be built will answer "what DevOps teams taxonomies exist". We do not attempt to answer questions related to "why such team structures exist" (explanation theory), nor do we intend to develop mathematical/probabilistic models to support predictions (prediction theory), nor do we intend to describe "how to do" things (Design and action theory or prescription theory).

To build the theory, we performed the following steps, which we now detail (Sjøberg et al., 2008):

1. Determining the scope of the theory
2. Defining the constructs & propositions of the theory
3. Providing explanations to justify the theory
4. Examination of validity the theory (see Section 5)

Determining the scope of the theory

The scope of our theory is based on the framework for SE theories by Sjøberg et al. (2008); however, as there are no explicit limitations to add new constructs and the type of relationships that can exist between constructs, we have adapted this framework to our theory. Fig. 6 illustrates a UML class diagram with the constructs

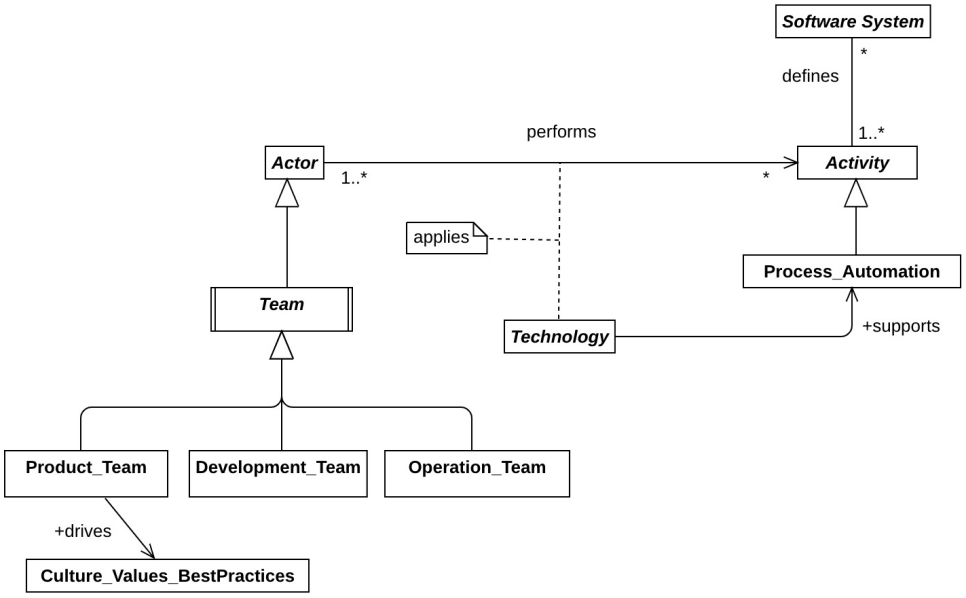

Fig. 6. Scope of the theory.

and relationships that determine the scope of this theory. We use the following four archetype classes: Actor, Technology, Activity, and Software System, which have been represented as abstract classifiers (classes), so that *"An actor applies technologies to perform certain activities on an (existing or planned) software system"* (Sjøberg et al., 2008). Additionally, to meet a theory in the DevOps domain, we added some classifiers (classes) to represent four concepts related to the CAMS framework[8]: culture, automation, measurement, and sharing. [The CAMS framework provides a well-structured and comprehensive approach to understanding and explaining complex systems and processes like DevOps. The CAMS framework helps us leverage DevOps theoretical foundations to conceptualize and delineate the scope of our theory effectively.] The $Culture\_Values\_BestPractices$ class represents the cultural movement promoted by DevOps, its values, and best practices. The $Process\_Automation$ class represents a task to automate any process of the application lifecycle management (ALM) as well as process metrics using some technology. Together with these classifiers, we have added others to represent the classic organizational structure (dev and ops teams) versus product teams driving DevOps culture, principles, and practices. These classes inherit from the $Team$ class, which in turn inherits from $Actor$, and thus execute activities in the context of constructing software systems. $Team$ is declared as abstract and active since instances of its subclasses will be active, and sharing is a main attribute.

Since the theory was constructed on the basis of qualitative analysis of a set of primary papers, it must necessarily be limited to a substantive (local) theory, as opposed to what could be a formal (all-inclusive) theory (Glaser and Strauss, 1967).

Defining the constructs and propositions of the theory

The constructs and the relationships between the constructs (named propositions) that make up the theory were derived from the sorting analysis (see Fig. 5). Table 8 shows the constructs and the code(s) from which they were derived, whereas Table 9 shows the textual statement of the description for a proposition between Construct C9 (Collaboration) and C3 (Team::ownedAttribute). The entire list of propositions is provided as Supplementary Material (see Section Data Availability), which includes more propositions between Construct C9 (Collaboration) and C3 (Team::ownedAttribute), C6 (Automation), and C8 (Silo).

[8] https://www.chef.io/blog/what-devops-means-to-me

The full description of the 28 propositions (see Section Data Availability) characterizes each proposition by means of four elements: (i) the textual statement of the proposition, (ii) its formalization by means of the Object Constraint Language (OCL), (iii) excerpts as a chain of evidence, and (iv) the relationships we defined in the resulting network from the sorting stage (see Fig. 5 for tracing both artifacts). Finally, Appendix shows the constructs and propositions in a UML class diagram together with the relations between the constructs and the elements of the scope.

For readability purposes, this diagram has been divided into some parts (see Figs. 7–10). Fig. 7 shows the $Team$ class and its subclasses, which represent constructs C1 to C3; $Culture\_Value\_BestPractices$, which represents construct C5; $Collaboration$ and $Communication$, which represent constructs C9 and C10, respectively; and $Silo$, which represents construct C8. Fig. 8 shows the $Management$ class and its subclasses, which represent construct C4. Fig. 9 shows the $Automated\_Infrastructure\_Management$ and $Automated\_Application\_Life\_Cycle\_Management$ classes, which represent constructs C6, and $Platform$, which represents Construct C7. Finally, Fig. 10 shows the enumeration types used in the previous diagrams.

Providing explanations to justify the theory

An explanation is a relationship among constructs and other categories that are not central enough to become constructs. An explanation answers a question of why categories and classes are related, and this helps explain how DevOps team structures differentiate in the theory context, including notions of causality and asymmetry (Sjøberg et al., 2008). To that end, the following explanations are based on the constructs and propositions shown in Appendix.

E1 - Team and specializations - We identified four-team specializations: $Product\_Team$, $Development\_Team$, $Operation\_Team$, and $Horizontal\_Team$. Each of these specializations has characteristics that justify its existence, whether in the attributions of variables they inherit from $Team$ or in actions they perform.

A $Product\_Team$ is completely responsible for a product/service (from scoping out the functionality, to architecting, building, and operating it). Thus, it is common to have $responsibility/ownership\_sharing = medium\_sharingorfull\_sharing$, i.e., shared responsibility of products and tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility, monitoring shared responsibility, and incident handling shared responsibility) and $autonomy = self\_organization$, i.e., access to all necessary information to develop, deploy, and operate

Table 8
Constructs.

Construct	Domain/Code
C1. Team. It is an artificial (abstract) concept that represents a team structure of an IT department.	(S6) Organization structure
C2. Team::inheretedMembers. It is the way in which the activities carried out by an organization are divided, organized, and coordinated. Possible team structures are: product teams, horizontal teams (is an artificial (abstract) concept) — either bridge teams and enabler teams —, development teams and operation teams.	•Enabler (platform) team •Devops (bridge) team
C3. Team::ownedAttribute. Attributes that explain and drive the different team structures. In our context, these attributes include the description of organizational, cultural, and technological aspects. These attributes are self-organization and autonomy, blame management, alignment of dev & ops goals, responsibility/ownership sharing, skills/knowledge sharing, stack & tools sharing, cross-functionality, and role definition.	•Self-organization & autonomy •blame •Alignment of dev & ops goals •Responsibility/ownership sharing •Skills/knowledge sharing •stack & tools sharing •Cross-functionality/skills •Role definition/attributions
C4. Management. It is an artificial (abstract) concept that represents the management activities (project and product management, change management, team self-organization, coordination and transfer of work between teams, as well as measuring, monitoring and feedback) to ensure that software development and maintenance are systematic, organized, and quantified.	•Change management •Metrics, visibility & feedback •Product management •Self-organization & autonomy •Transfer of work between teams
C5. Culture. It represents the set of common principles, values, and best practices that impact the way that people in an organization relate to each other and make decisions and actions around application life-cycle management.	•Culture, values & best practices •Continuous improvement
C6. Automation. It represents the set of activities to automate the processes of the application life-cycle management and infrastructure management where applications are deployed.	•Automated application life-cycle management •Automated infrastructure management
C7. Platform. It represents the technology to support automation. It provides interfaces for automated infrastructure management and automated application life-cycle management.	•Platform builder •Platform servicing
C8. Silo. It represents the concept of silos between teams. The silo can be organizational or cultural.	•Cultural silos/conflicts •Organizational silos/conflicts
C9. Collaboration. From the lack or eventual collaboration to daily collaboration.	•Collaboration
C10. Communication. From poor/rare communication to frequent communication.	•Communication

Table 9
Propositions example.

