P1. A team culture based on **responsibility/ownership sharing enables collaboration**

```
Context Team inv:

   (self. responsibility/ownership_sharing <> Sharing_Degree::null_sharing)

       implies

   (self.Collaboration.type = Collaboration_Degree::daily)
```

[5:18] *"Mixing the responsibilities brought Devs and Ops closer to each other. Employees mentioned that Devs and Ops now collaborate on different tasks, since they now realize the importance of collaboration".*

responsibility/ownership sharing [enables] collaboration

P2. Collaboration promoting **reduces organizational silos/conflicts**

```
Context Team inv:

   (self.Collaboration.type = Collaboration_Degree::daily)

       implies

   (self.conflicts->isEmpty())
```

[7:1] "*A collaborative culture essentially aims to remove the silos between development and operations teams and activities*."

[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*".

[7:3] "*When a collaborative culture is fomented, teams collaborate to perform the tasks from the first day of software development. With the constant exercise of provisioning, management, configuration and deployment practices, software delivery becomes more natural, reducing delays and, consequently, the conflicts between teams*."

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

```
Context Bridge_Team inv:

   (self. Collaboration.type = Collaboration_Degree::daily)

       implies

   (self.deployment_application.ALM_Interface.ownedOperation->notEmpty())
```

[6:1] *"[...] The amount of collaboration between teams and team members, in particular application developers and operations team, increased after adopting CD."*

[6:11] *"6:11 Organizations are increasingly improving collaboration among teams and team members to effectively initiate and adopt CD practices"*

However, [2:21] *"Collaboration and delivery automation, critical values of the DevOps movement, are not enough to achieve high delivery performance."*

collaboration [associated with] automated application life-cycle management

P4. A team culture based on **knowledge sharing *enables* collaboration**

```
Context Team inv:

   (self. skills/knowledge_sharing <> Sharing_Degree::null_sharing)

       implies

   (self.Collaboration.type = Collaboration_Degree::daily)
```

[7:10] *"[...] transparency and sharing can be used to ensure collaboration"*

[7:15] *"Sharing concepts contribute with the collaborative culture. For example, all team members gain best insight about the entire software production process"*.

skills/knowledge_sharing [enables] collaboration

P5. Collaboration *is a property of* cross-functional teams

```
Context Cross_Team inv:

   self. Collaboration
```

[6:16-18] *"Collaboration and communications among team members can considerably increase by establishing cross-functional teams"*.

collaboration [is a property of] cross-functional teams

P6. Collaboration *is a property of* teams in which skills take precedence over roles, this is, the code **role definition/attributions**; hence, if there are already separate roles, responsibilities are very clear and collaboration is not fostered or promoted

```
Context Team inv:

   (self.Collaboration.type = Collaboration_Degree::daily)

       implies

   (self. role_definition/attributions = false)
```

[2:23] *"Infra as development collaborator. The infrastructure staff contributes to the application code to optimize the system's performance, reliability, stability, and availability. Although this aptitude requires advanced coding skills from infrastructure professionals, it is a suitable strategy for maintaining large-scale systems, like the ones owned by #I31."*.

collaboration [is a property of] role definition/attributions

P7. A **collaboration**-based culture ***requires* alignment of dev & ops goals**

```
Context Team inv:

   (self.Collaboration.type = Collaboration_Degree::daily)

       implies
   (self. alignment_of_dev&ops_goals = Alignment_Degree:: product_thinking)
```

[7:5] *“A collaborative culture requires product thinking, in substitution to operations or development thinking”.*

Collaboration [requires] alignment of dev & ops goals

P8. A team culture based on **metrics/visibility/feedback *enables* collaboration**

```
Context Management inv:

   ((self.allOperations())->exist (op| op.name = ‘get_set_metrics))

       implies

   (self.manages.Collaboration.type = Collaboration_Degree::daily)
```

[7:21] *“regularly performing the measurement and sharing activities contributes to avoiding existing silos and reinforces the collaborative culture […]”*

[6:19] “*we found that the lack of suitable awareness on status of project (e.g., build status, release status) among team members can be a bottleneck for collaborative work and significantly hinders the CD success*.”

metrics/visibility/feedback [enables] collaboration

P9. Responsibility/ownership sharing *is a property* of cross-functionality/skills

```
Context Team inv:

   (self.cross_functionality/skills = true)

       implies

   (self.responsibility/ownership_sharing <> Sharing_Degree::null_sharing)
```

