Detecting and Resolving Bad Organisational Smells for Microservices

Michele Agostini, Jacopo Soldani[a] and Antonio Brogi[b]
Department of Computer Science, University of Pisa, Italy

Keywords: Microservices, DevOps, Bad Smells, Organizational Smells, Refactoring.

Abstract: The development and maintenance of microservices should be decentralised. The microservices in an application should be partitioned among DevOps teams so to reduce cross-team interactions, which are costly and slow the delivery of updates. To this end, this paper identifies three *bad organisational smells* for microservices, which may possibly denote decentralisation lapses in DevOps team assignments for microservice applications, together with the organisational refactorings allowing to resolve them. We then introduce a model-driven method to automatically detect and resolve bad organisational smells in a microservice application. The proposed method is based on extending μTOSCA, an existing metamodel for specifying microservice applications, to also support modelling the DevOps team assignment of microservices. Finally, we illustrate the feasibility and usefulness of the proposed model-driven method by providing its prototype implementation and reporting on a controlled experiment, respectively.

1 INTRODUCTION

Microservice applications (MSAs) are pervading enterprise IT (Bisicchia et al., 2024). This is mainly because of their cloud-native nature (Kratzke and Quint, 2017). MSAs are indeed highly distributed, dynamic, and fault-resilient, and such peculiar aspects make MSAs capable of fully exploiting the capabilities of cloud computing (Soldani et al., 2018).

MSAs are essentially service-oriented applications that adhere to an extended set of key design principles (Zimmermann, 2017). These include decentralisation, which should occur also in the development and maintenance of the microservices forming an MSA (Newman, 2021). The microservices forming an MSA should indeed be partitioned, with different partitions assigned to different DevOps teams, so that DevOps teams can independently decide when/how to proceed when updating the microservices they own (Carrasco et al., 2018). The rationale is the following: if an update requires two or more DevOps teams to interact, it will require such DevOps teams to agree based on a cross-team interaction, hence slowing the delivery of the update and requiring cross-team budgetary approval (Lewis and Fowler, 2014).

Therefore, a key question is the following, also considering that MSAs are continuously updated and evolving over their lifetime (Soldani et al., 2021):

How to identify decentralisation issues in the DevOps team assignment for an MSA?

To answer the above question, we hereby introduce three *bad organisational smells*, together with the organisational refactorings enabling to resolve them. Inspired by bad architectural smells (Neri et al., 2020), a bad organisational smell can be observed in the DevOps team assignment for an MSA. A bad organisational smell is essentially a possible symptom of a bad decision taken when assigning the microservices forming an MSA to different DevOps teams, which *may* possibly denote decentralisation lapses in the DevOps team assignment.[1]

We also introduce a model-driven method to detect and resolve the occurrence of bad organisational smells in MSAs. More precisely, we extend μTOSCA (Soldani et al., 2021), an existing metamodel for specifying MSAs as typed topology graphs, whose nodes and arcs represent the application components forming MSAs and their interactions, respectively. The proposed extension enables modelling also the assignment of application components to DevOps teams. We then illustrate how to automatically detect the occurrence of bad organisational smells in the DevOps team assignment modelled in an

[a] https://orcid.org/0000-0002-2435-3543
[b] https://orcid.org/0000-0003-2048-2468

[1] Whilst antipatterns necessarily result in issues, smells are just symptoms deserving further investigation to determine whether any issue is truly there (Neri et al., 2020).

Agostini, M., Soldani, J. and Brogi, A.
Detecting and Resolving Bad Organisational Smells for Microservices.
DOI: 10.5220/0012851200003753

In *Proceedings of the 19th International Conference on Software Technologies (ICSOFT 2024)*, pages 67-78
ISBN: 978-989-758-706-1; ISSN: 2184-2833

extended μTOSCA topology graph, as well as how to refactor such a graph to implement the organisational refactorings for resolving the detected smells.

Finally, to assess the feasibility of our method for detecting/resolving bad organisational smells, we introduce its open-source prototype implementation. The latter is an extension of μFRESHENER, an existing tool for modelling and analysing μTOSCA topology graphs (Soldani et al., 2021). We also report on a controlled experiment assessing the usefulness of the extended μFRESHENER, where 14 ICT experts were asked to edit the μTOSCA topology graph modelling an MSA, detect the smells therein, and refactor the MSA to resolve the occurrence of detected smells. The results of the experiment show that our model-driven method successfully enabled all participants to detect and resolve all bad organisational smells they were submitted to, as well as that the proposed extension of μFRESHENER was easy to use and useful.

In summary, the main contributions of this paper are the following:

- We identify three bad organisational smells for microservices and the refactorings allowing to resolve their occurrence.
- We introduce a model-driven method to automatically detect bad organisational smells in MSAs modelled as extended μTOSCA topology graphs and to resolve detected smells.
- We provide an open-source prototype implementation of our proposed method.
- We illustrate the usefulness of the proposed method by reporting on the results of a controlled experiment.

The rest of this paper is organised as follows. Section 2 provides the necessary background. Section 3 introduces three organisational smells for MSAs, whose detection/resolution method is proposed in Section 4. Sections 5 and 6 present the new version of μFRESHENER and report on its usage in controlled experiments, respectively. Finally, Sections 7 and 8 discuss related work and draw some concluding remarks, respectively.

2 BACKGROUND

The μTOSCA type system (Figure 1) allows specifying MSAs as typed topology graphs in TOSCA (OASIS, 2020). A μTOSCA topology graph includes nodes of type Service or Database, used to model components implementing business logic or used to store business data, respectively. It can also include

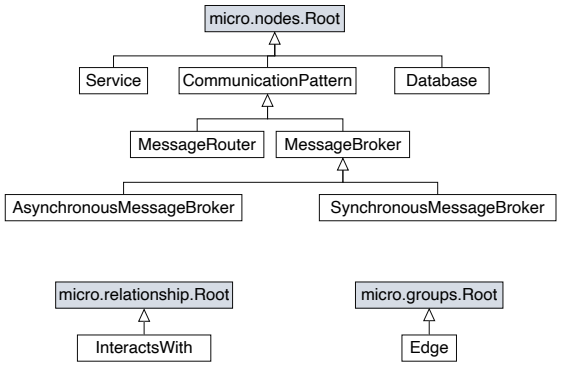


Figure 1: μTOSCA type system (Soldani et al., 2021).