P1. A team culture based on responsibility/ownership sharing enables collaboration.
P2. Promoting Collaboration reduces organizational silos/conflicts.
P3. Automated application life-cycle management is associated with collaboration. Collaboration impacts automated application life-cycle management and vice versa. Automation and collaboration mutually facilitate the adoption of the other, so they are complementary.
P4. A team culture based on knowledge sharing enables collaboration.
P5. If a team is characterized by cross-functionality/skills this will increase collaboration.
P6. Collaboration is a property of teams in which skills take precedence over roles, i.e., the role definition/attributions code; hence, if there are already separate roles, responsibilities are very clear and collaboration is not fostered or promoted.

the product (see Proposition P12). In this way, product teams can innovate quickly with a strong customer focus as there is alignment with business goals ($alignment\_of\_dev\&ops\_goals = product\_thinking$), which is a requirement of a collaboration-based culture (see Proposition P7), i.e., $Collaboration.frequency = daily$ and $Collaboration.quality = high$. Collaboration and responsibility sharing are properties of cross-functional teams (see Propositions P5 and P9, respectively), thus, product teams require to have $cross - functionality/skills = True$, and this reduces silos (see Propositions P2, P10, and P19). All these properties imply that product teams are self-sufficient and capable of promoting innovation with product thinking. The following excerpt supports this explanation [1:11] *"Consolidated product teams, which have dealt with both organizational and cultural silos by aligning dev & ops goals with business goals and show cross-functional teams with shared product ownership, end-to-end product vision and high-levels of self-organization and autonomy"*.

The relationship between $Development\_Team$, which focuses on feature development, and $Operation\_Team$, which focuses on creating and maintaining the project's infrastructure, establishes structures and taxonomies already known and cited by the authors in previous works that depend on the values of some attributes we identified in Construct C3. These attributes are as follows. Well-defined and differentiated roles, i.e., $role\_definition/attributions = True$, usually show poor collaboration ($Collaboration.frequency = eventual$) and communication ($Communication.type = poor/rare$), promoting a transfer of responsibilities. Hence, roles with clear and non-shared responsibilities ($responsibility/ownership\_sharing = minimal$ or $null\_sharing$), build a scenario where the collaboration is not fostered or promoted (see Proposition 6). It generates a transfer of responsibility between the teams (see Proposition 14) instead of sharing common responsibilities. However, as collaboration and shared responsibility increase, silos, and conflicts are reduced or eliminated. The following excerpts support this explanation [2:11] *The classical DevOps structure focuses on collaboration among developers and the infrastructure team. It does not eliminate all conflicts but promotes a better environment to deal with them* [1:9] *"resulting from the creation of teams in which developers and operators daily collaborate, but there exists still a transfer of work between them, showing some cultural barriers"*.

Finally, $Horizontal\_Team$ is a specific structure that helps and supports the needs of multiple development or product teams of an IT department. Sometimes it is not necessary to create new structures because the operation team/department takes over these needs and assists dev/product teams. These needs can be platform servicing and tools for automated application life-cycle management (see Proposition 26) and automated infrastructure management (see Proposition 27), and also consulting, training, evangelization, mentoring, human

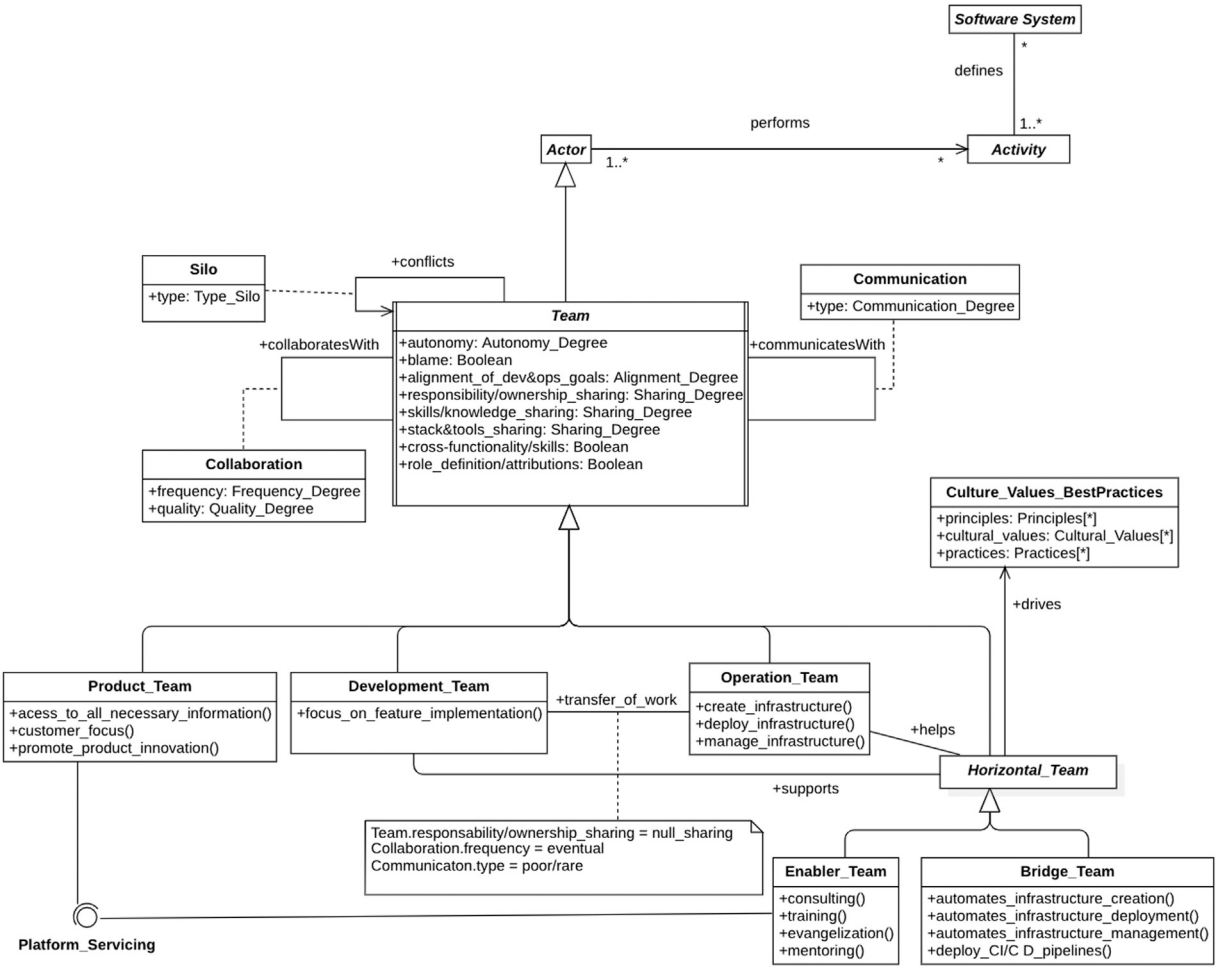

Fig. 7. Constructs of Team class and its subclasses.

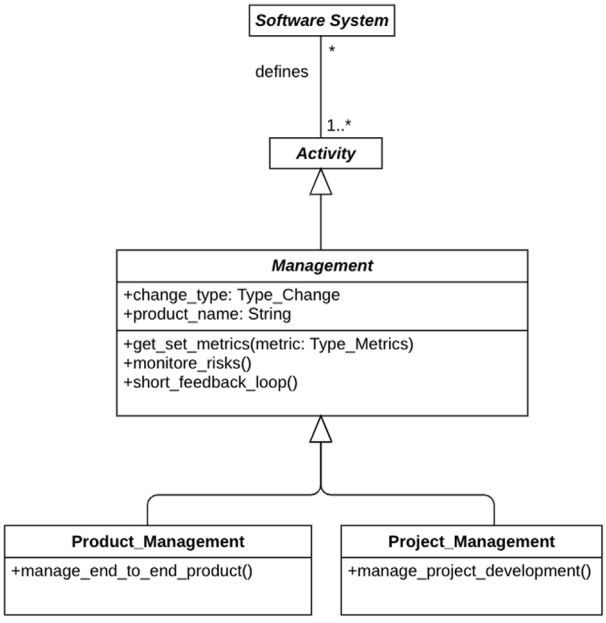

Fig. 8. Constructs of Management class and its subclasses.

resources, etc. $Horizontal\_Team$ is an abstract class as it will only serve as a model for the $Enabler\_Team$ and $Bridge\_Team$, which inherit the attribute $cross-functionality/skills : boolean$, thus, they could be cross-functional or not; usually, an $Enabler\_Team$ or a $Bridge\_Team$ has multiple skills, but it is not mandatory (see Proposition P18).