[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."*

responsibility/ownership sharing [is a property of] cross-functionality/skills

P10. Responsibility/ownership sharing *reduces* organizational silos/conflicts. This confirms a transitive relationship as **responsibility/ownership sharing *enables* collaboration and collaboration** promoting ***reduces* organizational silos/conflicts**.

```
Context Team inv:

   (self.responsibility/ownership_sharing <> Sharing_Degree::null_sharing)

       implies

   (self.conflicts->isEmpty())



Context Team inv:

   ((self. responsibility/ownership_sharing = Sharing_Degree::full_sharing)
     implies
  (self.Collaboration.type = Collaboration_Degree::daily))
     and
  ((self.Collaboration.type = Collaboration_Degree::daily)
     implies
  (self.conflicts->isEmpty()))
```

[9:16] *"[…] An effective shared ownership model and partner team relationships are necessary for SRE to function. Like DevOps, SRE also has strong values shared across the organization, which can make climbing out of team-based silos slightly easier".*

responsibility/ownership sharing [reduces] organizational silos/conflicts

P11. Responsibility/ownership sharing *is a property* of organizational structures that rely on an **enabler (platform) team.** The existence of platform teams does not lead to a separation of responsibilities but rather they become facilitators and make ownership sharing possible, unlike **devops (bridge) teams** that become new silos with their own responsibilities (e.g., deployment, monitoring, etc.).

```
Context Enabler_Team inv:
   (self.responsibility/ownership_sharing <> Sharing_Degree::null_sharing)
Context Bridge_Team inv:
   (self.Silo.type = Silo::organizational
```

[2:33] *"Although the product team becomes fully responsible for NFRs of its services, it is not a significant burden that developers try to refuse (#I33). The platform itself handles many NFR concerns, such as load balancing, auto-scaling, throttling, and high-speed communications between data-centers [...] Moreover, we observed infrastructure people willingly supporting developers for the sake of services availability, performance, and security (#I9, #I14)."*

Responsibility/ownership sharing [is a property of] enabler (platform) team

P12. Responsibility/ownership sharing *is a property of* team self-organization & autonomy.

```
Context Team inv:
   (self. autonomy = Autonomy_Degree:: self_organization)
       implies
   (self.responsibility/ownership_sharing <> Sharing_Degree::null_sharing)
```

[8:8] *"Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software [...]".*

Responsibility/ownership sharing [is a property of] team self-organization & autonomy

P13. A team culture based on **responsibility/ownership sharing *enables* communication**

```
Context Team inv:

   (self. responsibility/ownership_sharing <> Sharing_Degree::null_sharing)

       implies

   (self.Communication.type = Communication_Degree::frequent)
```

[7:16] *"with a solid understanding of shared responsibilities. A shared vocabulary also emerged from sharing and this facilitates communication."*

responsibility/ownership sharing [enables] communication

P14. Responsibility/ownership sharing *is associated with the* transfer of work between teams. If there is no shared responsibility, there is necessarily a transfer of work between development to production and operation teams (and vice versa).

```
Context Team inv:

   (self. responsibility/ownership_sharing = Sharing_Degree::null_sharing)

       implies

   ((self.transfer_of_work()->notEmpty()) and

    (self.type=Development_Team or self.type=Operation_Team))
```

[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 [...]". "4:18 DevOps teams are responsible for all deployments. Developers handover applications to these teams, who oversee the journey through the CI/CD pipeline. The teams also monitor the applications and function as the first line of support."*

responsibility/ownership sharing [is associated with] transfer of work between teams

P15. Automated infrastructure management *enables* responsibility/ownership sharing

```
Context Cross_Team inv:

   (self.Platform.support.type = Automated_Infrastructure_Management)

       implies

   (self. responsibility/ownership_sharing <> Sharing_Degree::null_sharing)
```

[5:6] *"[...] Assuming that infrastructure is code, this statement suggests that Approach 1 (mix responsibilities) is a natural approach, because Ops will be involved in Dev tasks by developing the infrastructure together with the Devs."*

automated infrastructure management [enables] responsibility/ownership sharing

P16. Automated application life-cycle management *enables* responsibility/ownership sharing

```
Context Cross_Team inv:

                                    (self.Platform.support.type           =
Automated_Application_Life_Cycle_Management)
```

```
      implies

   (self. responsibility/ownership_sharing <> Sharing_Degree::null_sharing)
```

[6:15] *"[...] Our results reveal that the speed and frequency demanded by DevOps and CD practices drive the need for a more holistic view, in which team members from each side of the fence are needed to jointly work together and adopt shared responsibility as much as possible."*

automated application life-cycle management [enables] responsibility/ownership sharing

