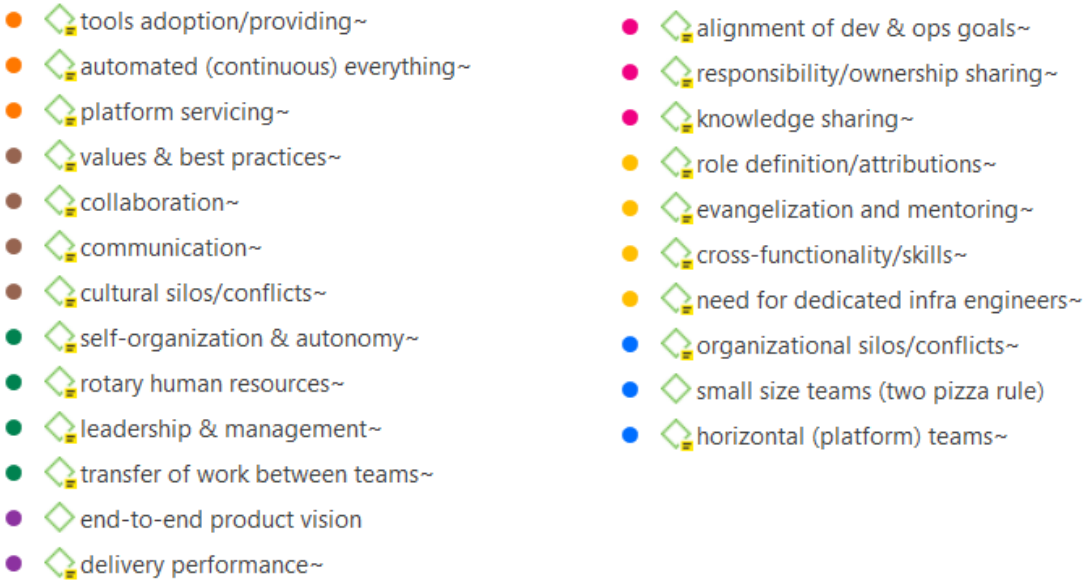
ITERATION 1

MASTER

- 2 documents: D1 and D2
- 71 quotations
- 7 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, MONITORING, SHARING, SKILLS & ROLES, STRUCTURE
- 23 codes



CODER1 (Jessica Diaz) - "UnifiedDevOps OpenCoding ITE1 (JD).atlproj9"

Changes: NA

CODER 2 (Isaque Alves) - "UnifiedDevOps OpenCoding ITE1 (IS).atlproj9"

Changes:

- 24 codes
  - added: system performance - Related to system quality, such as performance, reliability, stability, and availability.
- 71 quotations

ERRORS IN CODING

Quotation 2:1 has been deleted and added quotation 2:43

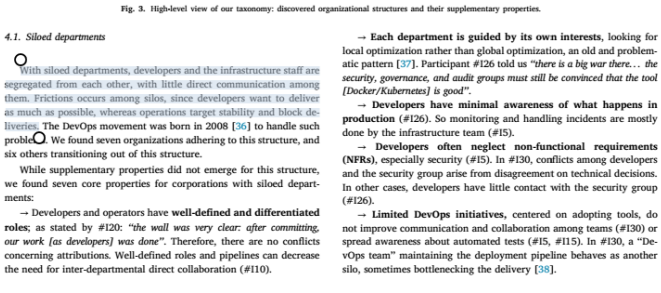


Fig. 3. High-level view of our taxonomy: discovered organizational structures and their supplementary properties.

4.1. Siloed departments

With siloed departments, developers and the infrastructure staff are segregated from each other, with little direct communication among them. Frictions occurs among silos, since developers want to deliver as much as possible, whereas operations target stability and block deliveries. The DevOps movement was born in 2008 [36] to handle such problem. We found seven organizations adhering to this structure, and six others transitioning out of this structure.

While supplementary properties did not emerge for this structure, we found seven core properties for corporations with siloed departments:

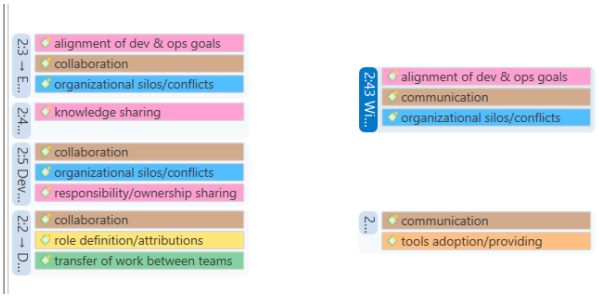
→ Developers and operators have **well-defined and differentiated roles**; as stated by #120: *"the wall was very clear: after committing, our work [as developers] was done"*. Therefore, there are no conflicts concerning attributions. Well-defined roles and pipelines can decrease the need for inter-departmental direct collaboration (#110).

→ **Each department is guided by its own interests**, looking for local optimization rather than global optimization, an old and problematic pattern [37]. Participant #126 told us *"there is a big war there... the security, governance, and audit groups must still be convinced that the tool [Docker/Kubernetes] is good"*.

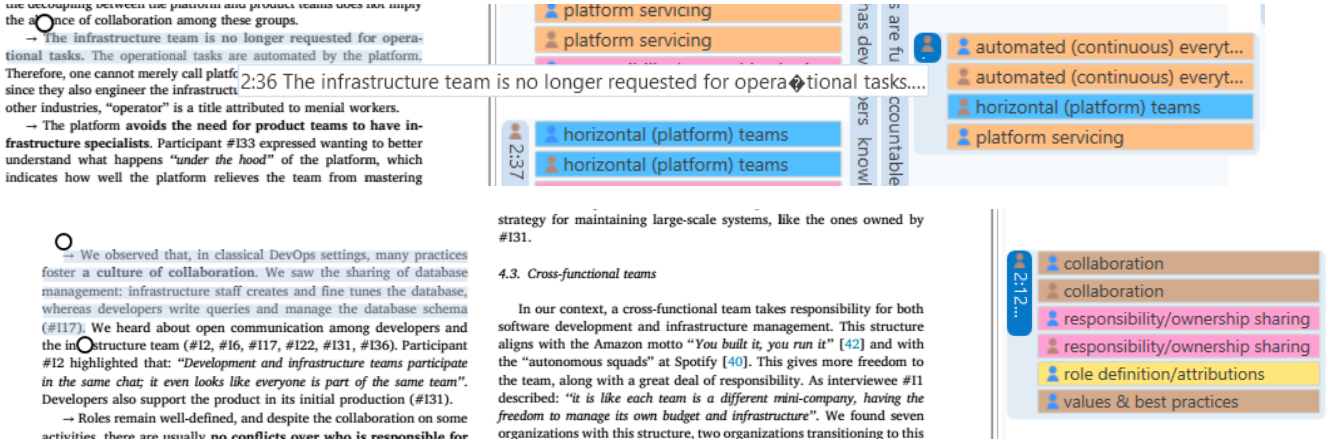
→ **Developers have minimal awareness of what happens in production** (#126). So monitoring and handling incidents are mostly done by the infrastructure team (#15).

→ **Developers often neglect non-functional requirements (NFRs)**, especially security (#15). In #130, conflicts among developers and the security group arise from disagreement on technical decisions. In other cases, developers have little contact with the security group (#126).

→ **Limited DevOps initiatives**, centered on adopting tools, do not improve communication and collaboration among teams (#130) or spread awareness about automated tests (#15, #115). In #130, a "DevOps team" maintaining the deployment pipeline behaves as another silo, sometimes bottlenecking the delivery [38].



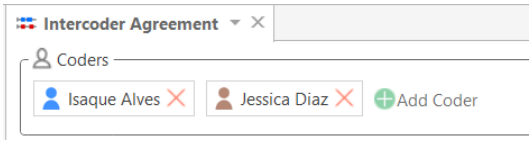
There isn't mutual exclusiveness in the following quotations:



ANALYSIS OF DISAGREEMENTS IN CODING

Krippendorf Cualpha = 0,789

See UnifiedDevOps OpenCoding ITE1 ICA (jd + ia) - alpha binary (PDF and XLSX formats)
See UnifiedDevOps OpenCoding ITE1 ICA (jd + ia) - cu alpha (PDF and XLSX formats)



1. There are disagreements between the two following codes (see Fig. 1):

- automated (continuous) everything
- tools adoption/providing

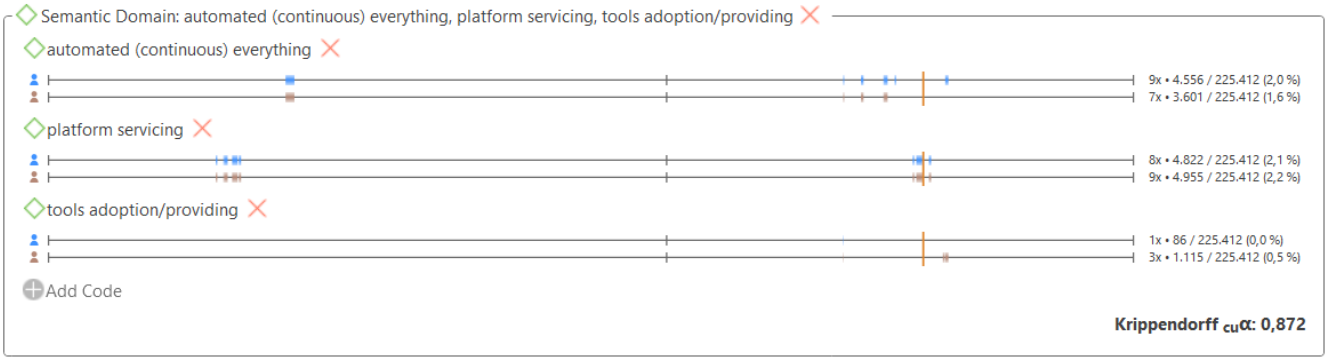


Fig. 1

Solution: merging of codes. The initial codebook is confusing as to the semantic boundaries of these concepts.

