Refinement suggestions/changes

First round

3 attributes added.
2 new propositions.

1 - [ATTRIBUTE] Insert an attribute related to the team's cognitive load.

2 - [PROPOSITION P29] Platform Servicing reduce product team cognitive load

CHANGE:
Insert the enumeration **Load_Degree**. Add cognitive load attribute to team.

COMMENT: The proposition P11, P29 talks about the platform, but just about the platform service provided by horizontals (Enabler and Bridge). Now we can represent the other side. The relations between platform servicing and team.

CHANGE: [NEW PROPOSITION]
P29 - Platform Servicing reduces team cognitive load.

References:

S1 Paes: "La plataforma tiene que reducir la **carga cognitiva** de los equipos de producto que consumen la plataforma."

"...en teoría **los servicios de plataforma reducen la carga cognitiva** si están bien hechos, si están ayudando los equipos de producto."

"Como definición, un equipo de **plataforma** es un equipo que tiene un objetivo primordial reducir la carga cognitiva de otros equipos y ayudarles a ser más autónomos, acelerando así su entrega de valor."

Paes interview: "...falta pensar en qué automatizaciones necesitamos para reducir la carga cognitiva de los equipos. Y también, se necesita automatizar a nivel de gobernanza y de temas como seguridad, privacidad, regulaciones. Todo esto tiene mucho sentido automatizarlo en algún tipo de servicios de plataforma, pero falta mucho para esto porque es mucho más difícil que comprar una herramienta. Este es tu negocio interno, y tú tienes que tener tus expertos que colaboran con plataforma para automatizar, no todo, hay cosas que no se puede automatizar, pero hay mucha cosa que puede automatizar (controles financieros, controles de privacidad, etc.) porque esto es otro tipo de dependencia que suele haber. Los expertos en seguridad, *compliance*, legal, esto es también un cuello de botella. Entonces es importante promover la automatización y aumentar así el flujo del valor y no tener todas estas dependencias."

Paes interview: "Es curioso porque empezaron a hacer que cada equipo tenga end-to-end ownership y lo único que tenían que hacer de forma standard era usar los servicios cloud del amazon, nada más, no tenían plataforma ni nada. Entonces, ahí los equipos eran stream aligned ideal en el sentido de no depender de ningún otro equipo, pero sin embargo llegaron a una carga cognitiva muy alta. Entonces tenían que preocuparse con tanta cosa, que ya ni siquiera tenían tiempo para analizar las necesidades de los clientes del servicio. Simplemente estaban ocupados con la parte técnica. Por esto, un equipo stream aligned por sí mismo no pueden sobrevivir, en el sentido que la carga cognitiva es muy alta y por esto necesitan que los equipos de plataforma y *enabling* les ayuden a reducir la carga cognitiva, para que puedan abstraerse de hacer ciertas cosas."

S3: "So, you create a layer of abstraction. I'm not sure if this is a silo in the sense that there's not even access, you have to understand a bit of a level below. So, when I'm setting up my service here, I say I need so many instances, what's the type of database, what's my data retention plan, data replication, how much redundancy do I need. I have to have a minimum understanding of what this means in how the infrastructure will actually be provisioned, but I don't need to understand cloud configuration, virtual machine, network, data allocation, and none of that."

3 - [ATTRIBUTE] Insert the attributes motivation, duration, and Definition of Done to Collaboration.

CHANGE:
Add attributes to collaboration.
Adjust explanation (E6 - Collaboration)

Motivation: The reason or purpose for the collaboration ("why we are doing this"). This explains the objective behind the interaction between teams.
Duration: The estimated or expected length of the collaboration. This sets an anticipated timeframe for the collaborative effort.
Definition of Done: The agreed-upon endpoint or success criteria for the collaboration. It serves as a marker to indicate that the collaboration is complete and can be measured, for example, with a boolean value (true/false) when the goal is achieved. Examples include having an API defined or a test automated.

COMMENT:
We adjust the validation to Definition of Done to align with the current terminology.

Paes interview:

"Pregunta (Dani):

¿Qué más parámetros crees que se pueden modelar sobre la **colaboración**?