On the one hand, $Enabler\_Team$ – named in different ways, e.g., DevOps Centers of Excellence, chapters, guilds, platform/SRE teams – provides platform servicing and tools (see Interface $Platform\_Servicing$ and Proposition P25), which enables product teams with automated application life-cycle management (see Proposition 28) and self-organization & autonomy (see Proposition 24), driving the DevOps culture, values, and practices.

On the other hand, $Bridge\_Team$ supports and helps development and operation teams, mainly by deploying and hosting applications in the platforms they build (platform builders), monitoring, and providing support. The engineers of these bridge teams are DevOps practices facilitators; hence, they usually create, deploy, and manage both the infrastructure (environments) and deployment (CI/CD) pipelines, although they may be also involved in other tasks, such as requirements (user stories) analysis and coding. They are usually the bridge interface between developers and IT operations to drive the DevOps culture, values, and practices.

E2 - Management - Management is an important activity that assists in developing a software system. It is responsible for generating metrics and indicators and monitoring risks. There are two management specializations. $Product\_Management$ is responsible for managing the product end-to-end (both development and operation) whereas

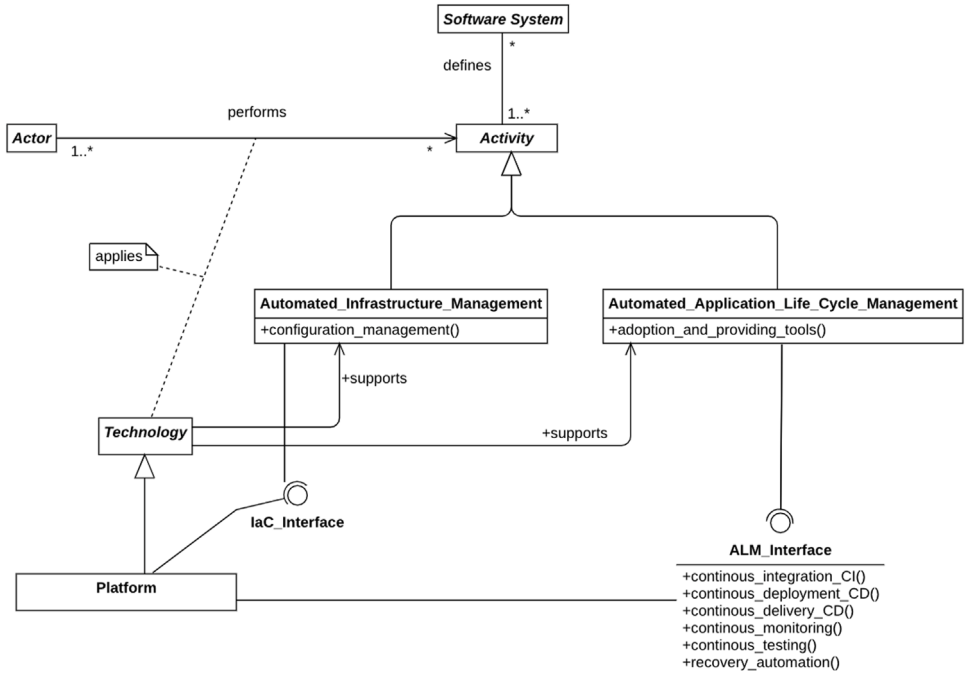

Fig. 9. Constructs of Technology abstract class, Activity class, and its subclasses.

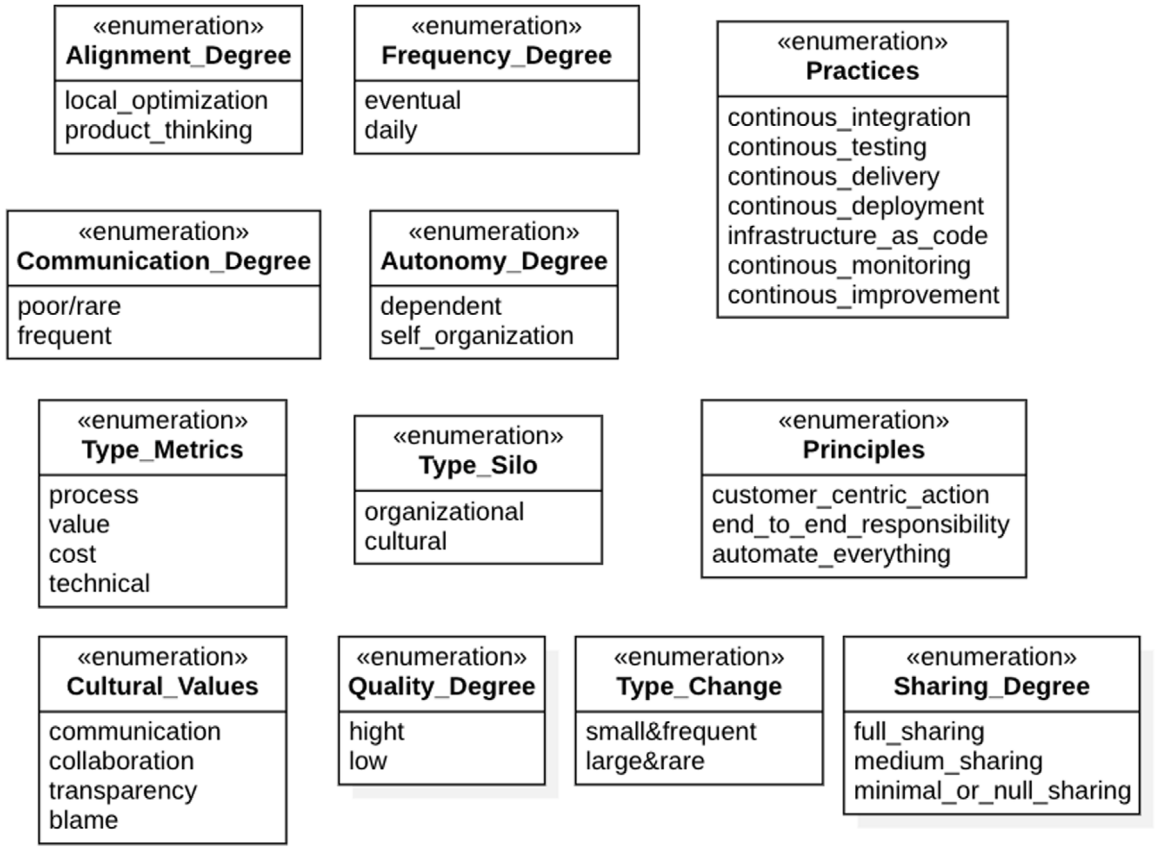

Fig. 10. Enumerations for Figs. 7, 8, and 9.

Project_Management is responsible for managing a temporary endeavor of that product lifecycle (e.g., development). In our theory, note that *Management* has the attribute *change_type*, which represents the frequency of changes. The following excerpts support this explanation [9:15] *"change is necessary in order to improve. Without that, there is not much room for maneuvering"*. [9:17] *"Change management is best pursued as small, continuous actions, the majority of which are ideally both automatically tested and applied"*.

E3 - Automation - Automation is an essential DevOps activity. It is then presented in the diagram in several ways. On the one hand,

Automated_Application_Life_Cycle_Management, for example, refers to the automation of some of the application life cycle processes (from development and deployment pipelines to monitoring tasks) and the adoption or providing tools for supporting these processes. For example: build automation (continuous integration, CI); testing automation (continuous testing or quality assurance automation); delivery automation (continuous delivery, CD); deployment automation (continuous deployment, CD); operational tasks automation (continuous measurement/feedback/monitoring from operations to development); and recovery automation.

On the other hand, $Automated\_Infrastructure\_Management$ refers to the automation of infrastructure management and configuration management tools.

E4 - Platform Servicing - $Platform$ provides automated CI/CD pipelines and managing infrastructure physical and/or virtual environments, containerization, Infrastructure as Code (IaC), pipeline automation (CI/CD and release tools), and IT operation. This platform can be provided as a service (see $Platform\_Servicing$ in Appendix), which represents a service automation that implements DevOps best practices. The specializations of this interface are $ALM\_Interface$ (Automation Life Cycle Management) and $IaC\_Interface$ (Infrastructure as a Code). The first one (see Proposition 26) implements continuous integration, continuous testing, continuous delivery and deployment, recovery automation, and continuous monitoring. The second one (Proposition 27) implements the managing and provisioning of infrastructure through code instead of through manual processes, this means, configuration files containing infrastructure specifications are created, making it easier to edit, distribute, and update configurations. **E5 - Communication** - $Communication$ in Fig. 7 is directly related to the $Team$ class. The objective is to inform that the team has relationships with another team. It has the attribute $type$, ranging from poor/rare to frequent. An example of frequent communication is [2:13] *"Development and infrastructure teams participate in the same chat; it even looks like everyone is part of the same team"*. Some initiatives aim to achieve a greater frequency of communication: [5:10] *"training for Devs and Ops on the responsibilities of other departments can be very beneficial for communication"* (see Proposition 13). However, we also identified that [2:6] *"Limited DevOps initiatives, centered on adopting tools, do not improve communication"*, this occurs mainly in cases where companies focus on the short-term benefits of DevOps with automation and tool adoption, but not giving the same value to cultural values.