2. There is a disagreement in the quotation 2:3 (see Fig. 2):

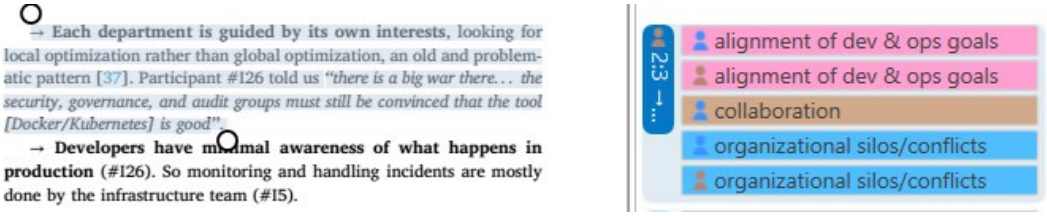


Fig. 2

Solution: NA

3. There is a disagreement in the quotation 2:9 (see Fig. 3):

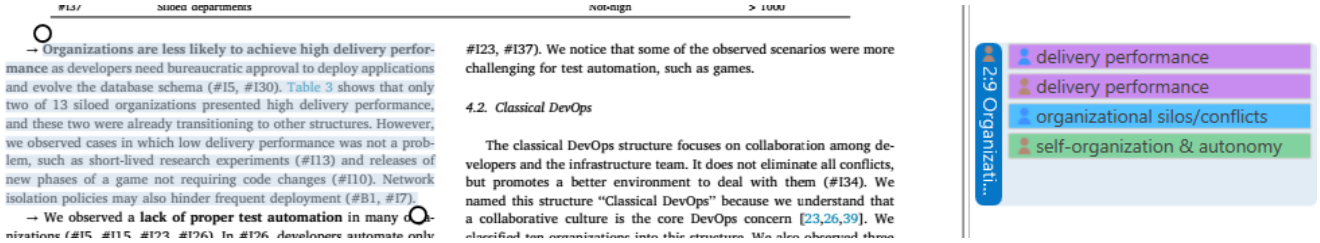


Fig. 3

Solution: NA

4. There is a disagreement in the quotation 2:15 (see Fig. 4):

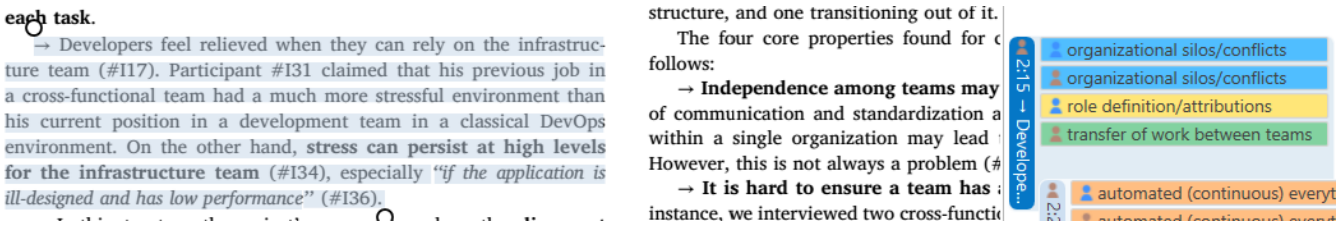


Fig. 4

Solution: NA

5. There is a disagreement in the quotation 2:23 (see Fig. 5):

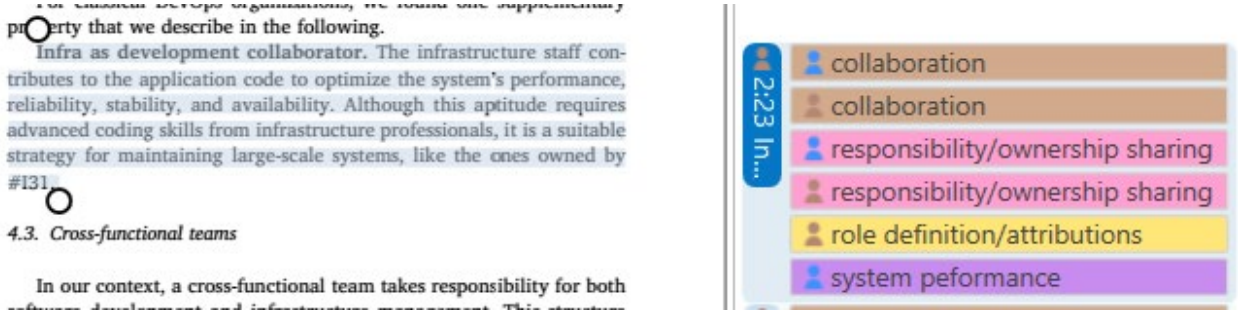


Fig. 5

Solution1: NA

Solution2: Deletion of the code "system performance" for several reasons: 1) performance is only one of all the NFR mentioned, therefore, it would be necessary to look for a more general NFR that encompasses the others to avoid future disagreements, and 2) I'm not able to find the relation between system's NFR and a model for devops team structures. Code "responsibility/ownership sharing" describes NFR shared responsibility among the team members, which I think is enough.

6. There is a disagreement in the quotation 2:31 (see Fig. 6):

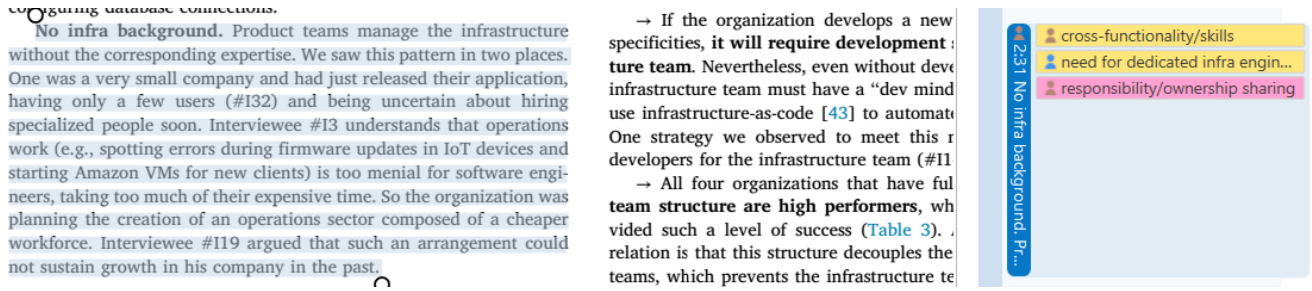


Fig. 6

Solution: **merging of codes** “cross-functionality/skill” and “need of dedicated infra engineers”. I’ve updated the comment of this code as follows: “From multidisciplinary/poly-skilled teams (i.e., teams with all the necessary skills such as development, infrastructure, etc.) to teams with a lack of skills/knowledge/background. This can be addressed by product teams with their own infrastructure staff/engineers. “

7. There is a disagreement in the quotation 2:35 (see Fig. 7):

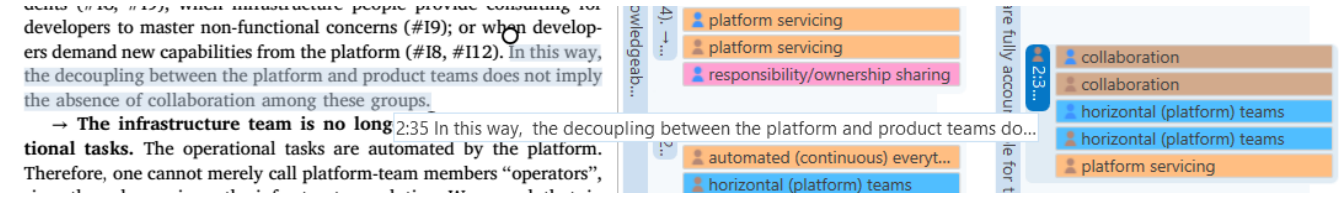


Fig. 7

Solution: NA

8. There is a disagreement in the quotation 2:36 (see Fig. 8):

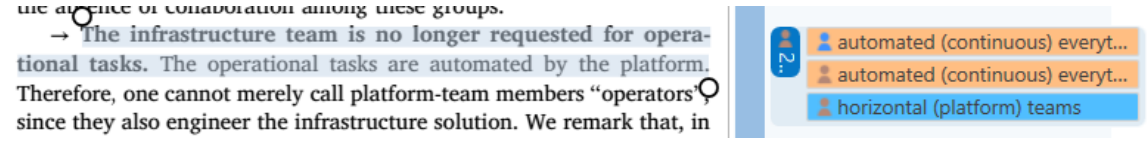


Fig. 8

Solution: I have lengthened the quotation to include the semantic information necessary to induce the code platform teams

9. There is a disagreement in the quotation 2:37 (see Fig. 9):

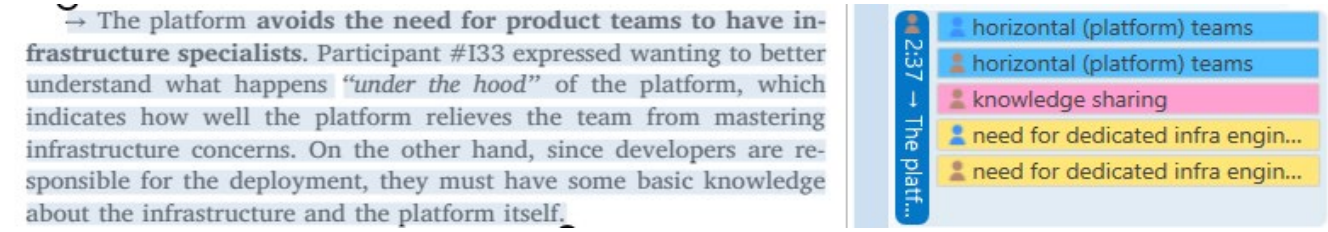


Fig. 9

Solution: NA

10. There is a disagreement in the quotation 2:40 (see Fig. 10):

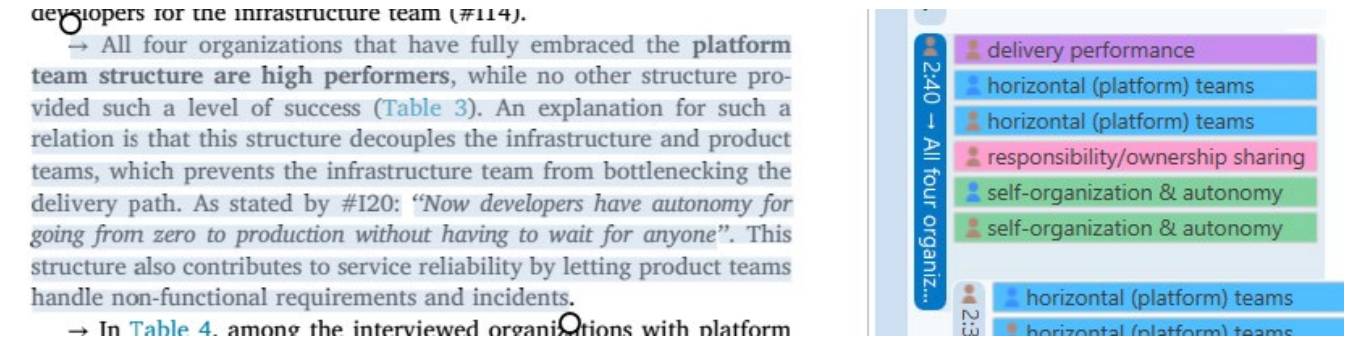


Fig. 10

Solution: I’ve added a new comment in the code “delivery performance”

11. There is a disagreement in the quotation 2:41 (see Fig. 11):

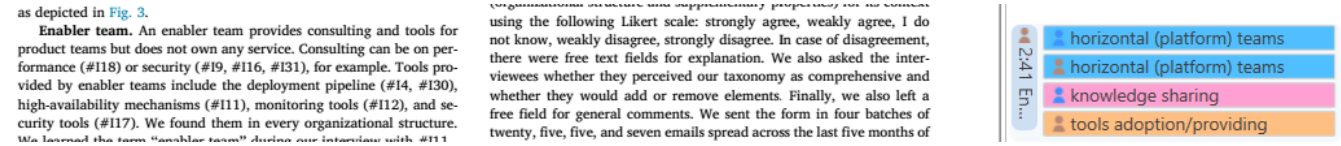


Fig. 11

Solution: I've added a new comment in the code "knowledge sharing"

12. There is a disagreement in the quotation 1:11 (see Fig. 12):

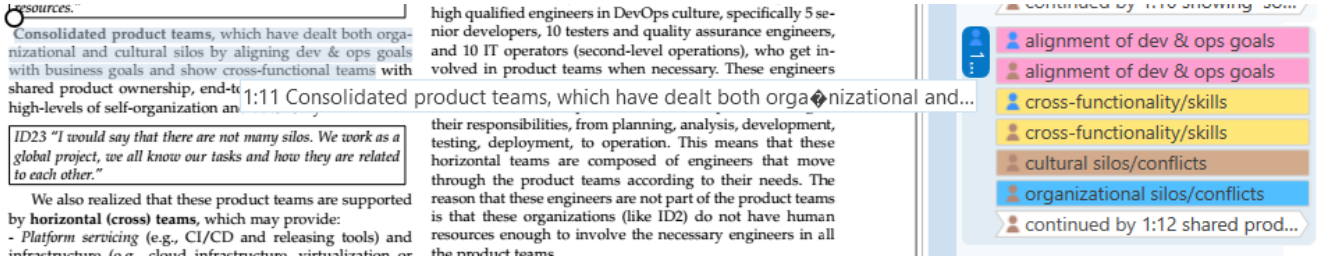


Fig. 12

Solution: NA

Results:

- 2 documents: D1 and D2
- 71 quotations
- 7 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, MONITORING, SHARING, SKILLS & ROLES, STRUCTURE
- 21 codes

CODER 3 (Jorge Pérez) - "UnifiedDevOps OpenCoding ITE1 (JP).atlproj9"

- 1 memo:

Coding has been easier (straightforward) in the UPM paper than in the USP paper. There may be a bias in this respect since the first coder is the author of the UPM document. In general terms, some clarifications are needed:

- The code "values & best practices" includes the definition of other codes such as "communication", "collaboration", ("continuous integration, continuous testing..." which can be confused with "automated") and "infrastructure as code" can be confused with the code "platform servicing". More differentiation is needed in this code of "values & best practices". For example, only use it when it refers jointly to more than one of these characteristics. Another option is to delete it.

- the differentiation of the codes "transfer of work..." and "responsibility/owner..." is not clear. In fact, the phrase "Ownership sharing is also... of done" in the code responsibility/... tends to be confusing. Non-ownership sharing implies a transfer of work and this rule can be used when coding.

CODER 4 (Carla Rocha) - "UnifiedDevOps OpenCoding ITE1 (CR).atlproj9"

Changes:

- 1 memo:

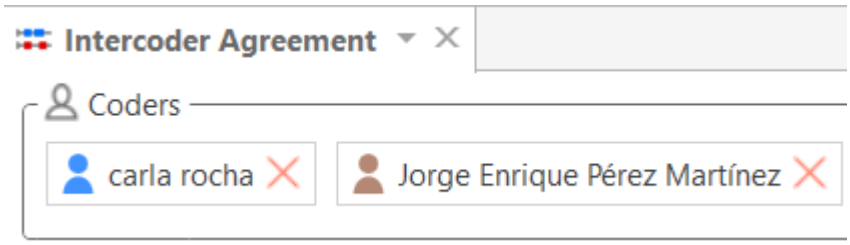
While coding the UPM paper, found the similarity between the code names similar or identical to the nomenclature used in the article could induce bias in the coding process. The same association is not observed in the usp article.