Respuesta (Manuel):

Pues **duración**, **motivo** y una **validación** que cuando termina la colaboración y cuanto más específico mejor (si puede ser un booleano mejor). Por ejemplo, vamos a colaborar para definir esta API, pues el punto de llegada es tenemos la API definida y los dos equipos están de acuerdo en que ese es el punto de llegada. Otro ejemplo, necesitamos automatizar algún test, entonces el punto de llegada es tener un test automatizado en el pipeline. Entonces, para la colaboración son estos puntos: la duración estimada o esperada (que luego se puede cambiar) y el punto de validación que indica que se terminó la colaboración. Bueno, y también en algún momento puede pasar que la colaboración no esté funcionando porque uno de los equipos o no tiene tiempo o conocimiento para colaborar en esta área. Y luego la **facilitación** también debería tener un motivo, ósea porque estamos haciendo esto, e igual que antes saber cuándo se habrá terminado porque también hay un riesgo en la facilitación (para los *enablers teams* en particular, pero también puede darse facilitación entre equipos, por ejemplo, un equipo de producto senior que está ayudando un equipo más junior haciendo una migración de microservicios). Entonces, el tipo de interacciones puede estar más definido y de manera más específica y permitir que cualquier tipo de equipo interactúe con otro."

4 - [PROPOSITION P30] - A high quality collaboration reduces dependency, increasing team autonomy.

COMMENT:

Collaboration is crucial, but it can be costly and should only be used where it clearly adds value. Avoid using it for routine tasks or to address unnecessary dependencies that could be resolved by improving existing structures or processes. A collaboration-based culture focuses on solving problems in the most effective way possible. Therefore, at the start of any collaborative effort, you must clearly define its motivation, expected duration, and a clear definition of "done." These variables will determine the ideal **collaboration frequency** and **collaboration quality**.

Collaborations are meant to solve specific problems and should not be used for everyday tasks. For example, teams collaborate to automate a process or resolve an issue. Once autonomy is gained, they can return to focusing on their tasks and improving their area of expertise.

CHANGE:
P30. A high quality collaboration reduces dependency, increasing team autonomy.

Paes interview:

"Sí, un contrato temporal digamos. La colaboración es importante, pero también tiene un coste alto, entonces vamos a usar la colaboración donde realmente tiene valor. No para un trabajo digamos más común en el que estamos colaborando porque hay una dependencia que no debería haber, una dependencia que deberíamos solucionar."

Paes interview:

"En el caso de UX, si hay un equipo centralizado de UX y muchos equipos dependen de este equipo centralizada, y esto es un problema (puede no serlo, pero normalmente lo es) porque dificulta el flujo de valor al cliente. Justo hace poco hable con una empresa que tienen ahí un problema porque hay un proceso que indica que el equipo de UX tiene que aprobar todo y eso les toma dos semanas y media para un cambio que ellos pueden hacer en dos días. Por eso **las interacciones sirven para reducir dependencias**, no para mantener interacciones con una determinada frecuencia (porque esto significa que tenemos una dependencia que a lo mejor no es útil a nivel de flujo), sino que muchas veces queremos colaborar para hacer una facilitación durante un tiempo y enseñar a un equipo, por ejemplo, a cómo hacer un buen diseño o cuales son las guías de diseño de la empresa, de forma que ese equipo pueda hacer, no el 100%, pero sí el 80% de las cosas de UX."

5 - [METHOD] - Insert customer focus on Horizontal Team

COMMENTS:
Since we have to add customer_focus() to horizontal teams, we can adjust the Product Team setting external as the focus.

CHANGE:
Add method + customer_focus(internal) to Horizontal Team
Set as external to the Product team customer focus.

REFERENCE:

Paes interview: "...en el modelo solo ponéis "*customer focus*" en los equipos de producto, pero los de plataforma y enabling también tienen que tener *customer focus*, la única diferencia es que el customer es interno y no externo. Os digo esto porque tiene que ver con la pregunta."