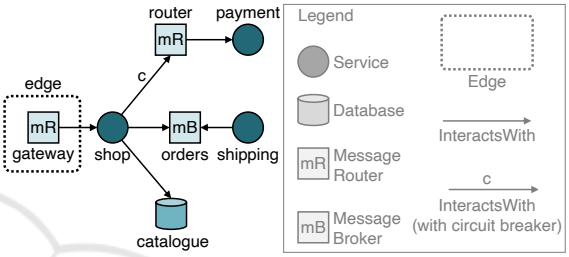


Figure 2: Example of MSA modelled with μTOSCA.

nodes modelling integration components decoupling the communication among two or more components by implementing two communication patterns defined by (Hohpe and Woolf, 2003), viz., MessageRouters or MessageBrokers. Examples of all the above components are available in the MSA displayed in Figure 2, which models a toy e-commerce application composed of a *gateway* routing external requests to the main *shop* service. The MSA in Figure 2 is completed by a database storing the *catalogue*, by two other services for the *payment* and *shipping* of orders, by a message *router*, and by a message queue used to asynchronously process to-be-shipped *orders*.

A μTOSCA topology graph can be completed by typed oriented arcs. Arcs of type InteractsWith allow to model that a source node invokes functionalities offered by a target node, e.g., *shipping* invokes *orders* to obtain to-be-shipped orders in our example (Figure 2). Arcs can also be enriched by setting the boolean properties, e.g., to indicate an interaction exploits a circuit breaker for the source component to tolerate the failure of the target component – as it happens between *shop* and *router* in our example (Figure 2).

Finally, nodes can be placed in Edge groups, to define which application components are publicly accessible from outside of the application. This allows to specify which components of an MSA can be directly accessed by external clients, without requiring to explicitly model the interaction with such external

clients. This is the case for the *gateway* in Figure 2, which is the only entry point for external clients to the toy e-commerce MSA in our example.

3 ORGANISATIONAL SMELLS

The development of MSAs should be decentralised and distributed among multiple DevOps teams, each reponsible for a separate portion of components (Neri et al., 2020). We hereafter define three organisational bad smells, i.e., *Single-Layer Team* (Section 3.1), *Tightly-Coupled Teams* (Section 3.2), and *Shared Bounded Context* (Section 3.3), by also discussing the organisational refactorings enabling to resolve them.[2] The decision on whether/how to apply a refactoring depends on the contextual information specific to a given MSA, which is accessible only to the product owners. They are indeed the only who can determine whether an organisational smell truly denotes a decentralisation issue and, if so, decide how to effectively implement an organisational refactoring.

3.1 The *Single-Layer Teams* Smell

To fully leverage the potential autonomy enabled by MSAs, the development, operation, and maintenance of microservices should be decentralised by delegating them to different DevOps teams that manage different microservices (Zimmermann, 2017). Adopting the typical splitting of teams by technology layers (e.g., frontend, backend, middleware, and database teams) is, therefore, considered an organisational bad smell. With one such organisation, any change to a microservice may require the setup of a cross-team interaction, which will take time and require budgetary approval (Lewis and Fowler, 2014).

The approach to splitting DevOps teams working on MSAs is orthogonal to the typical splitting approach. Each microservice should be indeed assigned to a DevOps team that is "cross-functional", namely composed of team members whose expertise span across all technology layers (Neri et al., 2020). This would indeed enable – in most cases – a DevOps team to independently decide when and how to proceed when applying changes to a microservice, by increasing the chances that the interactions needed for applying a change to a microservice are restricted to only the members of DevOps team owning such microservice (Carrasco et al., 2018).

In short, assigning microservices so that a DevOps team owns components pertaining to a single technology layer is considered a bad organisational smell, hereby called *Single-Layer Teams*. A *Single-Layer Teams* smell can be resolved by refactoring the assignment by *splitting teams by microservice*, rather than by technology layer (Neri et al., 2020).

3.2 The *Tightly-Coupled Teams* Smell

The microservices forming an MSA should be assigned to DevOps teams in a decentralised manner, so to possibly maximise the autonomy of teams in taking decisions on the microservices they own (Neri et al., 2020). At the same time, the microservices in an MSA are interdependent, which means that the decisions and changes applied to a microservice may impact on the microservices that depend on it (Soldani et al., 2018). If such microservices are assigned to other teams, the change under consideration will require the interested teams to agree based on a cross-team interaction, hence slowing the delivery of change/updates and possibly requiring cross-team budgetary approval (Lewis and Fowler, 2014).

The above situation is very likely to happen between two DevOps teams if the microservices they own are tightly coupled.[3] In particular, consider a microservice that is more coupled to the microservices owned by another DevOps team than to those owned by the DevOps team responsible of the microservice under consideration. In this scenario, any change or update to such microservice is likely to impact on a microservice owned by the other DevOps team, thus requiring the two DevOps teams to coordinate and collaborate in a cross-team project (Carrasco et al., 2018). Situations like that described above are hereby proposed as occurrences of the *Tightly-Coupled Teams* bad organisational smell for MSAs.

A *Tightly-Coupled Teams* smell can be resolved by changing the assignment of the microservice that is causing the smell. Such microservice should rather be assigned to the DevOps team that it is most coupled to, which is the one owning the highest number of microservices that would be impacted by a change/update to the microservice under consideration. This would actually result in enacting a sort of *inverse Conway maneuver*, as we would be changing the DevOps team organisation based on the coupling of the microservices in an MSA (Fowler, 2022).

[2] The *Single-Layer Team* smell is obtained by adapting to the organisational view the homonym architectural smell presented in (Neri et al., 2020). Instead, the *Tightly-Coupled Teams* and *Shared Bounded Context* smells are brand new and first presented in this paper.