I also found the codes from the codebook used to illustrate teams organization and attribution could be sometimes ambiguous. The specifics of devops center/platform teams attribution in its variations were hard to fit into a code (from skills and roles). Maybe a new group for attribution would solve this issue.

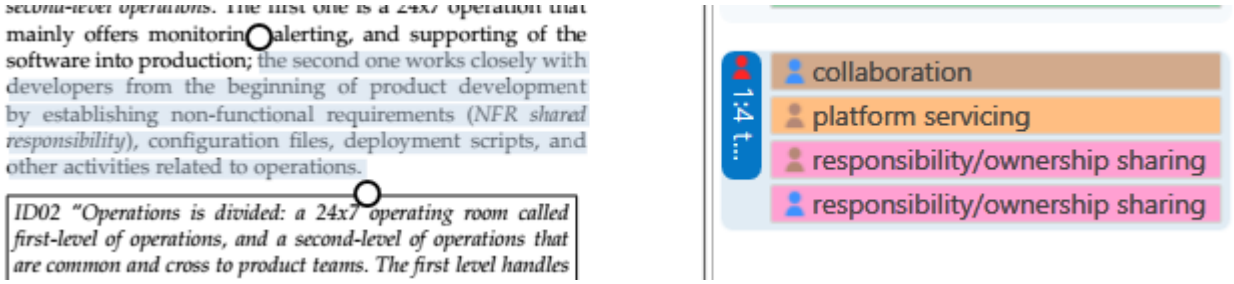
ANALYSIS OF DISAGREEMENTS IN CODING

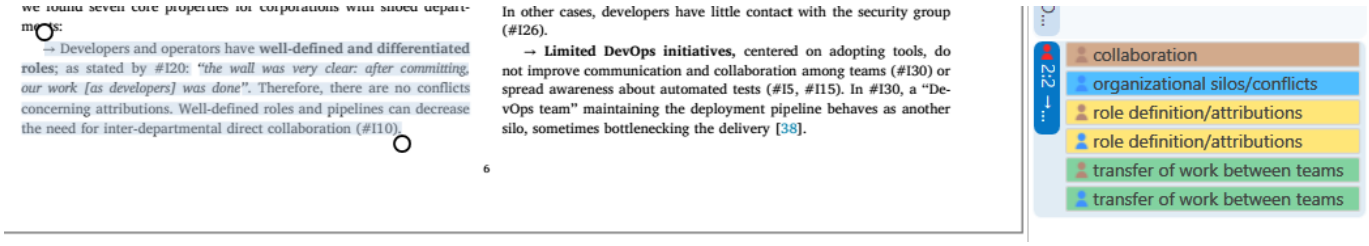
Krippendorf Cualpha = 0,761

See UnifiedDevOps OpenCoding ITE1 ICA (cr + jp) - alpha binary (PDF and XLSX formats)
See UnifiedDevOps OpenCoding ITE1 ICA (cr + jp) - cu alpha (PDF and XLSX formats)



1. The code **collaboration** is sometimes used by Carla and not used by Jorge and vice versa (see quotations 1:4 and 2.2). How do you (Carla and Jorge) think this code (or its explanation) can be improved?



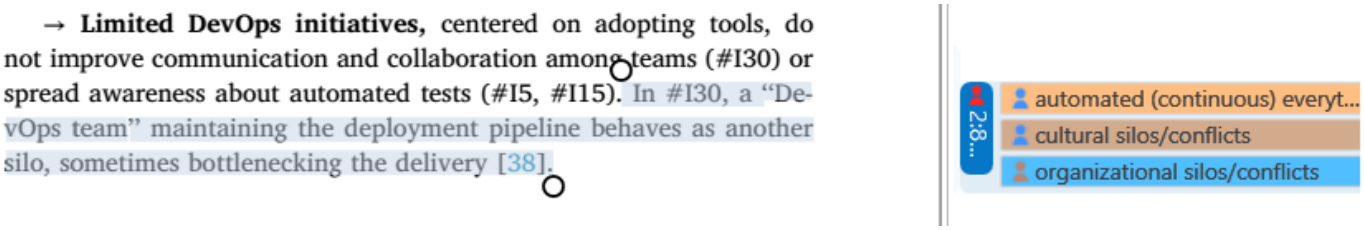


Solution/Discussion:

[Carla] In the sentence "the second one works closely with development from the beginning", i think it could be both collaboration and ownership sharing (Or one of them). Also the code "transfer of work between teams" also implies collaboration. One possible solution is to assign some sort of equivalence between these codes or refine the code "Collaboration" (with categories of collaboration = transfer work between teams or ownership sharing or regular communication between teams, etc). Or improve the description of collaboration code (with example when is applicable and differentiate it with more specific collaboration codes). Suggestion: Update the code collaboration to: "From eventual collaboration, which may generate conflicts and disagreements on decisions, to daily collaboration (working together regularly on a daily basis). Operation works closely with developers or collaborate with other teams/departments that own some of these skills"

[Jorge] I have not coded "collaboration" because it is not explicit in the text and the code "responsibility/ownership sharing" implicitly implies such collaboration. I believe that the code collaboration should only be assigned when it is explicitly mentioned in the text with whatever meaning it has (which is very broad). The rest of the codes that have to do with collaboration also have to do with other codes. It is precisely this task that will allow us to establish the relationships between them. It is a priority that this code, which is inherent to the DevOps principles, be formulated as a constructor with the multiple relationships that may arise and not start subordinating it to other concepts. To be operational: I maintain the collaboration code (in its current definition) but keeping in mind that this code must be related to the code "transfer of work between teams" and to the code "ownership sharing", as Carla points out.

2. In quotation 2:8 there was no match. How do you (Carla and Jorge) think this code (or its explanation) can be improved?

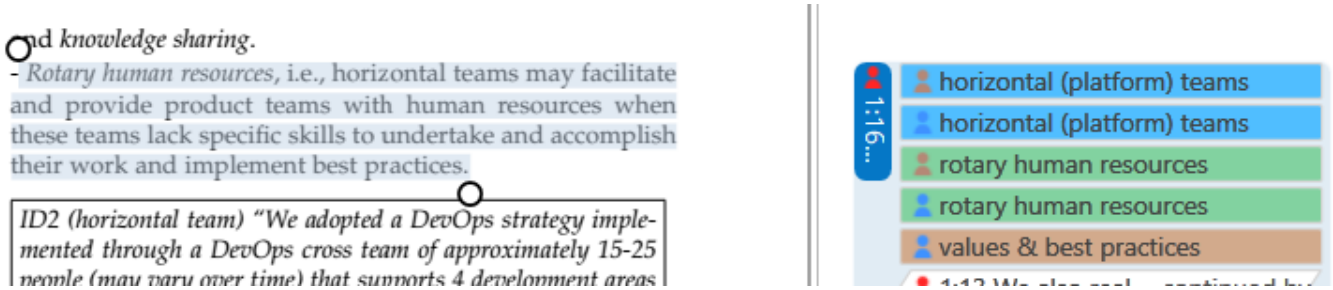


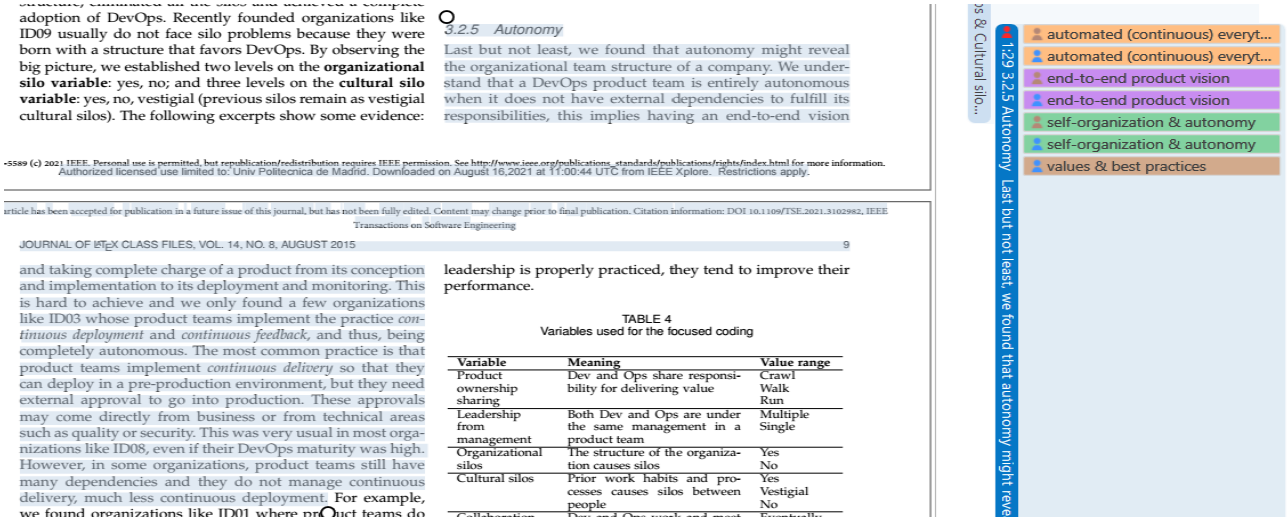
Solution/Discussion:

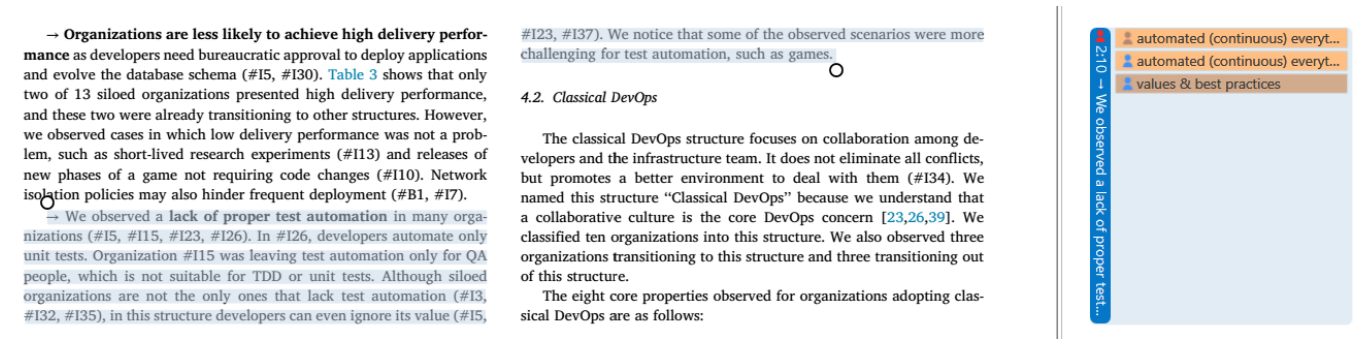
[Carla] Update the cultural/organization codes to avoid ambiguity. Suggestion: "Cultural silos conflicts: From non-cultural barriers to existing cultural barriers. Not to be confused with organizational/silos conflicts. While the cultural silos/conflicts focus on practices and culture that leads/break to barriers, organization silos/conflicts focus on team structures".

[Jorge]. I have not marked automated because I understand that the quotation does not include "automated test". I have interpreted from the text that the DevOps team does its work (deployment pipeline) without informing the development team. For me this is a silo in terms of work organization and I understand that there is no cultural silo if we are talking about the DevOps team. On the contrary, I understand that if there is a cultural silo, there will necessarily be an organizational silo. Personally I think there needs to be more characterisation of the cultural silo.

3. The code **values & best practices** is sometimes used by Carla and not used by Jorge (see quotations 1:16, 1:29 and 2:10). How do you (Carla and Jorge) think this code (or its explanation) can be improved?