S3 shows how they works as a enabler(platform) team leader: "My primary objective is to increase the number of products and customers on our platform. Often, teams choose not to create this dependency and prefer to do everything internally to gain speed. The platform will always provide an overview of various customers and products, but it doesn't cover all use cases. Ultimately, I believe it's beneficial for the business to invest in this type of team; however, it's an exciting challenge to effectively sell the advantages of this platform and deliver value to customers."

6 - [CONNECTION] - Product management in enabler (platform) team

COMMENT:
So far, we only have enough information to add the connection. We'll take a question to the next test round to better establish a proposition and hypothesis that helps describe and validate.

CHANGE:
Add the connection (+ manage) from Product Management to the horizontal team.
Add to the explanation: Product management activities enable a horizontal team to create a valuable product (services and platform).

REFERENCE:

S1 Paes interview: "hace falta tener más ***product management*** **dentro de la plataforma** y es lo que a muchas empresas les falta. Los equipos de producto también muchas veces son equipos de desarrollo, aunque hacen más cosas y son *build-and-run teams*, muchas veces no tienen ownership de la parte de negocio y producto. Tienen un *backlog* que alguien define o su *product manager* define y el equipo ejecuta. Y aún falta ahí. Pero, el tema de cross-functional está bastante extendido entre los equipos de producto general y en plataforma no tanto."

7 - [EXPLANATION] - Adjust explanation (E3 - Automation)

COMMENT:

Automation also reduces team cognitive load. Therefore, beyond automating technical tasks, it is essential to automate more complex areas such as governance, security, privacy, and regulatory compliance. The core idea is that these processes, traditionally manual and managed by specialists (such as security or compliance teams), create dependencies and bottlenecks that slow teams down. By transforming these activities into automated platform services, teams can focus on their core work, such as delivering value to customers.

CHANGE:

Automation is an essential DevOps activity. Beyond its role in streamlining technical tasks, automation is instrumental in reducing the cognitive load on teams. This principle enables extending automation to complex domains such as governance, security, privacy, and regulatory compliance. These processes, traditionally manual and reliant on a specialized team, often introduce significant dependencies and bottlenecks. By transforming these activities into automated platform services, teams can redirect their focus toward their core objective: the delivery of customer value.

Paes interview: "...falta pensar en qué automatizaciones necesitamos para reducir la carga cognitiva de los equipos. Y también, se necesita automatizar a nivel de gobernanza y de temas como seguridad, privacidad, regulaciones. Todo esto tiene mucho sentido automatizarlo en algún tipo de servicios de plataforma, pero falta mucho para esto porque es mucho más difícil que comprar una herramienta. Este es tu negocio interno, y tú tienes que tener tus expertos que colaboran con plataforma para automatizar, no todo, hay cosas que no se puede automatizar, pero hay mucha cosa que puede automatizar (controles financieros, controles de privacidad, etc.) porque esto es otro tipo de dependencia que suele haber. Los expertos en seguridad, *compliance*, legal, esto es también un cuello de botella. Entonces es importante promover la automatización y aumentar así el flujo del valor y no tener todas estas dependencias."

Second round refinement suggestions

1 - [hypotheses] H2.2

Teams with eventual collaboration are associated with fewer organizational silos ->
Teams with eventual collaboration are associated with organizational silos.

A colaboração eventual ou a colaboração baixa não está associada à redução de silos, mas sim à existência de silos organizacionais.

COMMENTS: A colaboração eventual ou a colaboração baixa não está associada à redução de silos, mas sim à existência de silos organizacionais, logo fizemos ajustes na hipótese para validar na próxima rodada.

COMMENT: The hypotheses H2.2 Teams with eventual collaboration are associated with fewer organizational silos and H2.4 Teams with low-quality collaboration are associated with fewer organizational silos were refuted. However, they were refuted with the argument that the need for frequent collaboration to break the silo. This weakens the validity of the hypothesis in the context described, but this hypothesis can describe a team with low-quality collaboration associated with organizational silos.