[3] Tight coupling occurs when microservices strongly depend on each other (Ntentos et al., 2020).

3.3 The *Shared Bounded Context* Smell

One of the main tenets of MSAs is adopting domain-driven design practices to identify and conceptualise microservices (Zimmermann, 2017). Microservices should be organised around bounded contexts,[4] typically based on the business capabilities realised by an MSA, by assigning the responsibility for each bounded context to a single DevOps team (Lewis and Fowler, 2014). Instead, if multiple DevOps teams were responsible for different microservices within the same bounded context, this may lead to inter-team frictions, which may slow the delivery of functionalities and raise the overall costs (Vernon, 2016).

According to microservices and domain-driven design practices, databases should not cover more than a bounded context (Mitra and Nadareishvili, 2020). This – combined with the above consideration that different DevOps teams should not be responsible for the same bounded context – means that the microservices assigned to a DevOps team should not directly interact with a database owned by another DevOps team. If this instead happens, we may have that different DevOps teams are operating on a *Shared Bounded Context*, which we hereby propose as a bad organisational smell for MSAs. The *Shared Bounded Context* may indeed denote a domain-level violation of the decentralisation principle in the DevOps team assignment for an MSA.

An occurrence of the *Shared Bounded Context* smell can be resolved in two ways, viz., *reorganise teams by bounded contexts* or *split bounded contexts by teams*. The choice of either of the two depends on contextual information on the service and database causing the smell itself. If they pertain to the same bounded context, then the easiest solution is applying the *reorganise teams by bounded contexts* organisational refactoring, namely assigning the application components causing the smell to a single DevOps team, e.g., that owning the database. Instead, if they actually pertain to different bounded contexts, the bounded contexts should be kept separate and assigned to different DevOps teams (Buchanan, 2024). This could be realised, e.g., by splitting the shared database into two different databases storing the data pertaining to the different bounded contexts. In this case, if the services were relying on some form of consistency on shared data, the consistency should continue to be assured by relying on existing distributed consistency protocols.

[4]In domain-driven design, large domain models are partitioned into multiple different bounded contexts, each defining a specific business domain area (Vernon, 2016).

4 DETECTING AND RESOLVING ORGANISATIONAL SMELLS

To enable resolving bad organisational smells in MSAs, we hereafter first extend μTOSCA to enable modelling the assignment of microservices to DevOps teams (Section 4.1). Then, we introduce a μTOSCA-based method for automatically detecting bad organisational smells in MSAs and to reason on whether/how to refactor detected smells (Section 4.2).

4.1 Modelling DevOps Team Assignment in μTOSCA

According to (Soldani et al., 2021), an MSA can be modelled by a μTOSCA topology graph, which is formally represented by a triple containing (i) the typed nodes representing application components, (ii) the relationships forming the graph representing the architecture of an application, and (iii) the edge group, which contains the nodes that are directly accessible to external clients. To model DevOps team assignment, we hereby add a fourth element, namely (iv) a set of groups of nodes, each representing the assignment of the nodes therein to a DevOps team. The choice of using groups for modelling DevOps team assignment is motivated by the notion of TOSCA groups themselves, which are natively intended to group application components to be managed together (OASIS, 2020).

Definition 1 (MSA)**.** *Let L be the set of properties that can hold onto an interaction*[5] *and let T denote the universe of possible team names. An* MSA *is represented by a quadruple $\langle N,R,E,A\rangle$, where*

- *(i) N is a finite set of typed nodes representing the components forming an application,*
- *(ii) $R \subseteq N \times N \times 2^L$ is a finite set of triples, each denoting a typed interaction between two components and the properties holding on such interaction,*
- *(iii) $E \subseteq N$ is a set of nodes defining the components forming the edge of the architecture, and*
- *(iv) $A \subseteq T \times 2^N$ is a finite set of pairs, each denoting the assignment of a set of components to a team.*

The assignment defined in an extended μTOSCA topology graph is well-formed when different teams own different components.

[5]Properties can be set on interactions to indicate that, e.g., circuit breakers are used therein (such as between *shop* and *router* in Figure 2). Further information on interactions' properties can be found in (Soldani et al., 2021).

Definition 2 (Well-Formed Team Assignment)**.** *Let* $\langle N,R,E,A\rangle$ *be an MSA. The team assignment* A *is* well-formed *iff*

$$\forall\langle t,C\rangle,\langle t',C'\rangle \in A : t \neq t' \Rightarrow C \cap C' = \emptyset.$$

For simplicity, and without loss of generality, we hereafter assume the DevOps team assignment modelled in a μTOSCA topology graph to be well-formed. We also assume each team assignment to be complete, namely such that (i) each application component is assigned to a DevOps team and that (ii) each team manages at least one application component.

Definition 3 (Complete Team Assignment)**.** *Let* $\langle N,R,E,A\rangle$ *be an MSA. The team assignment* A *is* complete *iff*

(i) $\forall n \in N \exists\langle t,C\rangle \in A : n \in C$*, and*

(ii) $\forall\langle t,C\rangle \in A : C \neq \emptyset$.

An example of a well-formed and complete assignment of application components to DevOps teams is displayed in Figure 3. The figure shows a toy MSA, whose application components are partitioned among the *blue*, *green*, *orange*, and *yellow* DevOps teams. For instance, the *yellow* team owns the *frontend* of the MSA, the *catalog* service, and the *items* database.

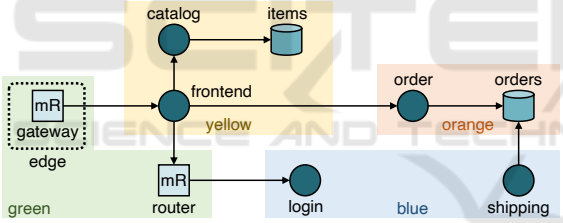


Figure 3: Running example. DevOps team assignment is represented by background colour boxes.