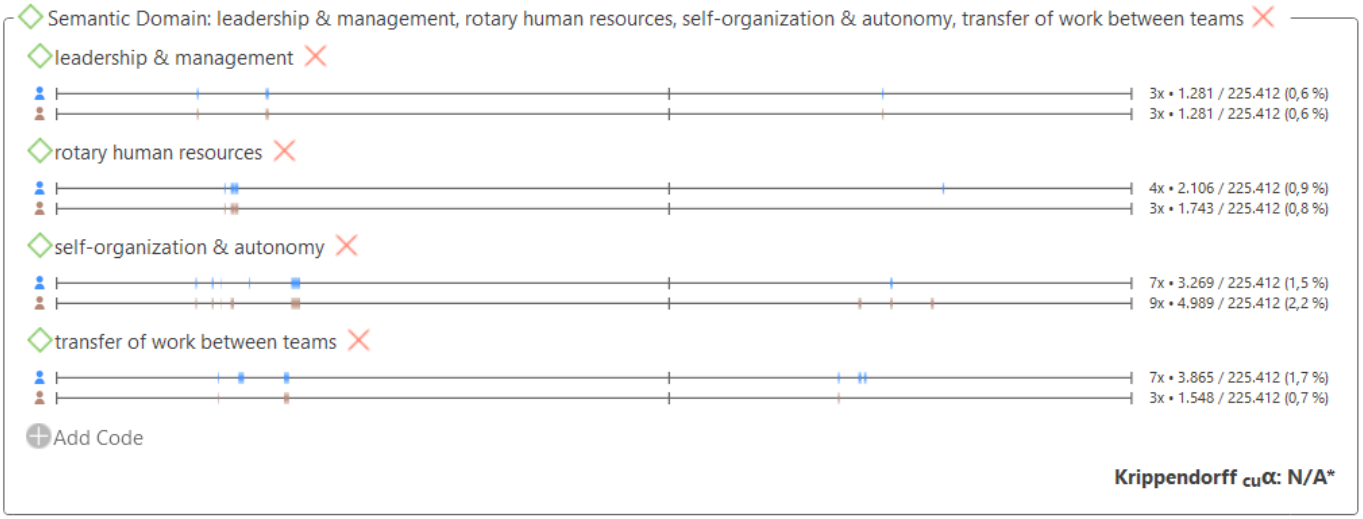


Solution/Discussion:

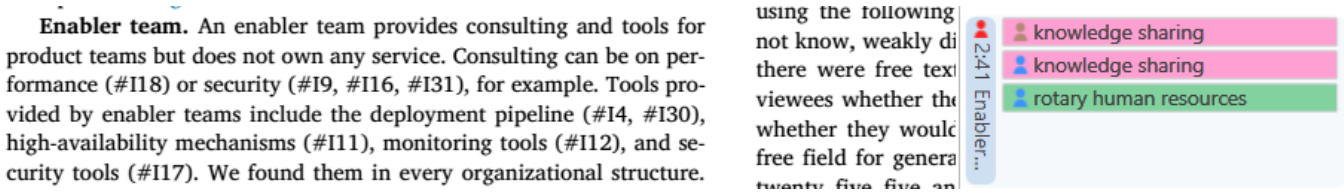
[Carla] - NA

[Jorge] It is true that in quote 1.16 I forgot to put the code "value & best practices". As for quote 1.29, I don't see the need to insert this code. As for quote 2.10, I put that code on the understanding that test automation corresponds to a best practice. Perhaps, as I said at the beginning, it is necessary to indicate when to use this code and when not to use it (or to eliminate it).

following semantic domain shows some disagreements.



4. The code **rotary human resources** was used by Carla and not used by Jorge (see quotation 2:41). How do you (Carla and Jorge) think this code (or its explanation) can be improved?



Solution/Discussion: [Jessica] Although it is not my turn, it strikes me that the code **automated (continuous) everything** is not used as it says "*Tools adoption or tool providing for deployment pipeline, monitoring, security, etc. Not to be confused with platform servicing; it only provides tools and pipelines for automated delivery/deployment in certain contexts.*" and the quotation explicitly mentions these terms. How do you (Carla and Jorge) think this code (or its explanation) can be improved so that you had used it in this quotation?

Also, it strikes me the absence of the code **horizontal (platform) teams**. I think that it is necessary to complete the description of this code. I suggest the following:

DevOps Centers of Excellence, chapters, guilds, platform teams, etc. or any other enabler team which could provide some of the following items: platform servicing, tools, consulting, evangelization, mentoring, human resources, etc. (some of them, not all).

[Carla] Agree with jessica. When looking from this perspective, it looks like codes are missing. To differentiate and complement the code rotatory human resources (with exclusive dedication but limited in time), agree with @jessica suggestion (consulting).

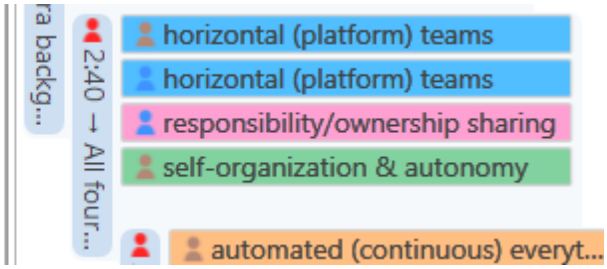
[Jorge] Right. In quote 2.41 the terms that Jessica points out appear and can be associated with automated. But I do not agree with increasing the description of horizontal code (platform) teams. The extended definition contains terms from other codes: platform servicing, evangelisation and mentoring, human resources (from rotary human resources). If the extension is made and these terms are included, even though they are codes from different domains, it seems to me to be overcoding. It gives the impression that the same concept is present in several codes without any nuance to differentiate it.

5. The code **self-organization & autonomy** was used by Jorge and not used by Carla (see quotation 1:20, 2:9 and 2:40). How do you (Carla and Jorge) think this code (or its explanation) can be improved?

platform, infrastructure, operation, and/or mentoring among others, as a service. This gives autonomy to product teams. We wrote some memos to clarify this meaning as follows:

MEMO: *Horizontal (cross) DevOps teams, either DevOps CoE, DevOps chapter, or Platform team, aim to provide a DevOps platform, IT operation, or mentoring, for autonomous product teams. They own DevOps skills & culture, platform, tools, and infrastructure to provide product teams with (i) servicing of CI/CD platform and environments for dev, test, or even pre-production, (ii) mentoring and evangelizing, and sometimes (iii) engineers who get involved in product teams with exclusive dedication but limited in time until these teams are capable of the "you build it, you run it".*

→ 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.

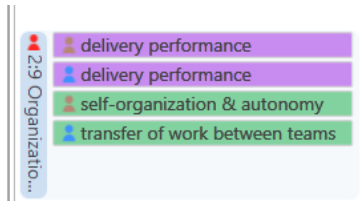


Additionally, in quotation 2:9, there is a disagreement in the use of a code of the same semantic domain. How do you (Carla and Jorge) think this code (or its explanation) can be improved?

→ **Organizations are less likely to achieve high delivery performance** as developers need bureaucratic approval to deploy applications and evolve the database schema (#I5, #I30). Table 3 shows that only two of 13 siloed organizations presented high delivery performance, and these two were already transitioning to other structures. However, we observed cases in which low delivery performance was not a problem, such as short-lived research experiments (#I13) and releases of new phases of a game not requiring code changes (#I10). Network isolation policies may also hinder frequent deployment (#B1, #I7). #I23, #I37). We notice that some of the observed scenarios were more challenging for test automation, such as games.

4.2. Classical DevOps

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 (#I34). We named this structure "Classical DevOps" because we understand that



Solution/Discussion:

[Carla] The quotes were obviously the Self-organization and autonomous code. My mistake. It is the the management group, so I suggest we update the code to "Teams self-organized & autonomous" : From external and/or bureaucratic dependencies and approvals to high levels of Teams self-organization and autonomy (i.e., the team has freedom and autonomy to organize its tasks, budget, infrastructure, etc.). Alignment with the Amazon motto ''You built it, you run it''. It explicits the context of the code (when reading for the first time, and considering the group it was not clear to me the context.

[Jorge] N/A

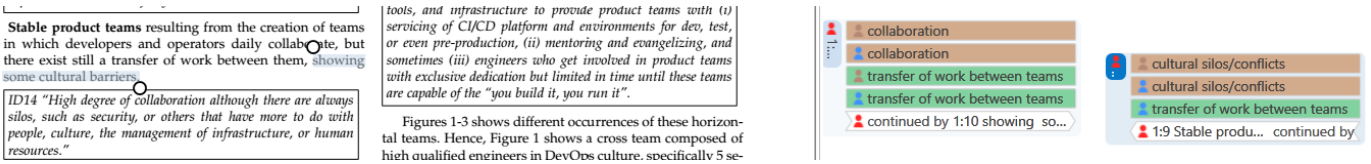
[Carla] With respect to the quotation 2:9, THE FRASE "As developers need bureaucratic approval to deploy application and evolve database schema" could be interpreted as a lack of self-organization and autonomy (From external and/or bureaucratic dependencies and approvals to high levels of self-organization and autonomy), or caused by the transfer between teams (here we imply teams could also mean management, since we are in management group - Hence, the definition of development "done" may go from committing a change transfer of work between teams. The ambiguity could be minimized with the update the transfer work between teams:

Transfer of work and responsibilities between teams (e.g., between development and infrastructure/operation teams). Hence, the definition of development "done" may go from committing a change transfer of work between teams, bureaucratic approvals, to running the change in production-like environments.

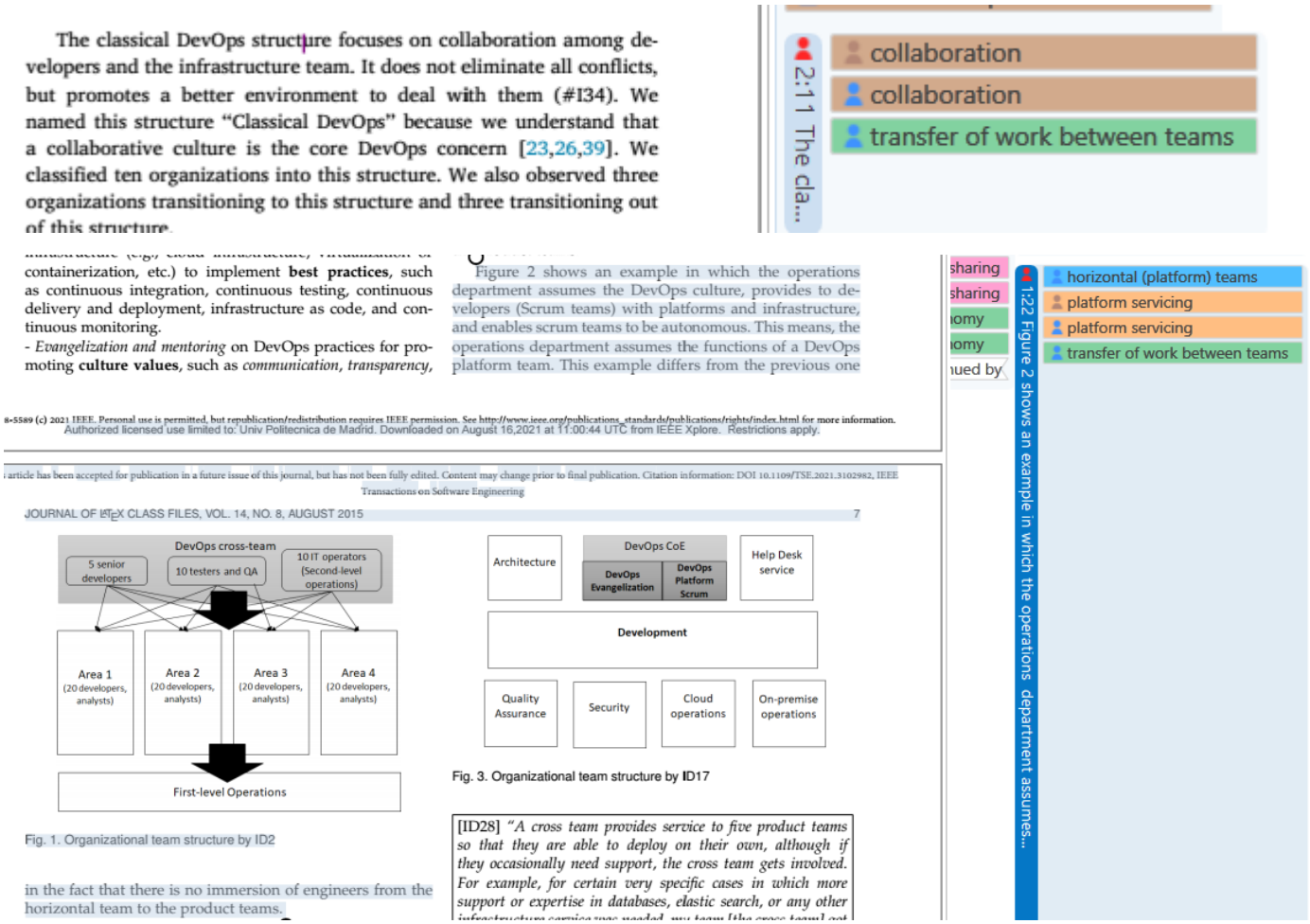
When there is transfer of work, depending on the high/poor trust/confidence on the other team, it may generate stress in the teams.

6. The code **transfer of work between teams** was used 7 times by Carla and only 3 times by Jorge. The quotations in disagreement are 1:10, 1:22, 2:9 (that was described in the previous item), and 2:11. How do you (Carla and Jorge) think this code (or its explanation) can be improved?