E6 - Collaboration - $Collaboration$ in Fig. 7 is directly related to the $Team$ class. It has two attributes: $frequency$ that goes from eventual to daily, and $quality$ that goes from low to high. Collaboration is considered an essential aspect of specifying the structure of teams as shown in [1:28] *"collaboration frequency is highly related to the team structures and is a critical variable for DevOps adoption. Indeed, Collaboration is one of the key values of DevOps culture"*.

In some cases, members work together daily. This implies frequent collaboration between team members and usually a daily meeting in addition to other less frequent meetings with related teams. Promoting Collaboration reduces organizational silos/conflicts (see Proposition 2) as shown in [7:1] *"A collaborative culture essentially aims to remove the silos between development and operations teams and activities"*. However, members of product teams in other organizations have more differentiated roles (Dev versus Ops) so they work together but on different tasks. This means there is not a real collaboration but a transfer of responsibilities as shown in [4:10] *"Here, developers are not responsible for application deployment and management. Completed applications or features are handed over to the DevOps teams for deployment and management* [...]".

E7 - Culture - For DevOps, establishing a culture is as crucial as defining automation tools and practices. Communication, collaboration, transparency, and blame represent team cultural values. Some teams benefit more from these factors, which is why they appear in Appendix at times as attributes and at other times as classes. Customer-centric action, end-to-end responsibility, and automate everything represent DevOps principles. Reaching cultural maturity is built slowly together with the team. [12:1] *"Trust, the cultural core of DevOps and microservices, needs to be cultivated across software teams"*.

5. Validity of the theory

The last step of the theory-building process involves the examination of the validity of the theory. To this end, we analyzed the following elements (Sjøberg et al., 2008):

1. The extent to which the theory, once it is instantiated, represents the theories that were the input for this study.
2. The clarity and precision of the elements that are part of the theory.
3. The extent to which a theory has been validated.
4. The scope of the theory must be explicitly and clearly specified.

We also checked whether the harmonization of the theory led to the loss of concepts regarding specific domains, which might be critical to deriving the hypotheses. This phenomenon arises when, in the "sorting procedure", we favor codes with co-occurrence in multiple documents. Concepts/properties appearing in only one paper will likely be discarded in the sorting procedure and not be part of the final theory.

5.1. Representativeness of the theory

DevOps taxonomy is a new research topic, and we believe we used all the relevant taxonomies proposed in both white and grey literature (complete or partial) in this paper to propose a harmonized taxonomy. Therefore, we instantiate some of the works used in this research to validate the proposed theory. The goal is to verify that the main concepts were covered in our theory and to map the properties of the proposed methodology.

Researchers Jorge Pérez (R2), Carla Rocha (R4), Paulo Meirelles (R5), and Leonardo Leite (R6) instantiated a sample of DevOps taxonomies. In addition, they were asked to highlight the gaps present in the theory and identify potential improvements in the theory. The complete dataset and the analysis of this procedure are available in the public repository.

We instantiated documents D1, D2* (latest research update), and D7. We chose D1 and D2* because they present four DevOps Team structures and their properties. We chose the latest research updates from document D2, which we labeled as D2* (Leite et al., 2022). Finally, document D7 presents only one team structure. We aimed to test whether our theory is capable of representing each of these team structures.

D1 instantiation

López-Fernández et al. (2021) identified four team structures: (A) Interdepartmental Dev & Ops collaboration, (B) Interdepartmental DevOps team, (C) Boosted (cross-functional) DevOps team, and (D) Full (cross-functional) DevOps team; and six properties that characterize these structures: leadership from management, shared ownership, collaboration frequency, organizational silos, cultural silos, and autonomy. First, we performed a mapping of these properties with the constructs and propositions of our theory shown in Appendix as follows:

- The *leadership from management* and *shared ownership* properties map to the attribute $responsibility/ownership\_sharing$ of the $Team$ class.
- *Collaboration frequency* maps to the attribute $frequency$ of $Collaboration$.

- The *organizational silo* and *cultural silo* properties map to the $Silo$ association class. Its attribute $type$ indicates whether it is organizational or cultural.
- *Autonomy* maps to the $autonomy$ attribute of the $Team$ class.

Finally, we instantiated the four structures (A-D) using the constructs and propositions of our theory shown in Appendix as follows:

- (A) *Interdepartmental Dev&Ops collaboration* can be instantiated with the $Development\_Team$ and $Operation\_Team$ classes together with the $transfer\_of\_work$ relationship since the conditions expressed in the note associated with this relationship are true. Some other conditions are $autonomy = dependent$ and there are organizational and cultural silos (i.e., the relationship is instantiated twice, one with $type = organizational$ and another with $type = cultural$).
- (B) *Interdepartmental Dev-Ops team* can be instantiated with the $Development\_Team$ and $Operation\_Team$ classes with no organizational silos, although cultural silos are maintained (i.e., the relationship is instantiated with $type = cultural$). There is no alignment between Dev and Ops department goals ($alignment\_of\_dev\&ops\_goals = local\_optimization$). There is frequent collaboration ($Collaboration.frequency = daily$), beyond the transfer of work, and some autonomy and shared ownership emerge.
- (C) *Boosted (cross-functional) DevOps team* can be instantiated with the $Product\_Team$ class, there are no silos, and there is an increasing shared responsibility and ownership ($responsibility/ownership\_sharing = medium\_sharing$).
- (D) *Full (cross-functional) DevOps team* can be instantiated with the $Product\_Team$, there are no silos, there is full shared responsibility and ownership ($responsibility/ownership\_sharing = full\_sharing$) and self-organization ($autonomy = self\_organization$). Product teams are provided with platform servicing through the instantiation of $Enabler\_Team$, which works as a DevOps center of excellence, and the $Platform\_servicing$ interface.

D2 instantiation*

Leite et al. (2021) identified four common organizational structures: (1) Siloed Departments, (2) Collaborating Departments, (3) Cross-functional, and (4) Platform Teams. These structures were identified alongside their core properties (characteristics found in corporations with a given structure) and supplementary properties (used to refine the explanation). Meanwhile, the core properties also were used to discuss delivery performance. Since this work focuses on describing the organizational structure, we assume delivery performance as an output of DevOps adoption, related to a team structure. Therefore, we only consider the core properties that describe the structures.

We performed the instantiation of the structures 1-4 based on the constructs and propositions of our theory shown in Appendix as follows:

- (1) *Siloed departments* identified by 7 core properties: (1) *"Developers and operators have well-defined and differentiated roles"*, which can be instantiated by the $Team$ class with $role\_definition/attributions = true$; (2) *"Each department is guided by its own interests"* instantiated by $Team$ with $blame = true$ and $Collaboration$ with $quality = low$; (3) *"Developers have minimal awareness of what happens in production"* represented by $Team$ with $responsibility/ownership\_sharing = minimal$ or $nullsharing$; (4) *"Developers often neglect non-functional requirements (NFRs)"* instantiated by $Team$ with $responsibility/ownership\_sharing = minimal$ or $nullsharing$; (5) *"Limited DevOps initiatives"* represented by *Automated_Application_Life_Cycle_Management* and $Team$ with $alignement\_of\_dev\&ops\_goals = minimal$ or $null$ sharing; (6) *"Organizations are less likely to achieve high delivery performance"*, in this case our theory does not represent delivery performance; (7) *"lack of proper test automation"* instantiated by $Automated\_Application\_Life\_Cycle\_Managements$ implements $ALM\_Interface$ without $continuous\_testing()$.
- (2) *Collaborating departments* identified by 9 core properties: (1) *"a culture of collaboration"* instantiated by $Team$ with $responsibility/ownership\_sharing$ and $stack\&tools\_sharing = full\_sharing$ and $Collaboration$ with $frequency = daily$; (2) *"stress can persist at high levels for the infrastructure team"* instantiated by $Collaboration$ with $quality = low$; (3) *"alignment of different departments"* instantiated by $Team$ with $alignement\_of\_dev\&ops\_goals = product\_thinking$; (4) *"Development and infrastructure teams share NFR responsibilities"* instantiated by $Team$ with $responsibility/ownership\_sharing = fullsharing$; (5) *"the infrastructure staff is the front line of tracking monitoring and incident handling"* instantiated by $Collaboration$ with $quality = low$; (6) *"rebranded their infrastructure teams as DevOps or SRE"* the so called "DevOps Team" usually is simply $Operation\_team$, and not $Bridge\_team$; (7) *"Culture of collaboration prescind from a "DevOps team"* instantiated by $Team$ with the $Collaboration.type = daily$ and $Bridge\_team$ not instantiated; (8) *"Not enough to achieve high delivery performance"*, in this case our theory does not represent delivery performance; (9) *"infra as development collaborator"* instantiated by $Team$ with $responsibility/ownership_sharing = full\_sharing$.
- (3) *Single department* identified by 7 core properties: (1) *"Independence among teams may lead to misalignment"* instantiated by $Team$ with $autonomy\_degree = self\_organization$ and $Collaboration$ with $quality = low$; (2) *"It is hard to ensure a team has all the necessary skills"* instantiated by $Team$ with $autonomy\_degree = self\_organization$ and $Collaboration.quality = low$; (3) *"no evidence of idleness for specialists"*, our theory did not fit this; (4) *"Most of the cross-functional teams we interviewed were in small organizations"*, in this case our theory does not represent organization size; (5) *"Dedicated infra professionals"* instantiated by $Team$ with $cross-functionality/skills = true$ and $role_definition/attributions = true$; (6) *"Developers with infra background"* instantiated by $Team$ with $cross-functionality/skills = true$ and $role\_definition/attributions = false$; (7) *"Lightweight infra effort"* instantiated by $Team$ with $cross-functionality/skills = false$ and $role\_definition/attributions = false$.
- (4) *Platform teams* Identified by 9 core properties: (1) *"Product teams are fully accountable for the non-functional requirements of their services"* instantiated by $Product\_Team$ with $access\_to\_all\_necessary\_information()$; (2) *"The platform itself handles many NFR concerns"* instantiated by $Platform$ with $continuous\_monitoring()$ and $recovery\_automation()$; (3) *"Product teams become decoupled from the members of the platform team"* instantiated by the $Collaboration$ between the $Product\_Team$ and the $Bridge\_Team$ (which $automates\_infrastructure\_creation()$, $automates\_infrastructure\_deployment()$, and $automates\_infrastructure\_management()$) are eventually requested; (4) *"The infrastructure team is no longer requested for operational tasks"* instantiated by the $Bridge\_Team$ with $automates\_infrastructure\_creation()$, $automates\_infrastructure\_deployment()$, and $automates\_infrastructure\_management()$; (5) *"avoids the need for product teams to have infrastructure specialists"* instantiated by the $Product\_Team$ with $crossfunctionality/skills = false$; (6) *"The platform may not be enough to deal with particular requirements"* instantiated by the $Bridge\_Team$ with $automates\_infrastructure\_creation()$, $automates\_infrastructure\_deployment()$, and $automates\_infrastructure\_management()$; (7)*"it will require development skills from the infrastructure team"* instantiated by the $Bridge\_Team$ with $cross-functionality/skills = true$; (8) *"[organizations with] platform team structure are high performers"*, in this case, our theory did not fit this; (9)*"platform teams not suitable for small companies"*, our theory does not represent organization size.

D7 instantiation

Luz et al. (2019) built a theory with the workflow for DevOps adoption. They proposed a model for the successful adoption of DevOps adoption in companies. The scope of their theory relates to enterprise systems (in particular those based on a service-oriented architecture), and the only organization team structure used as reference is the Interdepartmental Dev & Ops collaboration.

We performed a mapping of these properties to the constructs and propositions of our theory shown in Appendix as follows:

- *Interdepartmental Dev & Ops collaboration* Identified by 11 properties: (1) *"Development Team"* instantiated by the $Development\_Team$ class; (2) *"Production Team (actor)"* instantiated by the $Operation\_Team$ class; (3) *"DevOps Collaborative Culture (Technology)"* instantiated by the $Development\_team$, $Operation\_team$, and $Collaboration$ classes with $frenquency = daily$ and $quality = high$; (4) *"Enterprise Systems Service Oriented Systems (Software System)"* is out of the scope of our theory; (5) *"Continuous measurements (Activity)"* instantiated by *Automated_Application_Life_Cycle_Management* implements $ALM\_Interface$ with *continuous_monitoring()* and *Management.get_set_metrics(technical)*; (6) *"Automation (Enabler)"* instantiated by $Automated\_Application\_Life\_Cycle\_Manegements$ implements $ALM\_Interface$ and the $Automated\_Infraestructure\_management$ class; (7) *"Transparency and Sharing (Enabler)"* instantiated by $Team$ with $skills/knowledge\_sharing = full\_sharing$; (8) *"Continuous Measurements (Enabler)"* instantiated by *Automated_Application_Life_Cycle_Manegements*, which implements $ALM\_Interface$ with $continuous\_monitoring()$; (9) *"Quality assurance (Enabler/outcome)"* not instantiated, because in our theory, quality assurance is an outcome; (10) *"segregation of development and operations teams in a rigid silo structure"* instantiated by the $Silo$ class with $type = Organizational$; (11) *"collaborative culture, removing silos and implementing a more direct communication between the teams"* instantiated by $Silo$ with $type = Cultural$.

In general terms, our theory was able to instantiate most (> 90%) of the constructs and relationships that the tested taxonomies define.

5.2. Clarity and precision

The constructs and propositions of a theory should be clear and precise so that they are understandable, internally consistent, and free from ambiguities. In our case, the definitions and descriptions of both constructs and propositions have been expressed in UML and, in the case of the propositions, also in OCL. The semantics of each of the elements that appear in the UML/OCL diagrams are described in their respective specifications (Object Management Group, 2017) and Object Management Group (2014) (see also Rumbaugh et al. (2004)). Due to the formal language applied, there is no room for ambiguity or inconsistency, problems that would have been detected by the tool used to draw the diagrams (starUML (MKLab, 2021)). It is worth mentioning that the semantics of some of the operations/attributes defined in some of the classifiers might be misleading. But, if we would like to clarify them, this information would be artificially added from the researchers' knowledge, since it is not reflected in the documentation analyzed. In this sense, we limited ourselves to defining and characterizing the elements that emerge in the theory only based on the data extracted from the primary papers, trying not to include any extra knowledge.

5.3. Extent to which a theory has been validated.

According to Sjøberg et al. (2008), we must differentiate between two terms: scope of interest and scope of validity of a theory. In our case, the scope of interest was explained in . According to Sjøberg et al. (2008), "The theory's scope of validity refers to that part of the scope of interest in which the theory has actually been validated. The scope of validity of a theory is the accumulated scopes of validity of the results of the studies that have tested the theory, or the studies from which the theory has been generated". In our case, the scope of validity is the 10 primary papers we used to generate the theory.

5.4. The scope of the theory

In general, this concept refers to the fact that conditions must be explicitly and clearly specified, so that the domain or situations in which the theory should be (dis)confirmed and applied are clear. In our case, the scope was set in and graphically depicted in Fig. 6. Roughly speaking, the theory can be applied to organizations that adopt DevOps culture and structure their teams to develop and operationalize software systems.

6. Discussion and implications

This section converges the harmonized theory's construction, its alignment with the applied methodology, and the theoretical sensitivity inherent to DevOps' multifaceted nature.

6.1. Harmonizing taxonomies

The harmonized theory substantially relies on the frequency of code occurrences across multiple documents and their co-occurrences to establish relationships. These form the bedrock for constructing the network and, ultimately, the taxonomy diagram or model.

In the "Select Core Categories" stage, refining and updating the codebook based on density analysis, code co-occurrences, and decisions from the disagreement diary are key activities. Codes with low statistical significance are eliminated from the analysis. For example, although the impact of containerization on continuous deployment is well-established, the code "containerization" appeared in only one document. It was deemed unrelated to direct DevOps team structures and subsequently removed from the codebook and the final theory.

Notably, properties or codes that appear with reasonable frequency and representative samples are included in the harmonized theory. This criterion signifies that even if divergent or competing theories arise within the same domain, they can be incorporated into the harmonized theory, fostering a comprehensive perspective. However, the "Select Core Categories" stage filters out unsound and poorly-defined properties, including those exclusive to specific domains.

6.2. Evaluating our emerging theory

We instantiated three documents to evaluate our theory's representativeness, resulting in a comprehensive assessment. Our model did not cover two properties that emerged during the instantiation of D2*: delivery performance and organization size. One was added to the core property of "Siloed Departments", while the other was included under "Collaborating Departments", highlighting the importance of quality attributes in structuring properties.