The DevOps team assignment in Figure 3 is devised on purpose to exhibit all the bad organisational smells proposed in Section 3. We shall, therefore, use it as a running example to showcase the proposed smell detection method, as well as the smell resolution via refactoring.

4.2 Model-Driven Detection and Resolution of Organisational Smells

We hereby illustrate how to automatically detect the bad organisational smells defined in Section 3 in μTOSCA topology graph modelling an MSA. We also enable reasoning on whether/how to refactor the MSA modelled by a μTOSCA topology graph to resolve the occurrence of detected smells.

4.2.1 Single-Layer Teams

A *Single-Layer Teams* smell occurs when a DevOps team owns components pertaining to a single technology layer (Section 3.1). In a μTOSCA topology graph, this situation is denoted by a DevOps team assigned with components all of the same type – or of compatible types (e.g., a DevOps team owning only MessageBrokers and MessageRouters, which are all implementing CommunicationPatterns).

Notation 1 (Node Types)**.** *Let* $\langle N,R,E,A\rangle$ *be an MSA. We denote by* $\mathcal{T} = \{$Service, Database, CommunicationPattern, MessageRouter, MessageBroker, AsynchronousMessageBroker, SynchronousMessageBroker$\}$ *the set of possible types for a node* $n \in N$*, and we write* $n.$type *to denote the type of node* n*. We also write* $x.\text{type} \geq \tau$ *to denote that the type of* x *extends or is equal to the type* $\tau \in \mathcal{T}$.

Definition 4 (Single-Layer Team Smell)**.** *Let* $\langle N,R,E,A\rangle$ *be an MSA. A DevOps team assignment* $\langle t,C\rangle \in A$ *denotes a* Single-Layer Team *smell iff*

$$\exists\tau \in \mathcal{T} : \forall c \in C : c.\text{type} \geq \tau.$$

Our running example (Figure 3) provides an example of *Single-Layer Teams* smell, with the *green* team owning only MessageRouters. This may suggest that the *green* team is not cross-functional, but rather composed of middleware experts only – who actually handle all the middleware components in the MSA considered in our running example.

Our choice here is to refactor the DevOps team assignment to resolve the occurrence of the detected smell. We hence implement the *splitting teams by microservice* refactoring described in Section 3.1. In this case, the refactoring consists of changing the responsible teams for the *frontend* service, so that the *green* team owns not only the *gateway*, but the whole user-facing part of the MSA in our running example (Figure 4). Correspondingly, the *green* DevOps team must adapt to now hold the expertise to also handle services, if this was not yet the case.

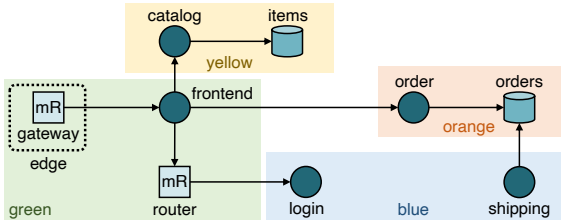


Figure 4: Running example refactored to resolve the *Single-Layer Team* smell.

4.2.2 Tightly-Coupled Teams

A *Tightly-Coupled Teams* smell occurs when an application component is more coupled to the application components owned by another DevOps team than to those owned by the DevOps team responsible for the component itself (Section 3.2).

In a μTOSCA topology graph, this situation is denoted by a DevOps team t owning a node c that is coupled more to another DevOps team t' than to t. To formalise this, we first introduce some shorthand notation to generalise the notion of coupling from node-to-node to node-to-team. For simplicity, we shall use the number of interactions to measure node coupling, but other existing metrics – such as those in (Ntentos et al., 2020) – can be used for the same purpose.

Notation 2. *Let $\langle N,R,E,A\rangle$ be an MSA, let $n \in N$ be a node, and let $\langle t,C\rangle \in A$ be a team assignment. We denote by $\gamma(n,t)$ the coupling degree of a node n to a team t, defined as*

$$\gamma(n,t) = \#\{\langle n,c,\_\rangle, \langle c,n,\_\rangle \in R \mid c \in C\}$$

Definition 5 (Tightly-Coupled Teams Smell)**.** *Let $\langle N,R,E,A\rangle$ be an MSA. A DevOps team assignment $\langle t,C\rangle \in A$ denotes a* Tightly-Coupled Teams *smell iff*

$$\exists c \in C, \langle t',C'\rangle \in A : \gamma(n,t') > \gamma(n,t)$$

The *login* service in Figure 4 provides an example of *Tightly-Coupled Teams*, as *login* is coupled more to the components owned by the *green* DevOps team than to those owned by the *blue* one, which is the responsible team for *login*. This increases the probability of cross-team interactions between the *green* and *blue* teams, as these might probably be required when updating, e.g., the *login* service.

Considering what is above, we opt for resolving the *Tightly-Coupled Teams* smell occurrence under consideration. To this end, we apply an *inverse Conway maneuver* to *login* (Section 3.2). More precisely, we change the responsible DevOps team for the *login* service, by assigning it to the team it is more coupled to, i.e., the *green* team (Figure 5).

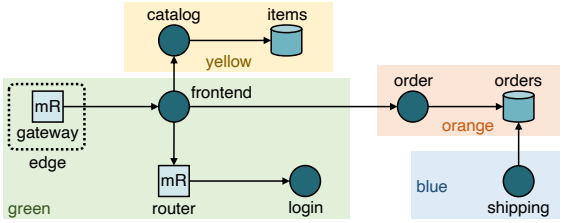


Figure 5: Running example refactored to resolve the *Tightly-Coupled Teams* smell.

4.2.3 Shared Bounded Context

A *Shared Bounded Context* smell occurs when a service assigned to a DevOps team interacts with a database owned by another DevOps team (Section 3.3). In μTOSCA, this situation is denoted by a cross-team relationship, which models a direct interaction from a Service node owned by a team t to a Database node owned by another team t'.