The quotation 1:10 is related to 1:9 so I think that Jorge forgot coding **transfer of work between teams**



[Jorge] Yes, my mistake



Solution/Discussion:

[Carla] NA

[Jorge] It is true that I should have coded horizontal (platform) teams since it is explicitly stated, but I don't see the need to transfer...

The following semantic domain shows some disagreements (alpha binary below the threshold).

7. Quotation 1:21 explicitly mentions "these horizontal teams are composed by …". However, Carla didn't use the code **horizontal (platform) teams**. How do you think this code (or its explanation) can be improved?

Figures 1-3 shows different occurrences of these horizontal teams. Hence, Figure 1 shows a cross team composed of high 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.

Solution/Discussion:

[Carla] Nothing to do. My mistake

[Jorge] N/A

8. Quotation 1:22 explicitly mentions "the operations department assumes the functions of a DevOps platform team". However, Jorge didn't use the code **horizontal (platform) teams**. How do you think this code (or its explanation) can be improved?

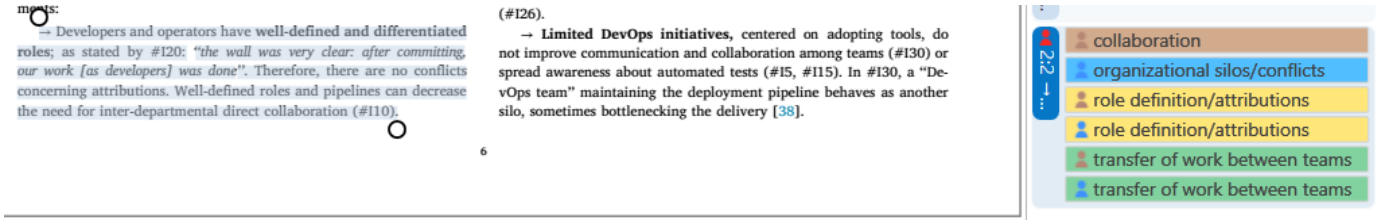
the product teams.

Figure 2 shows an example in which the operations department assumes the DevOps culture, provides to developers (Scrum teams) with platforms and infrastructure, and enables scrum teams to be autonomous. This means, the operations department assumes the functions of a DevOps platform team. This example differs from the previous one

Solution/Discussion: @Jorge: I indicated above that it was indeed my mistake and I should have coded it.

9. In quotation 2:2 Carla used the code **organizational silos/conflicts** and Jorge did not. This code is described as "*From non-organizational silos/barriers to siloed departments and existing organizational barriers (segregated departments; frictions, conflicts, and disagreements among silos; silos that become bottlenecks; minimal or no*

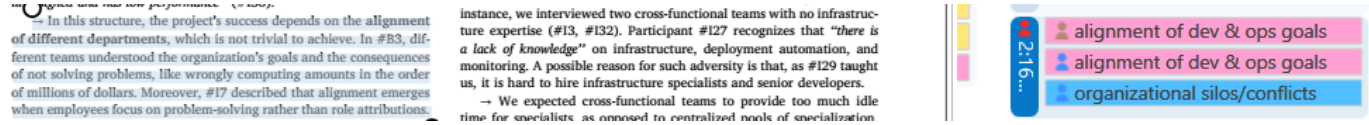
awareness of what is happening on the other side of the wall)." How do you (Jorge) think this code (or its explanation) can be improved?



Solution/Discussion: [Jorge] I don't see in the quote the need to use that code. It does talk about collaboration but the quote talks about differentiated roles, not about organizational conflict.

10. This disagreement was highlighted in item number 2.

11. Carla used the code **organization silos/conflicts** and Jorge not. How do you (Carla and Jorge) think this code (or its explanation) can be improved?

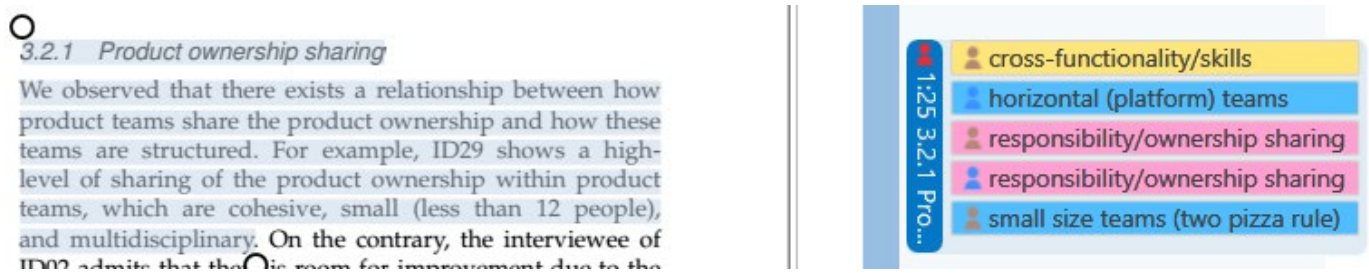


Solution/Discussion:

[Carla] NA

[Jorge] I see no reason to use that code

12. Carla used the code **horizontal (platform) teams** and Jorge used the code **small size teams (two pizza rule)**. Why was the code **horizontal (platform) teams** used and how do you (Carla) think this code (or its explanation) can be improved?



Solution/Discussion:

[Carla] Suggestion- Add comment to "small size teams (two pizza rule)" to differentiate to cross-functionality (Which I guess I ended confusing with small size teams - From multidisciplinary/poly-skilled teams (i.e., teams with all the necessary skills such as development, infrastructure, etc.)

Small size teams (Two pizza rule) - Product teams promoting skills over roles. Differentiate from multidisciplinary/poly-skilled teams

ITERATION 2

MASTER

- 1 document: D4
- 21 quotations
- 7 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, MONITORING, SHARING, SKILLS & ROLES, STRUCTURE
- 21 codes

CODER1 (Jessica Diaz) - "UnifiedDevOps OpenCoding ITE2 (JD).atlproj9"

- 28 codes: I've created 7 codes (and a new semantic domain: IT INFRASTRUCTURE) and modified the comments of some previous codes as follows:

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 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.

platform builder

This code refers to the platform built to provide the means to an end in software development by creating automated pipelines and managing infrastructure (physical and/or virtual environments) to enable CI/CD.

automated (continuous) everything automated application life-cycle management

This code refers to the automation of the processes of the application life cycle (from development and deployment pipelines to monitoring and operational tasks) and the tools adoption or tool providing for supporting these processes. Not to be confused with platform servicing; this code only refers to pipelines and tools for automated delivery/deployment and operation. The level of automation may range FROM high-level of build automation (continuous integration, CI); testing automation (continuous testing); delivery automation (continuous delivery, CD); deployment automation (continuous deployment, CD); operational tasks automation (continuous feedback/monitoring from operations to development) TO non-automation.

automated infrastructure management

This code refers to the automation of the infrastructure management and the use of configuration management tools. Not to be confused with platform servicing; this code only refers to techniques, technologies, and tools for automated infrastructure management. The level of automation may range from low to high levels of automated infrastructure (aka. Infrastructure as Code, IaC) to non-automation.

responsibility/ownership sharing

From shared responsibility of the tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility[1], monitoring and incident handling shared responsibility, etc.), the product and output artifacts (e.g., databases) to separated responsibilities and tasks (developers, infrastructure/staff and operators have different responsibilities and tasks). Thus, if there is no shared responsibility, there is necessarily a transfer of work development to production and operation (and vice versa). However, ownership sharing is related to a new definition of "done" (e.g., developers work doesn't finish with coding, but they support deployment in production).

[1] e.g., developers perform automated infrastructure management (write infrastructure code, IaC)

horizontal platform team enabler platform team

This code refers to specific structures/teams that organizations create to satisfy some needs of product teams, such as platform servicing and tools (mainly for infrastructure and deployment pipelines), consulting, evangelization, mentoring, human resources, etc. Thus, they behave as enabler teams by providing these capabilities. Sometimes enabler teams are the bridge interface between developers and IT Operations to drive the DevOps values and practices. These structures/teams are named in different ways, e.g., DevOps Centers of Excellence, chapters, guilds, platform 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.

devops (bridge) team

DevOps teams collaborate, assist, support, help developers, mainly by deploying and hosting applications in the platforms they build (platform builders), monitoring and providing support. In these DevOps teams work DevOps engineers, who are the DevOps practices facilitators; hence, they create, deploy, and manage both the infrastructure (environments) and deployment (CI/CD) pipelines. 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 values and practices.

cloud

containerization

hybrid (on-premises & cloud)

on-premises

role definition/attributions

From "skills over roles" and T-shape/DevOps engineers (aka. full stack engineers) to well-defined and differentiated roles (e.g., dev versus ops, so that they work together but in different tasks).

cross-functionality/skills

From multidisciplinary/poly-skilled teams (i.e., teams with all the necessary skills such as development, infrastructure, etc.) to teams with a lack of skills/knowledge/background. This can be addressed by product teams with their own infrastructure staff/engineers or senior developers that are responsible of infrastructure (with knowledge on automated infrastructure, and thus, DevOps facilitators).

- 2 Memos

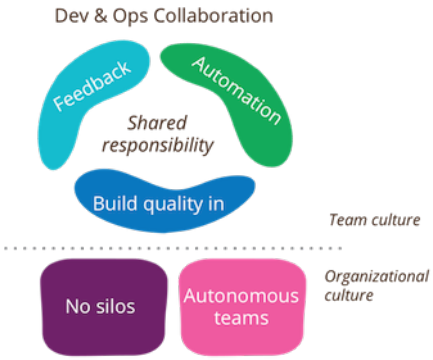
Organizational structures

DevOps Teams may behave as a new silo.

Shared Responsability

Although shared responsibility is a key aspect of DevOps culture (as a step beyond collaboration), the papers we're analyzing show some DevOps implementations/taxonomies with separate dev and ops teams, even a devops bridge team, in which each one is responsible of their tasks (non-shared responsibility), which implies a transfer of work between teams.

https://martinfowler.com/bliki/DevOpsCulture.html --> An attitude of shared responsibility is an aspect of DevOps culture that encourages closer collaboration. It's easy for a development team to become disinterested in the operation and maintenance of a system if it is handed over to another team to look after. If a development team shares the responsibility of looking after a system over the course of its lifetime, they are able to share the operations staff's pain and so identify ways to simplify deployment and maintenance (e.g. by automating deployments and improving logging). When operations staff share responsibility of a system's business goals, they are able to work more closely with developers to better understand the operational needs of a system and help meet these. In practice, collaboration often begins with an increased awareness from developers of operational concerns (such as deployment and monitoring) and the adoption of new automation tools and practices by operations staff.



CODER 2 (Carla Rocha) - "UnifiedDevOps OpenCoding ITE1 (IS).atlproj9"

NA

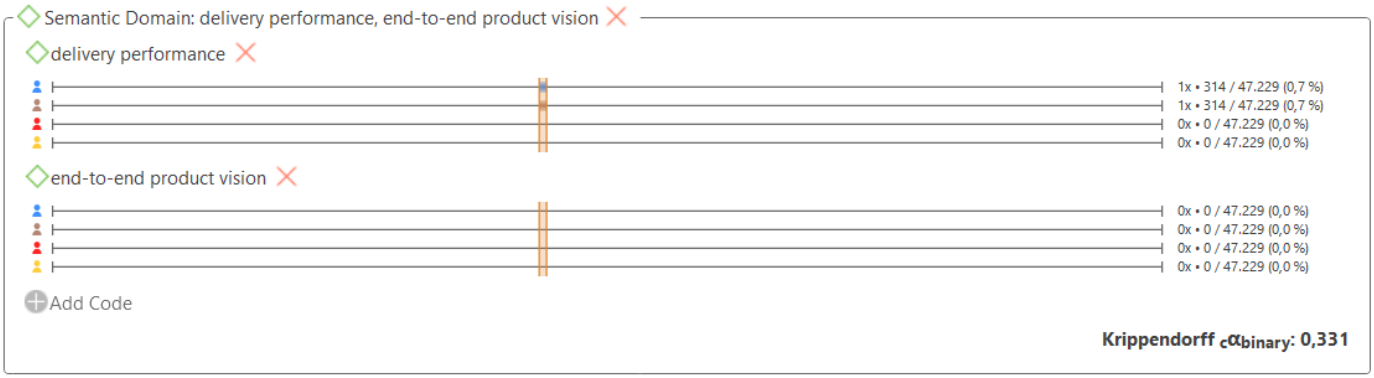
CODER 3 (Jorge Pérez) - "UnifiedDevOps OpenCoding ITE1 (JP).atlproj9"

NA

CODER 4 (Isaque Alves) - "UnifiedDevOps OpenCoding ITE1 (CR).atlproj9"

NA

MISCONCEPTIONS IN CODING



If we remove this disagreement on the code **delivery performance** (Carla & Isaque unlink the code or Jessica & Jorge link this code) the Cu-alpha = 0,989, which is great! Can we consider this disagreement as a mistake? The quotation says:

> *A. Description of DevOps*
>
> Generally, participants described DevOps as better collaboration between developers and Ops teams. Some interviewees also described DevOps as end-to-end automation of the software development pipeline, providing better software quality, and creating seamless workflow of products at the shortest possible time. We divided these descriptions into two groups: DevOps as a culture, and as a job description.