Similarly, the instantiation of D7 highlighted challenges stemming from stringent legal frameworks. This taxonomy diverged from ours due to its specific technical enablers. While our theory may not encapsulate all nuances, ongoing steps aim to reinforce the theory through practical implementation and rigorous testing.

6.3. Theoretical sensitivity

The multidimensional nature of DevOps, encompassing both technical and social aspects, introduces the potential for bias or missed concepts. In our study, diverse coders' expertise enhances theoretical sensitivity. The collaborative coding, ICA, and disagreement discussions help identify concepts requiring further exploration, reveal agreements, and signify theoretical saturation.

Engaging experienced researchers during validation broadens perspectives, revealing ambiguities and offering enhancements. This process uncovers aspects not initially considered and mitigates biases introduced by the leading authors. The interactive, collaborative, multistage methodology minimizes bias in theory construction. The meticulously structured methodology and tools like Atlas.ti for traceability provides a solid foundation for readers to assess and validate.

7. Threats to validity, reliability, and limitations

Criteria for judging the quality of research designs are crucial to establishing the validity, i.e., the accuracy of the findings, and the reliability, i.e., the consistency of the procedures and the researchers' approach (Yin, 2017; Creswell, 2003). We considered the quality criteria defined by Lincoln and Guba's (Lincoln and Guba, 1985) for qualitative research, which we now discuss

Credibility is also referred to as trustworthiness, i.e., the extent to which conclusions are supported by rich, multivocal evidence. The strategy to mitigate this threat was to define quality criteria to document inclusion in the research and consensus among researchers in each iteration. Also, we considered both academic and gray literature. The selection of documents was iterative and can be considered a combination of: (i) "Convenience sampling" as we were restricted to the little literature on the phenomenon to be studied. (ii) "Theoretical sampling", in the sense that we chose which data to collect based on the concepts and categories relevant to the emerging theory. (iii) Finally, "Maximum variation sampling", in the sense that we tried to choose highly diverse documents in our sample, strengthened our theory's transferability.

Aiming to select the most relevant data in gray literature (GL) from the practitioners perspective, we established the following criteria to select the GL database: excluding data sources whose purpose was to share the developments and information about a particular company; including data sources that present information and posts that present structures, properties, methods, and challenges in adopting DevOps.

Resonance is the extent to which the study's conclusions make sense to (i.e., resonate with) the community. To test the proposed diagram, we shared it with the author of one of the base works (not involved in this study's theory-building process) to instantiate and check whether the theories presented in D1, D2*, and D7 could be instantiated by our theory. The results and discussion were presented in Section 5.1.

Transferability shows whether the findings could plausibly apply to other situations. Data were iteratively gathered from a number of papers (10) that is large enough to build a complete picture of the phenomenon. This multiplicity is what provides the basis for theoretical generalization, according to which the results are extended to cases that have common characteristics and, hence, for which the findings are relevant (Wohlin et al., 2012). Furthermore, as mentioned, our theory is substantive in that the scope of validity refers to the ten primary studies processed. Like any GT study, the result is only applicable to the domain and context being studied, and therefore cannot be assumed to be applicable to other contexts, or in general. However, the theory developed could be transferable to evolutions of the DevOps phenomenon, such as DevSecOps, by simply incorporating the security aspect into the proposed taxonomy.

Usefulness refers to the extent to which a study provides actionable recommendations to researchers, practitioners, or educators and the degree to which results extend our cumulative knowledge. We chose a standard UML representation of the harmonized DevOps taxonomy to guarantee readability across the community. The theory we built is useful to better understand the DevOps phenomenon as well as to create a common language for the entire DevOps community.

Dependability implies that the research process is systematic, well documented, and can be traced. We recorded and made the entire chain of evidence available by executing the proposed methodology in a public repository. We versioned and documented the entire process of the theory building.

Conformability assesses whether the findings emerge from the data collected from cases and not from preconceptions. We have omitted any interpretation of the data analyzed, even if this would lead to ambiguities or vague interpretations (inductive approach). When appropriate, we used abductive reasoning, i.e., the premise is considered true and the most probable explanation is deduced (e.g. to explain some phenomena in the section "Providing Explanations to justify the Theory").

Finally, **qualitative reliability** indicates that the researchers' approach is consistent across different researchers and among different projects (Creswell, 2003). The benefits associated with conducting collaborative qualitative analysis have been extensively reported in Hall et al. (2005), Cornish et al. (2014), Richards and Hemphill (2018) and include: (i) juxtaposing and integrating multiple and diverse perspectives (Olson et al., 2016) (e.g., insider/outsider, academic/practitioner, senior/junior, interdisciplinary and international perspectives (Cornish et al., 2014)), which is often viewed as a way to counteract individual biases (Olson et al., 2016) and enhance/increase credibility (Olson et al., 2016), trustworthiness (Patton, 1999), and rigor (Dubé and Paré, 2003); (ii) addressing large and complex problems (Hall et al., 2005) by effective management of large datasets (Olson et al., 2016); and (iii) effective mentoring of junior researchers (Cornish et al., 2014). In Patton's words, "*Having two or more researchers independently analyze the same qualitative data set and then compare their findings provides an important check on selective perception and blind interpretive bias*" (Patton, 1999). However, collaborative qualitative analysis can also be challenging, and time-consuming (Hall et al., 2005).

Limitations. As is common in any research methodology, our chosen research methods have certain limitations. The initial limitation of our study pertains to the number of papers available. Presently, few works aim to elucidate DevOps team structures. It is important to note that our study's objective did not involve generalizing a phenomenon observed within a sample to an entire population. Instead, we construct a theory regarding a multifaceted phenomenon based on a collection of observations obtained through theoretical sampling. Grounded Theory does not lend itself to statistical generalization. Although the theory we propose holds broad relevance, the specific context of individual organizations may give rise to distinct perceptions and interpretations of the DevOps phenomenon, consequently influencing their perspective on the theory expounded within this paper.

8. Conclusion

Over the past few years, there has been particular interest in how organizations structure their teams to adopt the DevOps culture and practices. Hence, several authors have attempted to better understand the organizational structure and characteristics of teams adopting DevOps (Shahin et al., 2017; Nybom et al., 2016a; Macarthy and Bass, 2020; Luz et al., 2019; Puppet and CircleCI, 2021). In fact, the authors of this paper, who are now working together for the first time, previously conducted two independent GT studies (Leite et al., 2021; López-Fernández et al., 2021) and described several team structures and how their characteristics impact the software delivery performance. The novel work presented in this paper constructed a theory that analyzes and maps similarities among existing team taxonomies in

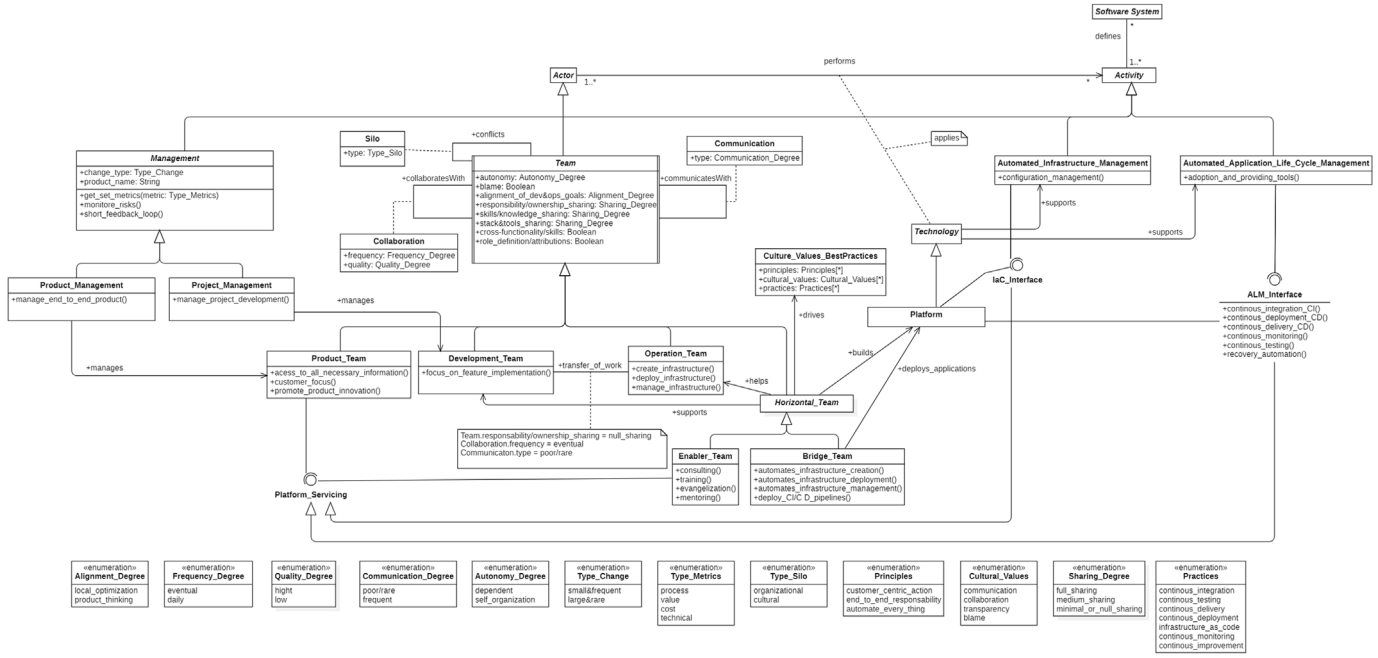

Fig. 11. Constructs and propositions.

the DevOps literature and proposes a model that harmonizes and explains these team structures and their common properties. This allows managers to focus on the key constructs extracted from all the analyzed taxonomies and define the structure best suited to an organization context, being aware of the relationships between these constructs.