CHANGE: change the propositions:

H2.2 Teams with eventual collaboration are associated with **fewer** organizational silos -> H2.2 Teams with **eventual collaboration** are associated with **organizational silos**

H2.4 Teams with low-quality collaboration are associated with **fewer** organizational silos -> H2.4 Teams with **low-quality collaboration** are associated with **organizational silos**

Third Round refinements

1 - [PROPOSITION] remove P11

A proposição P11 afirma que *"A existência de times de plataforma não leva à separação de responsabilidades, mas sim... torna o compartilhamento de propriedade possível"*. As entrevistas mostram o oposto: A existência de times de plataforma exige uma separação clara de responsabilidades para funcionar (Modelo *as-a-Service*), onde o Time de Plataforma é dono do serviço (infra) e o Time de Produto é dono do valor de negócio (produto). Tentar "compartilhar" tudo cria confusão ou dependência, não autonomia.

Igor (Banco do Brasil) - Refuta fortemente

É categórico ao afirmar que a Plataforma não compartilha o *ownership* dos produtos de negócio. Existe uma fronteira clara: a plataforma garante a engenharia, o time de produto garante o negócio.

O Time de Plataforma não se sente "dono" do produto bancário (ex: CDC), ele é dono da "performance de engenharia".

Trecho Chave:
"Qual é o produto? A plataforma tem um produto que é a plataforma. E realmente a plataforma não vai ter o senso de propriedade para os produtos do banco... O senso compartilhado por um produto igual o CDC ele ocorre para os atores que estão ali dentro da linha de produto do CDC. A plataforma realmente não entra nisso."

Manuel Pais (Co-autor de Team Topologies) - Refuta explicitamente

Ele corrige a premissa, afirmando que a responsabilidade sobre o produto **não deve** ser compartilhada com a plataforma. A responsabilidade deve ser inteiramente do time de produto; a plataforma apenas viabiliza isso reduzindo a carga cognitiva.

Se a responsabilidade for compartilhada, dilui-se a autonomia. A plataforma deve focar em seus serviços funcionarem, não no sucesso do produto em si.

Trecho Chave:
"...no debería ser responsabilidad compartida de producto y plataforma. La responsabilidad es de producto, pues son los dueños del producto... El equipo de plataforma está enfocado en que sus servicios funcionan bien, no a que tu producto funcione bien."

Alexandre (Google) - Refuta a dinâmica de "compartilhamento orgânico"

Ele descreve a relação como contratual (SLOs) e de fornecedor-cliente, o que implica uma separação de domínios e responsabilidades, e não uma fusão de *ownership*. Ele até alerta que a plataforma pode virar um "silo" se não entregar valor, reforçando a separação.

A plataforma provê capacidades, mas o valor final depende de quem usa (separação).

Trecho Chave:
"Of course, the platform itself doesn't deliver value; it can provide new capabilities, but customers need to configure or start using them." "As we're a platform team... we sign contracts with our customers, such as Service Level Objectives (SLO)..."

Conclusão:
Há um consenso forte de que não existe responsabilidade compartilhada sobre o produto final. Cada um é dono do seu "produto" (o time de produto é dono do negócio; o time de plataforma é dono da infraestrutura/serviço). A ideia de que o time de plataforma compartilha a responsabilidade pelo sucesso do produto de negócio (ex: "vender mais empréstimos") é rejeitada; o sucesso deles é "o deploy funcionou rápido".

EXPLANATIONS ADJUSTS

\subsubsection*{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~\cite{sjoberg:2008}. To that end, the following explanations are based on the constructs and propositions shown in Appendix~\ref{app:constructs_propositions}.

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 the 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_sharing or full_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 the product (see Proposition P12). When provided by a Platform Servicing, the product team can set a cognitive_load = median/normal. 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). Collaboration and responsibility sharing are properties of cross-functional teams (see Propositions P5 and P9, respectively), thus, product teams require having cross-functionality/skills = True, and this reduces silos (see Propositions P2, P10, and P19). Collaboration is crucial, but it can be costly and must be used when it clearly adds value. Product teams can establish a clear motivation, duration, and definition of done for a collaboration. Resulting in a Collaboration.frequency = daily and Collaboration.quality = median or high. 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] \textit{``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] \textit{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] \textit{``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 Teams are a specific structure that helps and supports the needs of multiple development or product teams within an IT department. With a customer_focus(internal) and Product_Management activities, they ensure that core operations are handled efficiently, delivering value and reducing the dev/product teams cognitive_load (see Proposition 29). 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 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).