I think that Isaque and Carla used the code **delivery performance** as the quotations mentions “workflow of products at the shortest possible time” Is it right? The comment of this code says:

> From high (in short, high performers) to low delivery performance (deployment frequency, mean-time to recovery, lead time). Sometimes, deployment/delivery frequency is limited to external factors, such as periodic time slots (windows release).

Is it possible to explicitly associate “workflow of products at the shortest possible time” to delivery performance indicators (defined by DORA reports) such as deployment frequency, mean-time to recovery, or lead time?

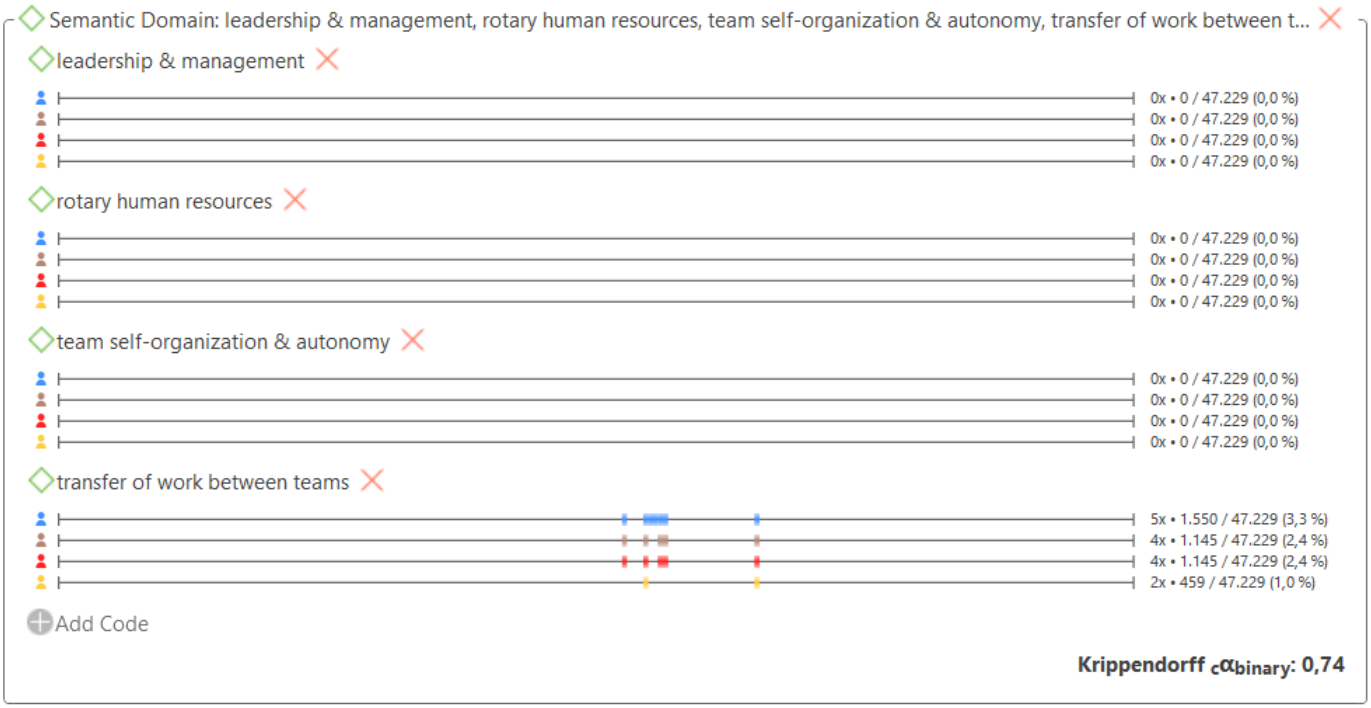
Jessica: I think there is no direct relation because “shortest possible time” does not explicitly refer to deployment/delivery. However, I understand that, given our background in the topic, we could interpret this. So, we could both unlink or link the code to this quotation. Although I am a little more inclined to the first option.

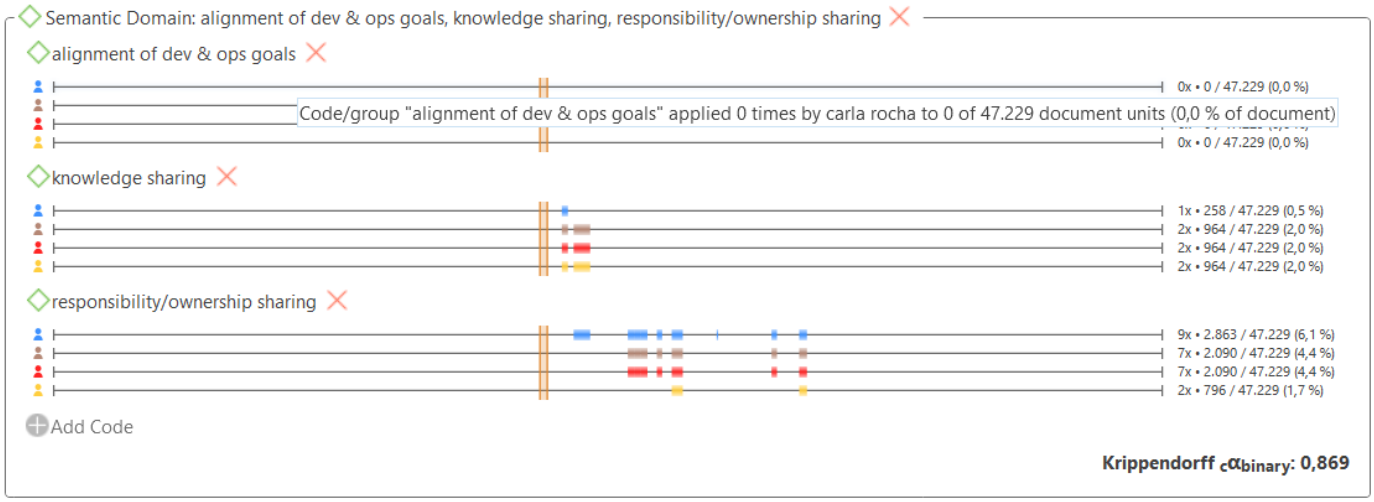
Isaque: I agree that according to the code comment, it is wrong. This section does not make it clear that it is about delivery performance. I inserted this code because, in my understanding to achieve a “workflow of products at the shortest possible time” it is necessary to think about delivery performance.

Carla: I agree with both views. "Workflow of product" suggests all the activities related to software products, including delivery. Therefore, when it says "shortest possible time", the metric that reflects this measure would be delivery performance. So, reach the conclusion the phrase refers to delivery performance or not link to it are both coherent.

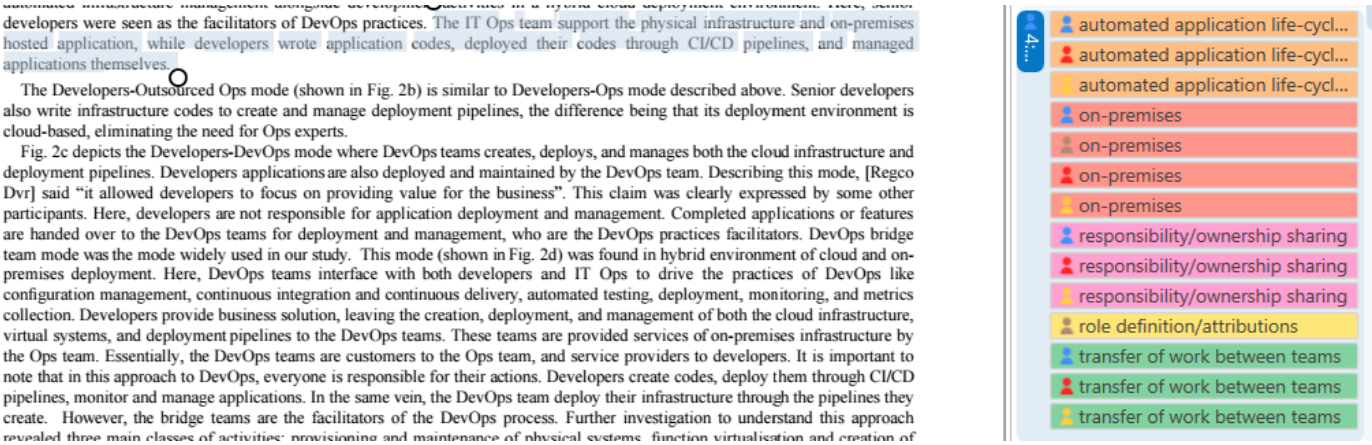
ANALYSIS OF DISAGREEMENTS IN CODING

Krippendorf Cualpha = 0,905





1. In quotation 4:7, Jorge didn't code "automated application".



developers were seen as the facilitators of DevOps practices. The IT Ops team support the physical infrastructure and on-premises hosted application, while developers wrote application codes, deployed their codes through CI/CD pipelines, and managed applications themselves.

The Developers-Outsourced Ops mode (shown in Fig. 2b) is similar to Developers-Ops mode described above. Senior developers also write infrastructure codes to create and manage deployment pipelines, the difference being that its deployment environment is cloud-based, eliminating the need for Ops experts.

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. Describing this mode, [Regco Dvr] said "it allowed developers to focus on providing value for the business". This claim was clearly expressed by some other participants. 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 on-premises deployment. Here, DevOps teams interface with both developers and IT Ops to drive the practices of DevOps like configuration 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. Developers create codes, deploy them through CI/CD pipelines, monitor and manage applications. In the same vein, the DevOps team deploy their infrastructure through the pipelines they create. However, the bridge teams are the facilitators of the DevOps process. Further investigation to understand this approach revealed three main classes of activities: provisioning and maintenance of physical systems, function virtualisation and creation of

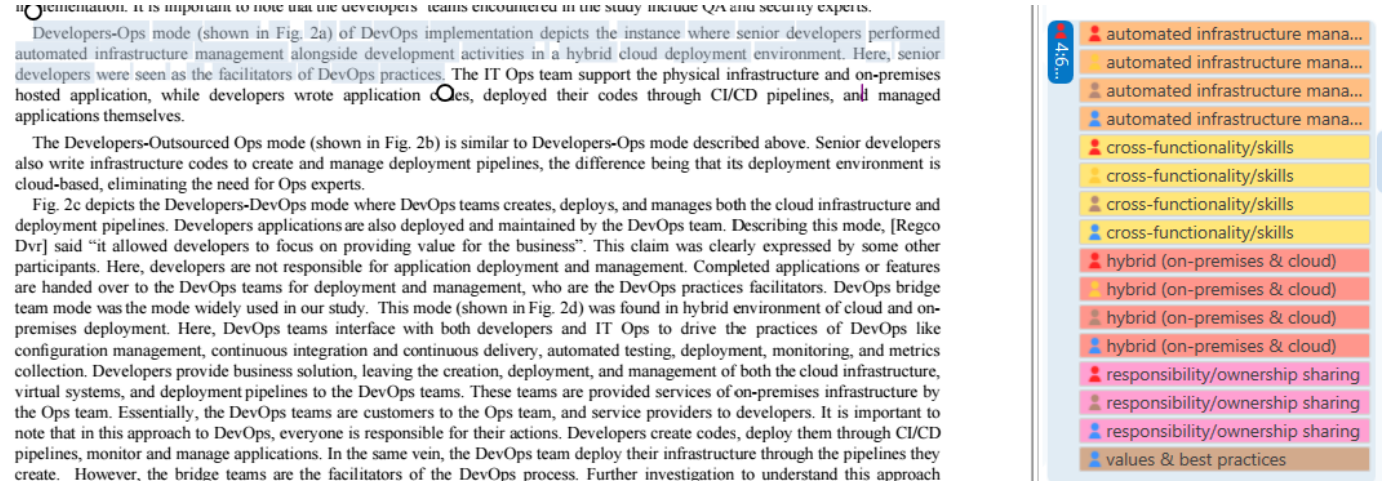
After discussion, we decided to modify the following codes:

platform builder: This code refers to the platform that an entity (e.g., a team, an external consultant, etc.) builds to provide automated CI/CD pipelines and managing infrastructure (physical and/or virtual environments).

automated application life-cycle management: This code refers to the automation of some of the processes of the application life cycle (from development and deployment pipelines to monitoring ~~and operational~~ tasks) and the tools adoption or tool providing for supporting these processes. Not to be confused with platform servicing; the code "automated application life-cycle management" refers to pipelines and tools for automated delivery/deployment and operation, whereas the code "platform servicing" refers to the offered services to product teams. The level of automation may range FROM high-level of build automation (continuous integration, CI); testing automation (continuous testing); delivery automation (continuous delivery, CD); deployment automation (continuous deployment, CD); operational tasks automation (continuous feedback/monitoring from operations to development) TO non-automation.

automated infrastructure management: This code refers to the automation of the infrastructure management and the use of configuration management tools. Not to be confused with platform servicing; the code "automated infrastructure management" refers to techniques, technologies, and tools for automated infrastructure management, whereas the code "platform servicing" refers to the offered services to product teams. The level of automation may range from low to high levels of automated infrastructure (aka. Infrastructure as Code, IaC) and configuration management to non-automation.