Definition 6 (Shared Bounded Context Smell)**.** *Let $\langle N,R,E,A\rangle$ be an MSA. A couple of DevOps team assignments $\langle t,C\rangle, \langle t',C'\rangle \in A$ (with $t \neq t'$) denote a* Shared Bounded Context *smell iff*

$$\exists c \in C, c' \in C' : \langle c,c',\_\rangle \in R \wedge$$
$$c.\text{type} \geq \text{Service} \wedge c'.\text{type} \geq \text{Database}$$

The InteractsWith relationship from *shipping* to *orders* in Figure 5 provides an example of *Shared Bounded Context* smell. The *shipping* service is indeed owned by the *blue* DevOps team, and it directly interacts with the *orders* database owned by the *orange* DevOps team. Any update to the *orders* database may impact the *shipping* service, hence possibly requiring cross-team interactions between the *blue* and *orange* DevOps teams.

Our choice is to resolve the occurrence of the smell, but this time we have to decide which of the two possible organisational refactorings to apply (Section 3.3). In this case, given that the *blue* team owns only the *shipping* service, and since the shipping of orders pertains to the bounded context of order management, we decide to *reorganise teams by bounded context*. In particular, we change the responsible team for the *shipping* service by assigning it to the *orange* team (Figure 6).

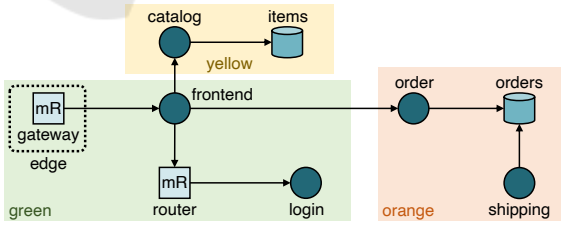


Figure 6: Running example refactored to resolve the *Shared Bounded Context* smell.

As a result, no more bad organisational smells affect the obtained DevOps team assignment (Figure 6). Additionally, the *blue* DevOps team has disappeared, which means that we can redistribute its corresponding resources to other DevOps teams working on the considered MSA or on other projects.

5 PROTOTYPE IMPLEMENTATION

To assess the feasibility of the proposed method for detecting and resolving bad organisational smells, we prototyped into a new version of μFRESHENER (Soldani et al., 2021), hereby called μFRESHENER v2.[6] Firstly, we enabled distinguishing between the *product owner* and *team member*s while editing the μTOSCA topology graph modelling an MSA, as well as while detecting and resolving bad smells for microservices. We hereafter illustrate how μFRESHENER v2 enables visualising and editing MSAs – and resolving bad smells therein – by distinguishing the two newly supported user roles, i.e., *product owner* and *team member*.

5.1 Product Owner Role

In μFRESHENER v2, the *product owner* has full power over the whole μTOSCA topology graph modelling an MSA. Therefore, beyond editing, analysing, and refactoring the application components and interactions forming an MSA, the *product owner* is the only user who can edit the DevOps team assignment. To this end, we implemented the automated detection of the bad organisational smells presented in Section 4, as well as the possibility of applying known refactorings to resolve their occurrence. For instance, if a *Tightly-Coupled Teams* smell is detected, the *product owner* can ask μFRESHENER v2 to apply the *inverse Conway maneuver*. As a result, the application component denoting the considered *Tightly-Coupled Teams* smell is automatically re-assigned to the DevOps team to which it is more coupled.

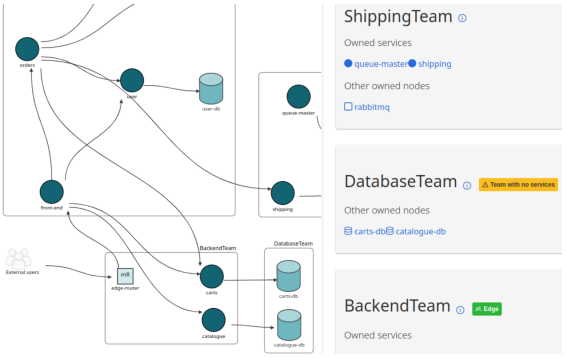
Additionally, to further support the *product owner* in managing DevOps team assignment, we also implemented multiple different views over the current DevOps team assignment, e.g., showing details on the types of services owned by DevOps teams or plotting figures about inter-team dependencies (Figure 7).

5.2 Team Member Role

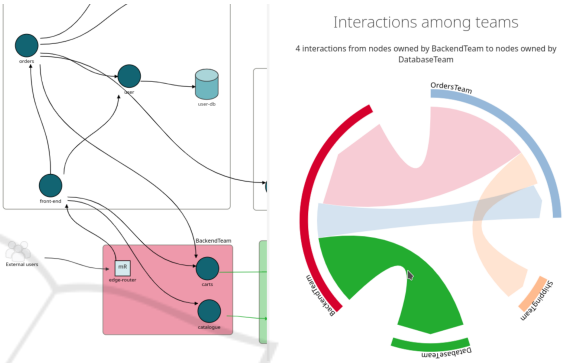
The view and editing capabilities of team members are instead restricted to only the application components owned by the corresponding DevOps team.

More precisely, when accessing as team members, μFRESHENER v2 displays only the portion of the MSA that is owned by the logged DevOps team. This is to enable team members to focus their visual thinking only on the components they actually own.

[6] μFRESHENER v2 is publicly available online at https://github.com/di-unipi-socc/microFreshener.

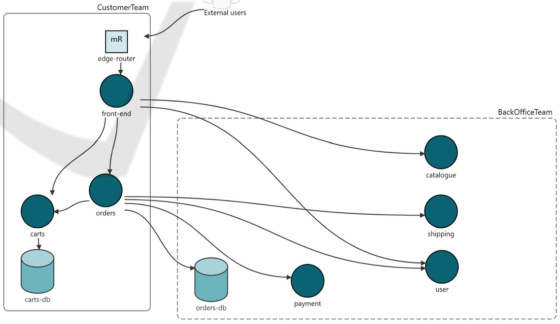


(a)

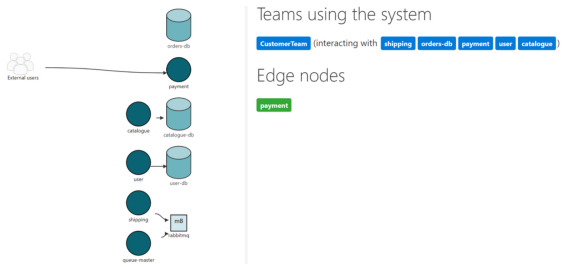


(b)

Figure 7: Informative views offered by μFRESHENER to *product owner*s on (a) nodes assigned to DevOps teams and (b) cross-team interactions.