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), consulting, training, evangelization, and mentoring. This 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. Since a single expert may temporarily perform enabling tasks, the team composition ranges from a single specialist to multiple specialists.

In contrast, 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 or provide a servicing. 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 Project_Management is responsible for managing a temporary endeavor of that product lifecycle (e.g., development). This specialization also enables a Horizontal_Team to create a valuable product, which includes services and platforms. 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] \textit{``change is necessary in order to improve. Without that, there is not much room for maneuvering''}. [9:17] \textit{``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. **Emerging from the team's collective experience, it serves as a mechanism for knowledge sharing by codifying best practices into reusable assets.** Beyond its role in streamlining technical tasks, automation is instrumental in reducing the cognitive load on teams. This principle enables extending automation to complex domains such as governance, security, privacy, and regulatory compliance. These processes, traditionally manual and reliant on a specialized team, often introduce significant dependencies and bottlenecks. By transforming these activities into automated platform services, teams can redirect their focus toward their core objective: the delivery of customer value.

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~\ref{app:constructs_propositions}), 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. Ultimately, this platform servicing reduces the cognitive load on teams (see Proposition 29). A platform's primary purpose is to reduce the cognitive_load, automating governance, security, and compliance, which are typically bottlenecks. By automating these processes, a platform reduces dependencies and increases the flow of value.

E5 - Communication

Communication in Figure~\ref{fig:team1} 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] \textit{``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] \textit{``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] \textit{``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 is directly related to the Team. Collaboration is crucial, but it can be costly and should only be used where it clearly adds value. Avoid using it for routine tasks or to address unnecessary dependencies that could be resolved by improving existing structures or processes. A

collaboration-based culture focuses on solving problems in the most effective way possible. Therefore, at the start of any collaborative effort, you must clearly define its motivation, expected duration, and a clear definition of "done." These variables will determine the collaboration frequency that goes from eventual to daily, and collaboration quality that goes from low to high. Collaboration is considered an essential aspect of specifying the structure of teams as shown in [1:28] \textit{``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] \textit{“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] \textit{“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~\ref{app:constructs_propositions} 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] \textit{``Trust, the cultural core of DevOps and microservices, needs to be cultivated across software teams.''}

Refinement questions

DevOps (bridge) team

CREATE PROPOSITIONS FOR THIS CONTEXT:

DevOps teams in these organisations understand the codes of developers and help out where necessary. Developers are also made aware of how the automated infrastructure works, though not directly involved in its creation or maintenance. According to some practitioners, a level of confidence brought about by the basic understanding of other aspects of the process and familiarity with the other actors. Intra-team collaboration is reported as brainstorming and coding together when issues are encountered. Collaboration in FinCo2 involves the DevOps team creating users’ stories from requirements, breaking them into manageable tasks and delegating these tasks to developers through Azure DevOps.
Some of the interviewees had the job title of ‘DevOps Engineer’ and worked in distinct DevOps teams or departments. “We don’t actually have developers in our team. So, in our case. . . it’s just DevOps” [Finco1 DOps1]. They further described their team as “platform builders” for developers, “who support them and host their

applications on our platform.” Here, we see DevOps being presented as a job description, with DevOps Engineers responsible for carrying out “DevOps functions”,

Fig. 2c depicts the Developers-DevOps mode where DevOps teams creates, deploys, and manages both the cloud infrastructure and deployment pipelines. Developers applications are also deployed and maintained by the DevOps team.

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, who are the DevOps practices facilitators.

DevOps bridge team mode was the mode widely used in our study. This mode (shown in Fig. 2d) was found in hybrid environment of cloud and onpremises deployment. Here, DevOps teams interface with both developers and IT Ops to drive the practices of DevOps like conffguration management, continuous integration and continuous delivery, automated testing, deployment, monitoring, and metrics collection.