2. In quotation 4:6, 4:7, 4:9, 4:10 and 4:12, Jorge didn't code "**responsibility/ownership sharing**" and coded "role definition".

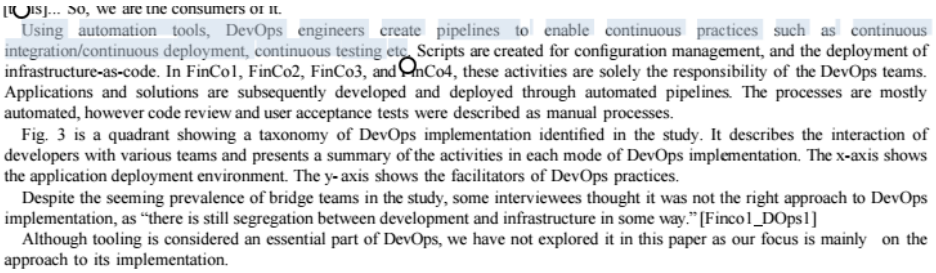


Jorge proposes the following:

responsibility/ownership sharing: From shared responsibility of the products, output artifacts (e.g., databases), and tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility*, monitoring shared responsibility, and incident handling shared responsibility, etc.) to separate responsibilities and tasks (each team member has different responsibilities and tasks). Thus, if there is no shared responsibility, there is necessarily a transfer of work development to production and operation (and vice versa). However, ownership sharing is related to a new definition of "done" (e.g., developers' work doesn't finish with coding, but they support deployment in production).

role definition/attributions: From "skills over roles" and T-shape/DevOps engineers (aka. full stack engineers) to well-defined and differentiated roles ~~(e.g., dev versus ops, so that they work together but in different tasks)~~.

3. In quotation 4:13, Jorge didn't code "**values & best practice**"

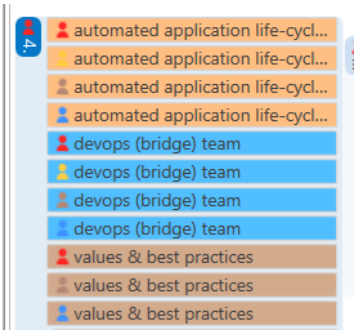

[tools]... So, we are the consumers of it.

Using automation tools, DevOps engineers create pipelines to enable continuous practices such as continuous integration/continuous deployment, continuous testing etc. Scripts are created for configuration management, and the deployment of infrastructure-as-code. In FinCo1, FinCo2, FinCo3, and FinCo4, these activities are solely the responsibility of the DevOps teams. Applications and solutions are subsequently developed and deployed through automated pipelines. The processes are mostly automated, however code review and user acceptance tests were described as manual processes.

Fig. 3 is a quadrant showing a taxonomy of DevOps implementation identified in the study. It describes the interaction of developers with various teams and presents a summary of the activities in each mode of DevOps implementation. The x-axis shows the application deployment environment. The y-axis shows the facilitators of DevOps practices.

Despite the seeming prevalence of bridge teams in the study, some interviewees thought it was not the right approach to DevOps implementation, as "there is still segregation between development and infrastructure in some way." [Finco1_DOps1]

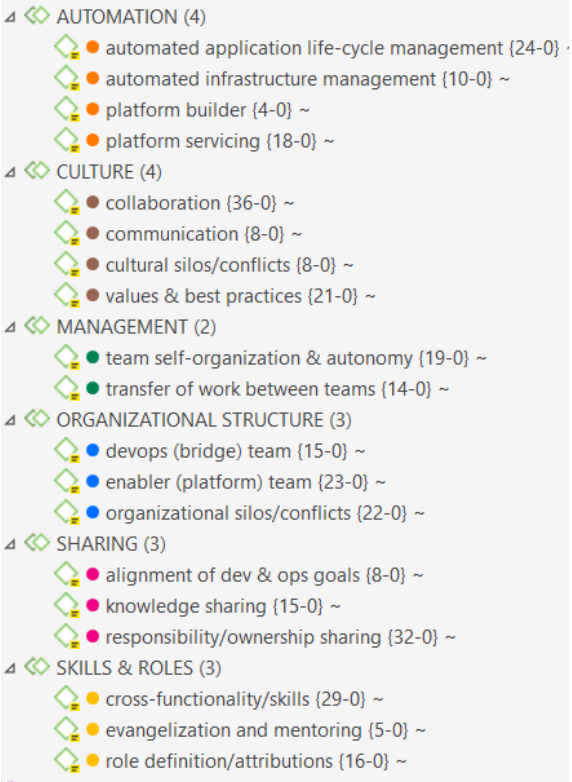
Although tooling is considered an essential part of DevOps, we have not explored it in this paper as our focus is mainly on the approach to its implementation.



values & best practices: This code refers to a generic concept to be used when a quotation explicitly refers to this generality such as DevOps best/good practices or mentions a large subset of these values and practices (continuous integration, continuous testing, continuous delivery and deployment, infrastructure as code, continuous monitoring, etc.), cultural values (collaboration, communication, transparency, etc.), and/or principles (customer-centric action, create with the end in mind, end-to-end responsibility, cross-functional autonomous teams, continuous improvement, automate everything you can, among others). If a quotation refers to a specific practice, value, and principle, specific codes should be used.

SELECTION OF CORE CATEGORIES

6 semantic domains and 19 codes



ITERATION 3

MASTER

- 1 document: D5 D6 D7 & D8
- 19 + 21 + 21 + 21 quotations
- 6 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, ORGANIZATIONAL STRUCTURE, SHARING, SKILLS & ROLES
- 19 codes

CODER 1 (Carla Rocha) - "UnifiedDevOps OpenCoding ITE1 (CR).atlproj9"

NA

CODER 2 (Jorge Pérez) - "UnifiedDevOps OpenCoding ITE1 (JP).atlproj9"

blame

From blameless contexts to blame contexts. In blameless contexts, the development team must not say that a given issue is a problem in the infrastructure, this is, operations team responsibility. Likewise, the operations team must not say that a failure was motivated by a problem in the application, this is, development team responsibility. The teams need to focus on solving problems, not on laying the blame on others and running away from the responsibility.

D7: new quotations 7.67, 7.68, 7.69, 7.70

D8: new quotations 8.12, 8.26 y 8.27

CODER3 (Jessica Diaz) - "UnifiedDevOps OpenCoding ITE2 (JD).atlproj9"

- 1 new memo

MEMO Conway's Law

Many organizations attempt to create small teams, but they often make the mistake of splitting them functionally based on technology and not on product or service. Amazon, in designing its organizational structure, was careful to follow Conway's Law: "Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.

MEMO Platform builder

According to D6 a platform builder team could be a centralized, temporary, or an external team.

- 3 new codes and 6 codes that were modified

(D7) alignment of dev & ops goals

From misalignment among teams that have their own interests and goals (local optimization) to alignment with business goals and global ones (product thinking). A typical example of misalignment is when "developers want to deliver as much as possible, whereas operations target stability".

(D7) automated application life-cycle management

This code refers to the automation of some of the processes of the application life cycle (from development and deployment pipelines to monitoring tasks) and the tools adoption or tool providing for supporting these processes. Not to be confused with platform servicing; the code "automated application life-cycle management" refers to pipelines and tools for automated delivery/deployment and operation, whereas the code "platform servicing" refers to the offered services to product teams. The level of automation may range FROM high-level of 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 TO non-automation.

Automated recovery refers to the ability to either replace a component that is not working or rollback a failed deployment without human intervention, this means, the capability that applications and infrastructure have to recover itself in case of failures (resilience). There are two typical cases of recovery automation: (1) in cases of instability in the execution environment of an application (a container, for example) an automatic restart of that environment will occur; and (2) in cases of new version deployment; if the new version does not work properly, the previous one must be restored. This auto restore of a previous version decreases the chances of downtimes due to errors in specific versions, which is the concept of zero down-time. [D7]

(D8) continuous improvement

This code refers to the ongoing improvement of products, services or processes through incremental and breakthrough improvements. Continuous improvement seeks to improve every process in your company by focusing on enhancing the activities that generate the most value for your customer while removing as many waste activities as possible.

(D8)metrics, visibility & feedback

Metrics (process, value, cost, and technical metrics) provide fast and continuous feedback from users, reduce the risk and cost of deployments, get better visibility into the delivery process itself, and manage the risks of software delivery more effectively.

(D8) product management

Product management (related to a product or even service model) versus traditional project management. Each product or service has a team associated with it, and that team is completely responsible for the product/service (from scoping out the functionality, to architecting it, to building it, and operating it). In this way, teams can innovate quickly with a strong customer focus.

(D5) knowledge sharing

From high to low levels of knowledge sharing (e.g., developers may have knowledge about infrastructure/platform, or minimal or no awareness of what is happening on the other side of the wall, aka. wall of confusion). Knowledge sharing may include a shared repository (the source of truth) and administrative rights to different environments (testing, pre-production, production). Knowledge sharing helps build trust between Devs & Ops personnel so that the former can share/delegate tasks to the latter second, and vice versa. However, knowledge sharing also increases the cognitive load (knowledge and expertise) that some members of the team acquire since sharing promotes that these members are responsible for everything.

(D5) enabler (platform) team

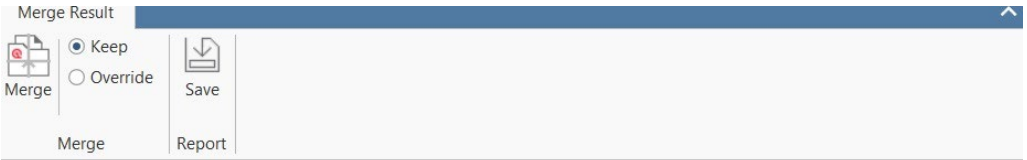
This code refers to specific structures/teams that organizations create to satisfy some needs of product teams. Sometimes it is not necessary to create new structures/teams because the operations team/departament takes over these needs and assists product teams. These needs can be platform servicing 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 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.

(D5) training, evangelization and mentoring

(D5 & D6) organizational silos/conflicts

From non-organizational silos/barriers (e.g. co-locating teams/departments) to siloed departments and existing organizational barriers (segregated departments; frictions, mistrust, conflicts, and disagreements among silos; silos that become bottlenecks; minimal or no awareness of what is happening on the other side of the wall).

CODER 4 (Isaque Alves) - "UnifiedDevOps OpenCoding ITE1 (IA).atlproj9"



master project "UnifiedDevOps Selective Coding ITE3 ICA (jp + jd + ia D5 to D8)"

Merge Conflicts

Codes

Code	Conflict	Master	Import
automated application life-cycle management	Comment	✓	✓
product management	Comment	✓	✓

automated application life-cycle management

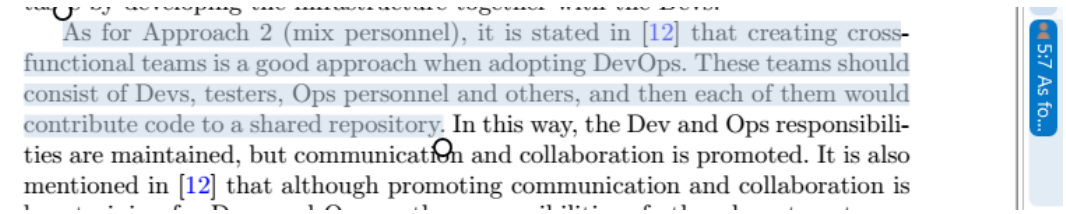
This code refers to the automation of some of the processes of the application life cycle (from development and deployment pipelines to monitoring tasks) and the tools adoption or tool providing for supporting these processes. Not to be confused with platform servicing; the code "automated application life-cycle management" refers to pipelines and tools for automated delivery/deployment and operation, whereas the code "platform servicing" refers to the offered services to product teams. The level of automation may range FROM high-level of 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 TO non-automation.