(a)



(b)

Figure 8: Teamwise visualisation of (a) outgoing and (b) ingoing cross-team dependencies in μFRESHENER.

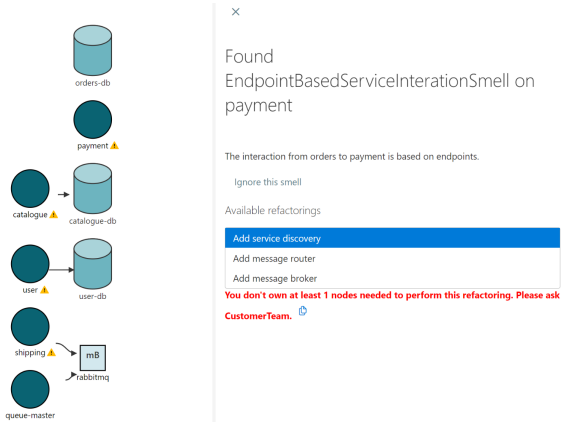


Figure 9: Prevention and alerting of changes requiring interactions with other DevOps teams.

Outgoing and ingoing cross-team interactions are also displayed differently to the members of a logged team, given that they have different meanings. Outgoing cross-team interactions indicate the dependencies of the logged DevOps team on other DevOps teams, and they are directly displayed on the editing pane (by showing only the target DevOps teams' components on which the logged team depends – Figure 8a). Instead, ingoing cross-team interactions indicate the dependencies that other DevOps teams have on the components owned by the logged DevOps team, being them the "users" of the business capabilities managed by the logged team. This is similar to the case of end-users invoking the functionalities offered by the application components exposed externally. For this reason, ingoing cross-team interactions and edge nodes – if any – are displayed together in a "users" sidebar (Figure 8b).

Team members can also edit the portion of the MSA they own. For instance, they can add new components and relationships to the MSA, with newly added components automatically assigned to the logged DevOps team. They can also remove or refactor existing components and relationships, e.g., to resolve bad architectural smells. This is supported by μFRESHENER v2 as long as the actions of a logged DevOps team do not interfere with the operativity of other DevOps teams. In particular, μFRESHENER prevents actions that would result in changing interactions starting from other teams' components, rather displaying an alert informing the logged team to interact with the other involved DevOps teams to perform the desired change. This holds also when exploiting μFRESHENER's refactoring capabilities for resolving bad architectural smells, as shown in Figure 9.

6 EVALUATION

To assess the usefulness of our proposal, we run a controlled experiment in which we asked participants to use μFRESHENER v2 to edit the μTOSCA specification of a given MSA, detect the bad smells therein, and refactor the MSA to resolve the occurrence of the detected bad smells.

6.1 Experiment Design

The experiment was run in two steps, with participants asked to first (1) take the role of *product owner*s and then (2) that of *team member*s.

As the *product owner* is the only role capable of managing DevOps team assignment, we exploited the first step to assess the proposed modelling of DevOps team assignment and the proposed method for detecting and resolving bad organisational smells. To this end, step 1 was designed as a sequence of five tasks: ($T_{1.1}$) model a service and an interaction, ($T_{1.2}$) change the ownership of an application component, ($T_{1.3}$) add a new DevOps team owning and assign it five application components, ($T_{1.4}$) create a new DevOps team, assign it all components owned by another DevOps team, and delete the latter team, and ($T_{1.5}$) run the automated smell detection and resolve a detected *Single-Layer Teams* smell.

Step 2, instead, focused on exploiting the DevOps team assignment to assess the usefulness of μFRESHENER v2 also from the perspective of *team members*. For this reason, step 2 was designed as a sequence of seven tasks: ($T_{2.1}$) model interactions between nodes owned by the DevOps team and towards nodes owned by other DevOps teams, ($T_{2.2}$) replace a component owned by the DevOps team with a new component and model its interactions with components owned by the same DevOps team, ($T_{2.3}$) delete a database that is used by a service owned by another DevOps team, ($T_{2.4-5}$) resolve two automatically detected architectural smells, both impacting only on components owned by the DevOps team, and ($T_{2.6}$) resolve an automatically detected architectural smell by refactoring a cross-team interaction. Differently from step 1, the tasks of step 2 were designed so that all but two could be successfully completed. Indeed, $T_{2.2}$ and $T_{2.6}$ could not be completed when acting as *team member*s, since they require to apply architectural refactorings that impact on application components owned by other DevOps teams.

6.2 Experiment Execution

The two steps described above were run by 14 ICT experts: eight practitioners working in the ICT industry (four holding an MSc in Computer Science, and three holding a BSc in Computer Science) and six coming from academic institutions (one holding a PhD in Computer Science, two holding an MSc in Computer Science, and three holding a BSc in Computer Science).

We remotely met all participants to collect information on their background and experience in Computer Science, and to provide them with a 5-minutes introduction to the experiment and to μFRESHENER v2. Participants were then proceeding autonomously with the experiment, by connecting to a running instance of μFRESHENER v2 deployed on AWS EC2. They were guided in the experiment – which took approximately 30 minutes on average – by an online questionnaire with gamification elements. The questionnaire stated the tasks to run by also asking the participants to mark them as *completed* or *not completed*, with the latter being the expected answer for $T_{2.2}$ and $T_{2.6}$ (as μFRESHENER v2 should prevent *team member*s to complete them, since their completion requires cross-team interactions). At the end of each step, the questionnarie also asked participants to evaluate whether/how μFRESHENER v2 truly helped them, by indicating their agreement with a set of given statements on a Likert scale from 1 (*strongly disagree*) to 5 (*strongly agree*).