P17. Skills/knowledge sharing *is a property of* teams characterized by **cross-functionality/skills**

```
Context Cross_Team inv:

   self. skills/knowledge_sharing <> Sharing_Degree::null_sharing
```

[5:7] *"As for Approach 2 (mix personnel), it is stated in [12] that creating cross-functional teams is a good approach when adopting DevOps. These teams should consist of Devs, testers, Ops personnel and others, and then each of them would contribute code to a shared repository.**"***

Skills/knowledge sharing [*is a property of]* teams characterized by cross-functionality/skills

P18. Enabler (platform) team is a team characterized by cross-functionality/skills, then **cross-functionality/skills *is a property of* enabler (platform) team.** An enabler (platform) team is responsible for providing platform servicing and tools (mainly for infrastructure and deployment pipelines), consulting, training, evangelization, mentoring, human resources, etc to product teams. Therefore, at times providing a **platform service** will require development skills. Some companies apply this structure due to a lack of resources.

```
Context Enabler_Team inv:

   self. cross_functionality/skills= true
```

[2:39] *"If the organization develops a new platform to deal with its specificities, it will require development skills from the infrastructure team. Nevertheless, even without developing a new platform, the infrastructure team must have a ''dev mindset'' to produce scripts and use infrastructure-as-code [43] to automate the delivery path (#I14). One strategy we observed to meet this need was to hire previous developers for the infrastructure team (#I14)."*

[1:21] *"Figure 1 shows a cross team composed of highly qualified engineers in DevOps culture, specifically 5 senior developers, 10 testers and quality assurance engineers, and 10 IT operators (second-level operations), who get involved in product teams when necessary. These engineers are involved in product teams with exclusive dedication but limited in time, until product teams are capable of doing all their responsibilities, from planning, analysis, development, testing, deployment, to operation. This means that these horizontal teams are composed of engineers that move through the product teams according to their needs. The reason that these engineers are not part of the product teams is that these organizations (like ID2) do not have human resources enough to involve the necessary engineers in all the product teams."*

cross-functionality/skills [is a property of] enabler (platform) team

P19. Cross-functionality/skills reduces organizational silos/conflicts. Often, as in the following quotation, when dealing with organizational silos/conflicts, organizations reach a team characterized by cross-functionality/skills

```
Context Team inv:

   (self. cross-functionality/skills = true)

       implies

   (self.Silo.type <> Silo::organizational)
```

[1:11-12] *“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.”*

[7:1] *“A collaborative culture essentially aims to remove the silos between development and operations teams and activities. As a result, operations tasks—like deployment, infrastructure provisioning management, and monitoring— should be considered as regular, day-to-day, development activities. This leads to the first concept related to this core category: operations tasks should be performed by the development teams in a seamless way.”*

Cross-functionality/skills [reduces] organizational silos/conflicts

P20. Automated application life-cycle management *is a property of* teams characterized by **cross-functionality/skills.** A cross-functional team is usually responsible for the entire application lifecycle. Therefore, the lifecycle must be as automated as possible. With that, there is a need to learn and apply new practices and tools and understand more about the entire project stack

```
Context Cross_ Team inv:

   (self. cross_functionality/skills = true)

       implies

                                      (self.Platform.support.type            =
Automated_Application_Life_Cycle_Management)
```

[6:21] *“Interestingly most of the respondents indicated that they have to constantly learn best practices and new tools for reliable release [...]. Working in CD context necessitates developing monitoring skills and spending more time on monitoring to triage and quickly respond to production incidents. As stated by R20 “Ensuring the product stays deployment ready all the time. Each check-in and change gets monitored” [...]. In addition, for some of the respondents adopting CD means to understand the whole stack of the application: database, backend, front-end, OS, and build. This helped them to further and better be involved in bug fixing (e.g., “More in depth knowledge of the entire stack - to debug when something fails” R38)".*

automated application life-cycle management [is a property of] cross-functionality/skills

P21. Organizational silos/conflicts *make* the adoption of an automated application life-cycle management *difficult*. The lack of automation is visible in organizations with silos/conflicts

```
Context Team inv:

    if self.conflicts->notEmpty()

    then self.Platform. Automated_Infrastructure_Management.allOperations()

                         ->isEmpty()_
```