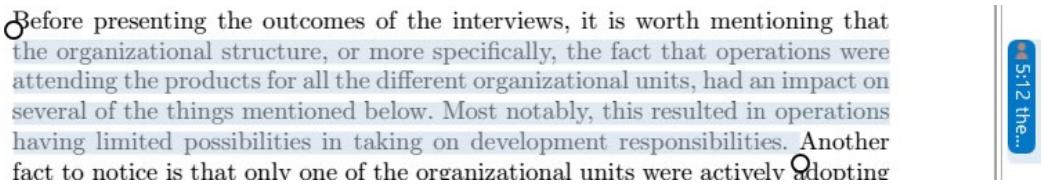
Automated recovery refers to the ability to either replace a component that is not working or rollback a failed deployment without human intervention, this means, the capability that applications and infrastructure have to recover itself in case of failures (resilience). There are two typical cases of recovery automation: (1) in cases of instability in the execution environment of an application (a container, for example) an automatic restart of that environment will occur; and (2) in cases of new version deployment; if the new version does not work properly, the previous one must be restored. This auto restore of a previous version decreases the chances of downtimes due to errors in specific versions, which is the concept of zero down-time. [D7]

This code refers to the ability to either replace a component that is not working or rollback a failed deployment without human intervention, this means, the capability that applications and infrastructure have to recover itself in case of failures (resilience). There are two typical cases of recovery automation: (1) in cases of instability in the execution environment of an application (a container, for example) an automatic restart of that environment will occur; and (2) in cases of new version deployment; if the new version does not work properly, the previous one must be restored. This auto restore of a previous version decreases the chances of downtimes due to errors in specific versions, which is the concept of zero down-time. [D7]

ANALYSIS OF DISAGREEMENTS IN CODING - DOCUMENTS D7 & D8
Krippendorf Cualpha = 0,9

ANALYSIS OF DISAGREEMENTS IN CODING - DOCUMENTS D5 & D6
Krippendorf Cualpha = 0,864





A New Source for Friction. In order to enable Devs to deal with operations tasks, it was necessary to give administration rights to Devs to different environments. Based on the comments from the employees, it was evident that gaining access served as a cause for friction and mistrust. It was also mentioned that the process for obtaining access was long and tedious.

The long process also had negative implications on the work efficiency, because employees often realized too late that they needed the access, causing extra delays. The decision process for who was granted the access was also described as unfair. Some employees mention that access seem to have been granted based on shown interest rather on experience and knowledge. Devs also complained about Ops getting access faster than Devs. This made them angry and irritated.

5:13 A New Source for Friction. In or...

Small Ops team with more responsibilities for Dev team: DevOps often recommends that developers take more accountability about their code in production environments [5]. Some interviewees' organizations have gradually and smoothly shifted operational responsibilities from infrastructure and operations teams to Dev team. By applying this change, Ops team is more responsible for mentoring, coaching and helping developers to write operational aspects of code, for example writing provisioning code. This strategy enabled the interviewees' organizations to make operations process easier and helped developers to commit codes that made less trouble. This is mainly because Ops team influenced the way the applications were configured to make them easier to deploy. Furthermore, Ops team may still exist to handle initial incidents in production environments. Hence, development team is not available like 24/7 to address incidents in production and initial incidents handling will be out of developers' accountability. As one interviewee commented:

6:4 S...
automated infrastructure mana...
devops (bridge) team
enabler (platform) team
responsibility/ownership sharing
training, evangelization and m...
training, evangelization and m...

devops (bridge) team

DevOps teams collaborate, assist, support, help 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) DevOps teams are the DevOps practices facilitators; hence, they usually create, deploy, and manage both the infrastructure (environments) and deployment (CI/CD) pipelines. 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 values and practices.

ITERATION 4

MASTER

- 1 document: D9
- 20 quotations
- 6 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, ORGANIZATIONAL STRUCTURE, SHARING, SKILLS & ROLES
- 23 codes

CODER 1 (Jessica Diaz) - "UnifiedDevOps OpenCoding ITE4 (JD).atlproj9"

culture, values & best practices

stack & tools sharing

From a holistic view of the tools and stack—the frontend, backend, libraries, storage, kernels, and physical machine—to non-shared stack.

change management

From managing small and frequent changes (small batch) to large and rare changes.

service-level objective (SLO) management → merged into **metrics, visibility & feedback**

Metrics (process, value, cost, and technical metrics) provide fast and continuous feedback from users, reduce the risk and cost of deployments, get better visibility into the delivery process itself, and manage the risks of software delivery more effectively.

Organizations are realizing the value of going beyond basic measurements for service levels and setting goals (service-level objective, SLO) that are based on meaningful metrics for the business.

alignment of dev & ops goals

From misalignment among teams that have their own interests and goals (local optimization) to alignment with business goals and global ones (product thinking). A typical example of misalignment is when "developers want to deliver as much as possible, whereas operations target stability". A typical example of alignment is when product teams and DevOps/SRE teams agree service-level objectives (SLO).

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.

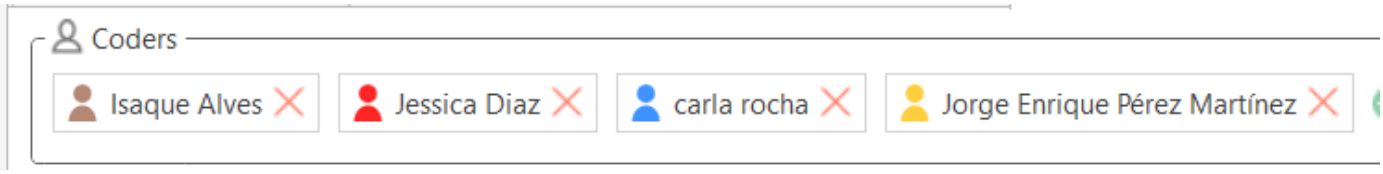
CODER 2 (Isaque Alves) - "UnifiedDevOps OpenCoding ITE4 (IA).atlproj9"

skills/knowledge sharing

From high to low levels of knowledge sharing (e.g., developers may have knowledge about infrastructure/platform, or minimal or no awareness of what is happening on the other side of the wall, aka. wall of confusion or siloization of knowledge). Knowledge sharing may include a shared repository (the source of truth) and administrative rights to different environments (testing, pre-production, production). Knowledge sharing helps build trust between Devs & Ops personnel so that the former can share/delegate tasks to the latter second, and vice versa. However, knowledge sharing also increases the cognitive load (knowledge and expertise) that some members of the team acquire since sharing promotes that these members are responsible for everything.

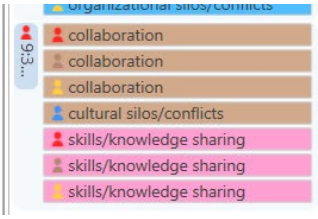
CODER3 (Jorge Pérez) - "UnifiedDevOps OpenCoding ITE4 (JP).atlproj9"
CODER4 (Carla Rocha) - "UnifiedDevOps OpenCoding ITE4 (CR).atlproj9"

ANALYSIS OF DISAGREEMENTS IN CODING
Krippendorf Cualpha = 0,792



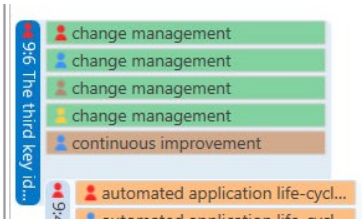
- The fact that extreme siloization of knowledge, incentives for purely local optimization, and lack of collaboration have in many cases been actively bad for business[4]

- The fact that extreme siloization of knowledge, incentives for purely local optimization, and lack of collaboration have in many cases been actively bad for business[4]

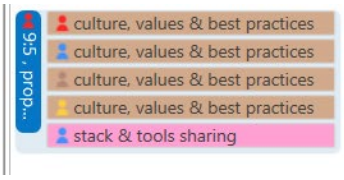


“From lack or eventual collaboration, which may generate conflicts and disagreements on decisions, to daily collaboration (working together regularly on a daily basis).”

The third key idea is that change is best when it is small and frequent. In environments where change committees meet monthly to discuss thoroughly documented plans to make changes to the mainframe configuration, this is a radical idea. However, this is not a new idea. The notion that all changes must be considered by experienced humans and batched for efficient consideration turns out to be more or less the opposite of best practice. Change is risky, true, but the correct response is to split up your changes into smaller subcomponents where possible. Then you build a steady



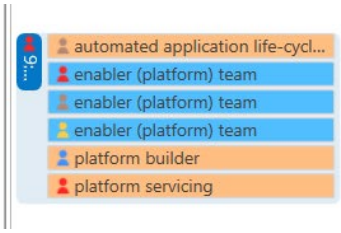
correctly—today, change management relies on highly specific tools. Overall, however, proponents of DevOps strongly emphasize organizational culture—rather than tooling—as the key to success in adopting a new way of working. A good culture can work around broken tooling, but the opposite rarely holds true. As the saying goes, culture eats strategy for breakfast. Like operations, change itself is hard.



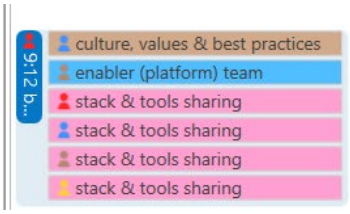
SRE does not attempt to give everything 100% availability. As discussed in our first book, *Site Reliability Engineering*, this is the wrong target for a number of reasons. Instead, the product team and the SRE team select an appropriate availability target for the service and its user base, and the service is managed to that SLO.[9] Deciding on such a target requires strong collaboration from the business. SLOs have cultural implications as well: as collaborative decisions among stakeholders, SLO violations bring teams back to the drawing board, blamelessly.

Work to Minimize Toil

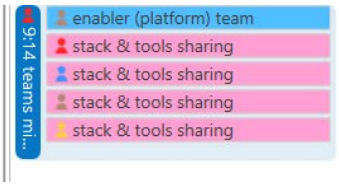
There is an unintuitive and interesting interaction between this benchmark and how it plays out when we think about automation and toil. Over time, an SRE team winds up automating all that it can for a service, leaving behind things that can't be automated (the Murphy-Beyer effect). Other things being equal, this comes to dominate what an SRE team does unless other actions are taken. In the Google environment, you tend to either add more services, up to some limit that still supports 50% engineering time, or you are so successful at your automation that you can go and do something else completely different instead.



competencies are the bread-and-butter of what SRE does for a product and for the associated product development team.[10] Ideally, both product development and SRE teams should have a holistic view of the stack—the frontend, backend, libraries, storage, kernels, and physical machine—and no team should jealously own single components. It turns out that you can get a lot more done if you "blur the lines"[11] and have SREs instrument JavaScript, or product developers qualify kernels: knowledge of how to make changes and the authority to do so are much more widespread, and

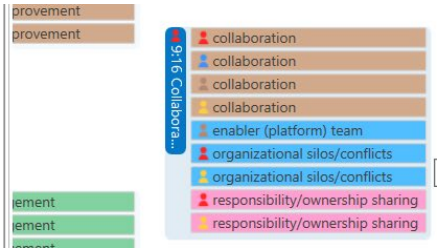


stack, and so on. Yet we share this absolute assumption with DevOps: teams minding a service[13] should use the same tools, regardless of their role in the organization. There is no good way to manage a service that has one tool for the SREs and another for the product developers, behaving differently (and potentially catastrophically so) in different situations. The more divergence you have, the less your company benefits from each effort to improve each individual tool.



- Collaboration is front and center for DevOps work. 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.

- Change management is best pursued as small, continual actions, the majority of which are ideally both automatically tested and applied. The critical interaction between



- Measurement is absolutely key to how both DevOps and SRE work. For SRE, SLOs are dominant in determining the actions taken to improve the service. Of course, you can't have SLOs without measurement (as well as cross-team debate—ideally among product, infrastructure/SRE, and the business). For DevOps, the act of measurement is often used to understand what the outputs of a process are, what the duration of feedback loops is, and so on. Both DevOps and SRE are data-oriented things, whether they are professions or philosophies.

ITERATION 5

MASTER

- 2 documents: D10 and D11
- 7 quotations
- 6 semantic domains: AUTOMATION, CULTURE, MANAGEMENT, ORGANIZATIONAL STRUCTURE, SHARING, SKILLS & ROLES
- 25 codes

CODER 1 (Jessica Diaz) - "UnifiedDevOps OpenCoding ITE5 (JD).atlproj9"
CODER 2 (Jorge Perez) - "UnifiedDevOps OpenCoding ITE5 (JP).atlproj9"

ANALYSIS OF DISAGREEMENTS IN CODING
Krippendorf Cualpha = 0,956