6.3 Results

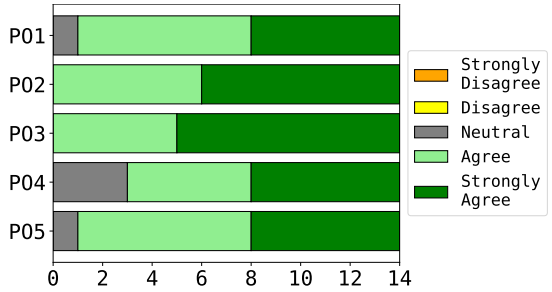
The answers collected from all 14 participants are accessible online.[7] Despite limited to 14 participants, the results of our experiment already provide insights on the usefulness and ease of use of our proposal. All tasks were indeed effectively run by 100% of participants, but for the case of $T_{2.1}$ and $T_{2.5}$, which were successfully completed by 64% and 93% of participants, respectively. The average success rate for task is, therefore, 100% when acting as *product owner*, namely when experimenting our proposed method for modelling DevOps team assignment and resolving bad organisational smells.

Still on the *product owner* side, we achieved quite good results also when assessing the user experience of running the given task with the proposed μFRESHENER v2. Figure 10a shows the statements submitted to participants to assess whether they agree on the fact that – as *product owners* – μFRESHENER v2 was (PO1) easy to use and (PO2-5) useful. In addition to the direct statement in PO5, the usefulness of μFRESHENER was assessed also based on PO2's outcomes and by asking (PO3-4) whether μFRESHENER simplified detecting bad organisational smells and reasoning on their resolution. As shown in Figure 10b, almost all the participants agreed that the

[7] https://zenodo.org/records/10952834

ID	Statement
PO1	I found μFRESHENER easy to use
PO2	I find the MSA refactored by μFRESHENER better than the one preceding the refactoring
PO3	μFRESHENER simplified the detection of the bad smells affecting the considered MSA (with respect to doing it manually)
PO4	μFRESHENER simplified the reasoning on the resolution of the bad smells affecting the considered MSA (with respect to doing it manually)
PO5	I found μFRESHENER useful

(a)

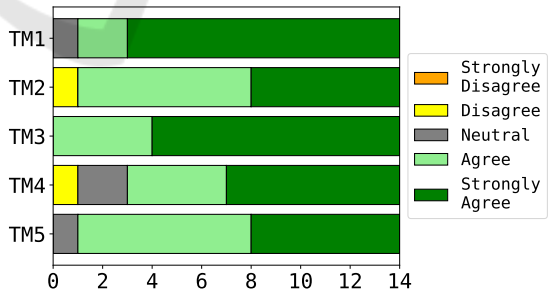


(b)

Figure 10: Evaluation of the new version of μFRESHENER as *product owner*: (a) statements and (b) agreement.

ID	Statement
TM1	I found μFRESHENER easy to use
TM2	I find the architecture refactored by μFRESHENER better than the one preceding the refactoring
TM3	μFRESHENER simplified the detection of the bad smells affecting the considered MSA (with respect to doing it manually)
TM4	μFRESHENER simplified the reasoning on the resolution of the bad smells affecting the considered MSA (with respect to doing it manually)
TM5	I found μFRESHENER useful

(a)



(b)

Figure 11: Evaluation of the new version of μFRESHENER as *team member*: (a) statements and (b) agreement.

modelling, detection, and resolution of bad organisational smells implemented in μFRESHENER v2 is easy to use and useful.

Similar results were achieved in the assessment of the user experience of participants while acting as *team member*s. We submitted them analogous statements (Figure 11a), however asking them to indicate their agreement by considering only the tasks they run in the second step of the experiment. Again, it turned out that participants agreed on the ease of use of μFRESHENER v2 and its usefulness (Figure 11b).

7 RELATED WORK

Various existing studies classify bad smells for microservices, as well as the refactorings allowing to resolve the occurrence of such bad smells in MSAs. For instance, (Carrasco et al., 2018) and (Taibi and Lenarduzzi, 2018) first elicited bad *architectural* smells for MSAs. (Neri et al., 2020) extends the set of known bad *architectural* smells by also eliciting the architectural refactorings known to resolve their occurrence. (Ponce et al., 2022b) instead elicits the bad *security* smells for microservices, along with the security refactorings for resolving their occurrence.

Based on the existing taxonomies of bad smells for MSAs, different methods and tools for their detection and resolution have been proposed. As for *architectural* smells, we already mentioned the possibility of detecting and resolving them in existing MSAs with μFRESHENER (Soldani et al., 2021). Arcan (Arcelli Fontana et al., 2017) and Designite (Sharma et al., 2016) are two other tools for detecting bad smells in software systems, which have been adapted to enable detecting bad *architectural* smells in MSAs in (Pigazzini et al., 2020) and (Capilla et al., 2023).

(Dell'Immagine et al., 2023), (Howard-Grubb et al., 2023), and (Wizenty et al., 2023) instead allow detecting bad *security* smells in MSAs. (Wizenty et al., 2023) actually also allows to resolve a subset of the detected smells by applying the security refactorings known to resolve them. The resolution of detected smells can also prioritised by relying on the triage methodology proposed in (Ponce et al., 2022a), which essentially combines the importance of security and other quality attributes for an MSA with the known impacts of security smells on such quality attributes (Ponce et al., 2023).

However, to the best of our knowledge – and also according to the tertiary study in (Cerny et al., 2023) – no support for detecting and resolving bad *organisational* smells is currently available. This is, therefore, the main novelty of our proposal: we introduce a first support to model DevOps team assignment, analyse modelled MSAs to automatically detect bad organisational smells, and reason on whether/how to refactor DevOps team assignment to resolve detected smells.