Developers provide business solution, leaving the creation, deployment, and management of both the cloud infrastructure, virtual systems, and deployment pipelines to the DevOps teams. These teams are provided services of on-premises infrastructure by the Ops team. Essentially, the DevOps teams are customers to the Ops team, and service providers to developers. It is important to note that in this approach to DevOps, everyone is responsible for their actions.

Using automation tools, DevOps engineers create pipelines to enable continuous practices such as continuous integration/continuous deployment, continuous testing etc.

The DevOps teams under study are tasked with migration from existing platforms to either cloud based or an automated on�premises environment, and its subsequent maintenance. Generally, they act as intermediary between IT Ops and developers, providing the means to an end in software development (SD) by creating automated pipelines on both physical and virtualized servers to enable continuous integration and continuous delivery.

our ffndings show DevOps being described by practitioners in the study as not just a culture and speciffc job description, but also distinct teams separate from both developers and IT Ops teams. Although members of these teams have backgrounds in either software development or IT Ops, the nomenclature “DevOps” now separates them from their original silos and classiffes them as a unique team of “platform builders”

Bridge team: create a separate DevOps team that functions as a bridge between Devs and Ops

FROM THE DOCUMENT: **sorting.docx**

No documento do sorting, não tem nada que fala sobre o DevOps (bridge) team.

AUTOMATION brings from experience

Automation emerges from the capacity to identify and act on repetitive processes, a sensitivity cultivated through direct, hands-on experience. Experience from diverse backgrounds introduces novel solutions that enhance efficiency and preemptively address potential problems, thereby improving overall system quality and streamlining workflows.

Specialist 1: "Because automation comes from experience, right? People do it, there are many people who do something repeatedly without realizing that it is repeated. So the opportunity for automation comes from experience, right? I saw something like this somewhere else, I can do it here, I've already done something similar and in another scenario, I'll do it here. Or even the sensitivity of realizing that this is being very manual because it's taking a lot of time. And I realize that I'm repeating the code, I'm repeating commands, how can I automate? I think all of this is experience. Both life experience and diversity, right? Even if we're talking about people who automate things. And another person who automates in the backend, for example, in this team I'm on, there was a foreign collaborator, he was Hungarian, he created a hook in Git that for each commit, before finishing the commit he runs a Bash script, in Git you can do this. There's a page there about hooks where you can configure the hooks, what did he do? Did he create the hook to run lint before committing? So it's a file there that runs lint on the frontend and runs lint on the backend if it passes, if it doesn't pass, it prints the error there and then the developer has to go there, only commit if it passes lint, right, man, I think that's phenomenal. I didn't know, I didn't even know about this Git thing and I hadn't even had this idea of, Oh, you can put these lints inside, that is, automation in the local environment, that opened up a whole other bunch of things for me. Damn, I can use the local developer computer, because we have the local environment, there's a test environment. The repository and there's production, anyway, there are several moments that we can automate at various moments as soon as possible and that reminds me, right? Other theories. Other principles. That the sooner you identify the problem. That is, the faster the cycle, right, of making a change, identifying a problem and correcting it, the better, then damn. I thought, the best scenario is that everything that is local is faster. Everything I can do to avoid the person's location because if they only commit to lint in the repository, it's better to receive the email later. They're focused on something else, they've already missed the boat. Just giving the example of multidisciplinarity, how much his experience alone has already contributed to teaching me a lot of other things, and I think that's all I need to do."

Team cognitive load limits cross-functionality.

Due to the team's cognitive load, it's not advisable to have a fully cross-functional and 100% independent team. In such a setup, each member will end up focusing on their own area because they will be overloaded with tasks and improvements to make. This ultimately hinders the proper evolution of both the project and the team.

Paes interview: "Ellos dicen ok, para ser cross-functional tenemos en el equipo un diseñador, un frontend, un backend, un tester, PO, etc. y al final necesitas un equipo de 15 personas y no tienen responsabilidad compartida porque cada uno está en su silo dentro de equipo. Esa idea de cross functional puede llevar a este entendimiento que es un poco *naive*. Es mejor mantener un equipo pequeño e ir a buscar cross-funcionalidad con más responsabilidad compartida que es típicamente el modelo T shape (cada persona tiene una especialidad, pero pueden hacer varias cosas distintas)."