This theory has been constructed in an international and collaborative context by using a strict GT process that includes quantitative measurements to improve the credibility, rigor, transparency, and systematicity of this qualitative research. Although the effort in collaborative coding was substantial (and time-consuming), we realized that it is the way for different researchers (with different profiles, contexts, training, culture, and theoretical sensitivity) to agree on a shared understanding of the phenomenon under analysis.

The theory developed together with the taxonomies we analyzed to develop it will allow us to carry out a reflection process. This process would aim at answering which elements have originated the differences in the existing taxonomies. The theoretical sensitivity of the researchers, the context in which the research is carried out, and the data collection and analysis methodologies are elements that influence the theory. Furthermore, this work serves as a basis for future research on developing a set of heuristics to help and guide companies to transition to the DevOps culture. As part of our future works, we aim to implement and assess our methodology in crafting theories pertaining to a different field, such as MLOps. This approach will allow for a comparative analysis of the resulting theories.

CRediT authorship contribution statement

Jessica Díaz: Conceptualization, Investigation, Methodology, Writing – original draft, Supervision, Visualization, Validation, Writing – reviewing. **Jorge Pérez:** Conceptualization, Investigation, Methodology, Writing – original draft, Supervision, Formal analysis. **Isaque Alves:** Conceptualization, Investigation, Methodology, Writing – original draft, Data curation, Visualization, Validation, Editing. **Fabio Kon:** Investigation, Data curation, Formal analysis, Visualization, Validation, Writing – review & editing. **Leonardo Leite:** Investigation, Data curation, Formal analysis, Visualization, Validation, Writing – review & editing. **Paulo Meirelles:** Investigation, Data curation, Formal analysis, Visualization, Validation, Writing – review & editing. **Carla Rocha:** Conceptualization, Investigation, Methodology, Writing – original draft, Supervision, Visualization, Validation, Editing, Writing – reviewing.

Declaration of competing interest

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

Data availability

All the data related to this research, providing a detailed chain of evidence, is available at https://github.com/jdiazfernandez/devops_taxonomies.github.io/tree/V1.0.0#readme. For assuring long-term archiving, the data is also stored in https://doi.org/10.5281/zenodo.6948397.

Acknowledgments

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq, Brazil proc. 465446/2014-0, CAPES, Brazil – Finance Code 001, and FAPESP, Brazil procs. 14/50937-1 and 15/24485-9. We also thank the support of FAPESP, Brazil proc. 19/12743-4.

Appendix. Constructs and propositions

Fig. 11 shows the full description of Constructs and propositions of the theory.

References

2020 DevOps Trends Survey, 2020. https://www.atlassian.com/whitepapers/devops-survey-2020. (Last accessed 01 July 2022).

Allspaw, J., Hammond, P., 2009. 10+ deploys per day: Dev and ops cooperation at flickr. In: Velocity: Web Performance and Operations Conference.

Armstrong, D., Gosling, A., Weinman, J., Marteau, T., 1997. The place of inter-rater reliability in qualitative research: An empirical study. Sociology 31 (3), 597–606. http://dx.doi.org/10.1177/0038038597031003015.

ATLAS.ti Scientific Software Development GmbH, 2019. https://atlasti.com/. (Last accessed 01 January 2020).

Campbell, J.L., Quincy, C., Osserman, J., Pedersen, O.K., 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. Sociol. Methods Res. 42 (3), 294–320. http://dx.doi.org/10.1177/0049124113500475.

Charmaz, K., 2014. Constructing Grounded Theory, second ed. Sage.

Cornish, F., Gillespie, A., Zittoun, T., 2014. Collaborative Analysis of Qualitative Data. SAGE Publications Ltd, pp. 79–93. http://dx.doi.org/10.4135/9781446282243.

Creswell, J., 2003. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 5ed. SAGE Publications.

Debois, P., 2008. Agile infrastructure and operations: how infra-gile are you? In: Agile 2008 Conference. IEEE, pp. 202–207.

Díaz, J., López-Fernández, D., Pérez, J., González-Prieto, Á., 2021. Why are many businesses instilling a DevOps culture into their organization? Empir. Softw. Eng. 26 (2), 1–50.

Díaz, J., Pérez, J., Gallardo, C., González-Prieto, Á., 2023. Applying inter-rater reliability and agreement in collaborative grounded theory studies in software engineering. J. Syst. Softw. 195, 111520. http://dx.doi.org/10.1016/j.jss.2022.111520, URL https://www.sciencedirect.com/science/article/pii/S0164121222001960.

Dubé, L., Paré, G., 2003. Rigor in information systems positivist case research: current practices, trends, and recommendations. MIS Q. 597–636.

Easterbrook, S., Singer, J., Storey, M.A., Damian, D., 2008. Selecting empirical methods for software engineering research. In: Guide to Advanced Empirical Software Engineering. Springer London, London, pp. 285–311. http://dx.doi.org/10.1007/978-1-84800-044-5_11.

Erich, F., Amrit, C., Daneva, M., 2014. A mapping study on cooperation between information system development and operations. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 277–280.

Gisev, N., Bell, J.S., Chen, T.F., 2013. Interrater agreement and interrater reliability: key concepts, approaches, and applications. Res. Soc. Adm. Pharm. 9 (3), 330–338.

Glaser, B., 1992. Emergence Vs Forcing: Basics of Grounded Theory Analysis. In: Emergence vs. forcing, Sociology Press.

Glaser, B., Strauss, A.L., 1967. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine de Gryter, New York.

González-Prieto, Á., Perez, J., Diaz, J., López-Fernández, D., 2023. Reliability in software engineering qualitative research through inter-coder agreement. J. Syst. Softw. 202, 111707. http://dx.doi.org/10.1016/j.jss.2023.111707, URL https://www.sciencedirect.com/science/article/pii/S0164121223001024.

Gregor, S., 2006. The nature of theory in information systems. MIS Q. 30 (3), 611–642.

Gruber, T.R., 1993. A translation approach to portable ontology specifications. Knowl. Acquis. 5 (2), 199–220. http://dx.doi.org/10.1006/knac.1993.1008.

Guest, G., MacQueen, K.M., 2008. Handbook for Team-Based Qualitative Research. AltaMira Press, Lanham, MD.

Hall, W.A., Long, B., Bermbach, N., Jordan, S., Patterson, K., 2005. Qualitative teamwork issues and strategies: Coordination through mutual adjustment. Qual. Health Res. 15 (3), 394–410. http://dx.doi.org/10.1177/1049732304272015.

Hayes, A.F., Krippendorff, K., 2007. Answering the call for a standard reliability measure for coding data. Commun. Methods Meas. 1 (1), 77–89. http://dx.doi.org/10.1080/19312450709336664.

Humble, J., Molesky, J., 2011. Why enterprises must adopt devops to enable continuous delivery. Cut. IT J. 24 (8), 6.

Iden, J., Tessem, B., Päivärinta, T., 2011. Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts. Inf. Softw. Technol. 53 (4), 394–406.

Krippendorff, K., 2004. Reliability in content analysis: Some common misconceptions and recommendations. Hum. Commun. Res. 30 (3), 411–433. http://dx.doi.org/10.1111/j.1468-2958.2004.tb0073.

Leite, L.A.F., Meirelles, P.R.M., Kon, F., 2022. A Grounded Theory of Organizational Structures for Development and Infrastructure Professionals in Software-Producing Organizations (Ph.D. thesis). Universidade de São Paulo, http://dx.doi.org/10.11606/T.45.2022.tde-28062022-132626.

Leite, L., Pinto, G., Kon, F., Meirelles, P., 2021. The organization of software teams in the quest for continuous delivery: A grounded theory approach. Inf. Softw. Technol. 139, 106672.

Leite, L., Rocha, C., Kon, F., Milojicic, D., Meirelles, P., 2019. A survey of DevOps concepts and challenges. ACM Comput. Surv. 52 (6), 1–35.