[2:10] *"We observed a lack of proper test automation in many organizations [...] Although siloed organizations are not the only ones that lack test automation (#I3, #I32, #I35), in this structure developers can even ignore its value (#I5, #I23, #I37). "* Furthermore, if automation is not correctly applied *"2:8 p 6 [...] a ''DevOps team'' maintaining the deployment pipeline behaves as another silo, sometimes bottlenecking the delivery." "6:7 [...] if organizations want to efficiently adopt and implement DevOps practices, in particular CD practices, they cannot really have operations silo (i.e., separate Ops team), even small one. Having operations silo may lead to a lot of frictions in deployment process and fail organizations to achieve the real anticipated benefits of CD practices."*

organizational silos/conflicts [make difficult] automated application life-cycle management *difficult*

P22. Metrics, visibility & feedback enables automated application life-cycle management. We noticed that when metrics, visibility & feedback are applied, it is possible to monitor risks and make decisions throughout the continuous delivery pipeline

```
Context Cross_Team inv:

   ((self.support.Project_Management.allOperations())->exist (op| op.name =

        'get_set_metrics'))

       implies

     (self.Platform. Automated_Application_Life_Cycle_Management.

        allOperations()->notEmpty())
```

[6:14] *"[...] A few number of the participants emphasized that having shorter feedback loop at each stage in CDP enables teams and team members to partner in producing high quality software."*

[7:21] *"[...] In the same way as continuous measurement, quality assurance is a category that can work both as enabler and as outcome. As enabler because increasing quality leads to more confidence between the teams, which in the end generates a virtuous cycle of collaboration. As an outcome, the principle is that it is not feasible to create a scenario of continuous delivery of software with no control regarding the quality of the products and its production processes [...]."*

metrics, visibility & feedback [enables] automated application life-cycle management

P23. Automated application life-cycle management enables skills/knowledge sharing

```
Context Cross_Team inv:
   (self.Platform. Automated_Application_Life_Cycle_Management.
    allOperations()->notEmpty())
       implies
   (self. skills/knowledge_sharing <> Sharing_Degree::null_shared
```

[7:10] *"In addition to contributing to transparency, automation is also considered important to ensure reproducibility of tasks, reducing rework and risk of human failure."*

automated application life-cycle management [enables] skills/knowledge sharing

P24. Enabler (platform) team *enables* team self-organization & autonomy

```
Context Enabler_Team inv:
   self.autonomy = Autonomy_Degree::self_organization
```

[2:40] *"All four organizations that have fully embraced the platform team structure are high performers, while no other structure provided such a level of success (Table 3). An explanation for such a relation is that this structure decouples the infrastructure and product teams, which prevents the infrastructure team from bottlenecking the delivery path. As stated by #I20: ''Now developers have autonomy for going from zero to production without having to wait for anyone''. This structure also contributes to service reliability by letting product teams handle non-functional requirements and incidents.""*

enabler (platform) team [enables] team self-organization & autonomy

P25. Enabler (platform) team *provides* platform servicing

```
Context Enabler_Team inv:
   self.Platform_Servicing
```

[1:18] *"[…] we also realized that there were different kinds of horizontal teams such as DevOps Center of Excellence (DevOps CoE), DevOps chapter and Platform team. Despite there are some differences among these teams, all of them refer to the same construct, i.e., teams that provide platform, infrastructure, IT operation".*

Enabler (platform) team [provides] platform servicing

P26. Automated application life-cycle management *is a* Platform servicing

```
Context ALM_Interface inv:
   self.oclIsKindOf (Platform_Servicing)
```

[2:32] *"Platform teams are infrastructure teams that provide highly automated infrastructure services that can be self-serviced by developers for application deployment. The infrastructure team is no longer a ''support team''; it behaves like a product team, with the ''platform'' as its product and developers as internal customers.".*

Automated application life-cycle management [is] *a* Platform servicing

P27. Automated infrastructure management *is a* Platform servicing

```
Context IaC inv:
```

`self.oclIsKindOf (Platform_Servicing)`
[7:17] *“The use of infrastructure as code was recurrently cited as a means for guaranteeing that everyone knows how the execution environment of an application is provided and managed”*
Automated infrastructure management [is] *a* Platform servicing
P28. Enabler (platform) teams provide automated application life-cycle management. This confirms a transitive relationship between enabler (platform) teams ***provides*** platform servicing and automated application life-cycle management **is a** platform servicing.

`Context Enabler_Team inv:`
`self.builds.ALM_Interfaz`

[2:32] *“Platform teams are infrastructure teams that provide highly automated infrastructure services that can be self-serviced by developers for application deployment. The infrastructure team is no longer a ‘‘support team’’; it behaves like a product team, with the ‘‘platform’’ as its product and developers as internal customers."* [9:10] *“an SRE team winds up automating all that it can for a service, leaving behind things that can’t be automated”.*
Enabler (platform) teams [provides] automated application life-cycle management