Lincoln, Y., Guba, E., 1985. Naturalistic Inquiry. SAGE Publications.

López-Fernández, D., Diaz, J., Garcia-Martin, J., Pérez, J., Gonzalez-Prieto, A., 2021. DevOps team structures: Characterization and implications. IEEE Trans. Softw. Eng.

Luz, W.P., Pinto, G., Bonifácio, R., 2019. Adopting DevOps in the real world: A theory, a model, and a case study. J. Syst. Softw. 157, 110384.

Lwakatare, L.E., Kuvaja, M., 2016. An exploratory study of DevOps - Extending the dimensions of DevOps with practices. In: Proc. the Eleventh International Conference on Software Engineering Advances. pp. 91–99.

Lynham, S.A., 2002. The general method of theory-building research in applied disciplines. Adv. Dev. Hum. Resour. 4 (3), 221–241.

Macarthy, R.W., Bass, J.M., 2020. An empirical taxonomy of DevOps in practice. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 221–228.

MacPhail, C., Khoza, N., Abler, L., Ranganathan, M., 2016. Process guidelines for establishing Intercoder Reliability in qualitative studies. Qual. Res. 16 (2), 198–212. http://dx.doi.org/10.1177/1468794115577012.

McDonald, N., Schoenebeck, S., Forte, A., 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI Practice. Proc. ACM Hum.-Comput. Interact. 3 (CSCW), http://dx.doi.org/10.1145/3359174.

MKLab, 2021. StarUML. URL http://staruml.io/.

Murphy, N.R., Fong-Jones, L., Beyer, B., Underwood, T., Nolan, L., Rensin, D., 0000. Site reliability engineering book, Chapter 1 - How SRE relates to DevOps, URL https://sre.google/workbook/how-sre-relates/.

Nybom, K., Smeds, J., Porres, I., 2016a. On the impact of mixing responsibilities between Devs and Ops. In: International Conference on Agile Software Development. Springer, pp. 131–143.

Nybom, K., Smeds, J., Porres, I., 2016b. On the impact of mixing responsibilities between Devs and Ops. In: Sharp, H., Hall, T. (Eds.), Agile Processes, in Software Engineering, and Extreme Programming. Springer International Publishing, Cham, pp. 131–143.

Object Management Group, 0000. Object Constraint Language (Version 2.4), URL https://www.omg.org/spec/OCL/2.4.

Object Management Group, 0000. Unified Modeling Language (Version 2.5), URL https://www.omg.org/spec/UML/.

O'Connor, C., Joffe, H., 2020. Intercoder reliability in qualitative research: Debates and practical guidelines. Int. J. Qual. Methods 19, http://dx.doi.org/10.1177/1609406919899220.

Olson, J., McAllister, C., Grinnell, L., Walters, K., Appunn, F., 2016. Applying constant comparative method with multiple investigators and inter-coder reliability. Qual. Rep. 21, 26–42. http://dx.doi.org/10.46743/2160-3715/2016.2447.

Paez, A., 2017. Grey literature: An important resource in systematic reviews. J. Evid.-Based Med. http://dx.doi.org/10.1111/jebm.12265.

Patton, M., 1999. Enhancing the quality and credibility of qualitative analysis. Health Serv. Res. 34 (5 Pt 2), 1189–1208.

Puppet, CircleCI, 0000. 2020 state of DevOps report, URL https://www2.circleci.com/2020-state-of-devops-report.html.

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

Ralph, P., et al., 2021. Empirical standards for software engineering research. URL arXiv:2010.03525v2 [cs.SE].

Richards, K.A.R., Hemphill, M.A., 2018. A practical guide to collaborative qualitative data analysis. J. Teach. Phys. Educ. 37 (2), 225–231. http://dx.doi.org/10.1123/jtpe.2017-0084.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L.E., Tiihonen, J., Männistö, T., 2016. DevOps adoption benefits and challenges in practice: A case study. In: Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., Mikkonen, T. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 590–597.

Rothstein, H., Sutton, A., Borenstein, M., 2005. Publication bias in meta-analysis: Prevention, assessment and adjustments. http://dx.doi.org/10.1002/0470870168.

Rumbaugh, J., Jacobson, I., Booch, G., 2004. Unified Modeling Language Reference Manual, second ed. Pearson Higher Education.

Saldaña, J., 2012. The Coding Manual for Qualitative Researchers. Sage publications.

Shahin, M., Babar, M.A., Zhu, L., 2016. 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. pp. 1–10.

Shahin, M., Zahedi, M., Babar, M.A., Zhu, L., 2017. 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. pp. 384–393.

Sjøberg, D.I., Dybå, T., Anda, B., Hannay, J.E., 2008. Building theories in software engineering. In: Guide to Advanced Empirical Software Engineering. http://dx.doi.org/10.1007/978-1-84800-044-5_12.

Stol, K.J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering. pp. 120–131. http://dx.doi.org/10.1145/2884781.2884833.

Weston, C., Gandell, T., Beauchamp, J., McAlpine, L., Wiseman, C., Beauchamp, C., 2001. Analyzing interview data: The development and evolution of a coding system. Qual. Sociol. 24 (3), 381–400. http://dx.doi.org/10.1023/A:1010690908200.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslen, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

Wolfswinkel, J.F., Furtmueller, E., Wilderom, C.P.M., 2013. Using grounded theory as a method for rigorously reviewing literature. Eur. J. Inf. Syst. 22 (1), 45–55. http://dx.doi.org/10.1057/ejis.2011.51.

Yin, R., 2017. Case Study Research and Applications: Design and Methods. In: Supplementary Textbook, SAGE Publications.

Zhou, X., Huang, H., Zhang, H., Huang, X., Shao, D., Zhong, C., 2022. A cross-company ethnographic study on software teams for DevOps and microservices: Organization, benefits, and issues. In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, IEEE, pp. 1–10.

Jessica Díaz (orcid.org/0000-0001-6738-9370) received her Ph.D. degree in Computer Science from UPM in 2012. Her research interests are focused on Empirical Software Engineering, Research methods in SE, Agile Software Development, DevOps, IoT, Cloud computing, Model-driven Development, and Software Product Lines. Currently she is researching about DevOps culture and practice in industry and contributing in the ACM SIGSOFT Empirical Standards.

Jorge E. Pérez (orcid.org/0000-0003-3349-6017) received the B.S. degree in computer science from the Universidad Carlos III de Madrid, Spain, in 1999, and the Ph.D. degree from the UPM, in 2004. His research interests include Software Architectures and meta-modeling, and qualitative research in Software Engineering, as well as Engineering Education, field in which he received several awards from the Rector of the UPM for educational innovation at the university.

Isaque Alves (orcid.org/0000-0002-7254-0378) is a Ph.D. candidate in Computer Science at the University of São Paulo (USP) in São Paulo, Brazil. His research interests include Software Engineering, Artificial Intelligence, and DevOps. Alves received his Bachelor's degree in Software Engineering from the University of Brasília (UnB). He is a researcher at the Free/Libre/Open Source Software Competence Center at the University of São Paulo's Institute of Mathematics and Statistics. Contact him at isaque.alves@usp.br.

Fabio Kon (orcid.org/0000-0003-3888-7340) received the bachelor's degree in music from the São Paulo State University, in 1992, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, in 2000. He is a full professor with the University of São Paulo (USP) and carries out research in software engineering, agile methods, distributed systems, data science, and smart cities. He is the coordinator of the Brazilian National S&T Institute on the Future Internet for Smart Cities (https://interscity.org).

Leonardo Leite (orcid.org/0000-0001-5762-5413) received the M.Sc. and Ph.D. degrees in computer science from the University of São Paulo (USP), in 2014 and 2022, respectively. Since 2014, he is a software developer with the Brazilian Federal Service for Data Processing (Serpro), in which he has successfully promoted DevOps practices, such as the adoption of automated tests, deployment pipelines, continuous delivery, and monitoring.

Paulo Meirelles (orcid.org/0000-0002-8923-2814) received the Ph.D. degree in computer science from the University of São Paulo (USP), in 2013. He is Assistant Professor of Computer Science at University of São Paulo (USP) and carries out research in Software Engineering, focusing on Free/Open Source Software, Agile Software Development, Static Code Analysis, and Mining Software Repositories.

Carla Rocha (orcid.org/0000-0003-3102-5166) is a professor at the University of Brasilia (UnB) in Brasília, Brazil. Her research interests include DevOps, project-oriented courses, human aspects of software development, Machine Learning, and free-software development. Rocha received her Doctoral Degree in Computer Science from Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM). She is a researcher at the Free/Libre/Open Source Software Competence Center at the University of São Paulo's Institute of Mathematics and Statistics. Contact her at caguiar@unb.br.