The *organisational* aspects are however crucial in MSAs (Soldani et al., 2018; Zimmermann, 2017). This is also witnessed by the recently proposed studies on the topic. For instance, (D'Aragona et al., 2023) analysed the actual usage of the *microservice-per-developer* pattern in open-source projects, observing that this is rarely the case, except for MSAs developed by dedicated DevOps teams. (Li et al., 2023) and (Zabardast et al., 2023) instead focus on monitoring the DevOps team assignment for measuring cross-team coupling and accumulated technical debt, respectively. Our proposal complements the above studies by providing a first support to detect and resolve bad organisational smells in MSAs.

8 CONCLUSIONS

We identified three bad organisational smells, which may denote decentralisation lapses in the organisation of MSAs, together with the organisational refactorings allowing to resolve their occurrence. We also introduced a model-driven method to automatically detect the identified bad organisational smells in the DevOps team assignment of an MSAs, modelled as an extended μTOSCA topology graph, and to resolve detected smells by applying their corresponding organisational refactoring. Finally, we illustrated the feasibility and usefulness of the proposed method by implementing it as an extension of μFRESHENER and by reporting on a controlled experiment, where 14 ICT experts were asked to use the extended version of μFRESHENER to model the DevOps team assignment of an MSA and to detect and resolve the bad organisational smells therein.

The three bad organisational smells introduced in this paper (and their refactorings) provide a first reference set, elicited from existing literature, but further investigation is needed to better cover the state-of-the-art and state-of-practice on the topic. For this reason, we plan to extend the set of bad organisational smells and refactorings for microservices, e.g., by running a multivocal literature review to elicit them, as done by (Neri et al., 2020) and (Ponce et al., 2022b) for other types of bad smells for microservices.

We also plan to accordingly extend our model-driven method to detecting/resolving bad organisational smells and its implementation in μFRESHENER. More broadly, we plan to further support product owners and DevOps teams in taking more informed

decisions, by relating the DevOps team assignment modelled in μTOSCA with other metadata gathered from other sources, e.g., activities recorded by version control systems, cognitive load estimated on a project's source code, or information on the DevOps team members themselves.

ACKNOWLEDGEMENTS

This work was partly funded by the following projects: *FREEDA* (CUP: I53D23003550006), funded by the frameworks PRIN (MUR, Italy) and Next Generation EU; *OSMWARE* (UNIPI_PRA 2022 64), funded by the University of Pisa, Italy.

REFERENCES

Arcelli Fontana, F., Pigazzini, I., Roveda, R., Tamburri, D. A., Zanoni, M., and Di Nitto, E. (2017). Arcan: A tool for architectural smells detection. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 282–285.

Bisicchia, G., Forti, S., Pimentel, E., and Brogi, A. (2024). Continuous QoS-compliant orchestration in the cloud-edge continuum. *Software: Practice and Experience*. In press.

Buchanan, I. (2024). Team topologies: How four fundamental topologies influence a devops transformation. Atlassian, https://www.atlassian.com/devops/frameworks/team-topologies.

Capilla, R., Arcelli Fontana, F., Mikkonen, T., Bacchiega, P., and Salamanca, V. (2023). Detecting architecture debt in micro-service open-source projects. In *49th Euromicro Conference on Software Engineering and Advanced Applications, (SEAA 2023)*, pages 394–401. IEEE.

Carrasco, A., van Bladel, B., and Demeyer, S. (2018). Migrating towards microservices: Migration and architecture smells. In *Proceedings of the 2nd International Workshop on Refactoring*, IWoR 2018, pages 1–6. ACM.

Cerny, T., Abdelfattah, A. S., Al Maruf, A., Janes, A., and Taibi, D. (2023). Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study. *Journal of Systems and Software*, 206:111829.

D'Aragona, D. A., Li, X., Cerny, T., Janes, A., Lenarduzzi, V., and Taibi, D. (2023). One microservice per developer: Is this the trend in OSS? In Papadopoulos, G. A. et al., editors, *Service-Oriented and Cloud Computing - 10th IFIP WG 6.12 European Conference (ESOCC 2023)*, volume 14183, pages 19–34. Springer.

Dell'Immagine, G., Soldani, J., and Brogi, A. (2023). KubeHound: Detecting microservices' security smells in Kubernetes deployments. *Future Internet*, 15(7).

Fowler, M. (2022). Conway's law. https://martinfowler.com/bliki/ConwaysLaw.html.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 1st edition.

Howard-Grubb, T., Soldani, J., Dell'Immagine, G., Arcelli Fontana, F., and Brogi, A. (2023). Smelling homemade crypto code in microservices, with kubehound. In Monti, F. et al., editors, *Service-Oriented Computing - ICSOC 2023 Workshops*, LNCS, pages 317–324. Springer.

Kratzke, N. and Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. *Journal of Systems and Software*, 126:1–16.

Lewis, J. and Fowler, M. (2014). Microservices: A definition of this new architectural term. https://martinfowler.com/articles/microservices.html.

Li, X., D'Aragona, D. A., and Taibi, D. (2023). Evaluating microservice organizational coupling based on cross-service contribution. In Kadgien, R. et al., editors, *Product-Focused Software Process Improvement - 24th International Conference (PROFES 2023*, volume 14483 of *LNCS*, pages 435–450. Springer.

Mitra, R. and Nadareishvili, I. (2020). *Microservices: Up and Running*. O'Reilly Media, Inc.

Neri, D., Soldani, J., Zimmermann, O., and Brogi, A. (2020). Design principles, architectural smells and refactorings for microservices: A multivocal review. *SICS Software-Intensive Cyber-Physical Systems*, 35:3–15.

Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2nd edition.

Ntentos, E., Zdun, U., Plakidas, K., Meixner, S., and Geiger, S. (2020). Assessing architecture conformance to coupling-related patterns and practices in microservices. In Jansen, A. et al., editors, *Software Architecture*, pages 3–20, Cham. Springer International Publishing.

OASIS (2020). TOSCA Simple Profile in YAML, version 1.3. OASIS Standard.

Pigazzini, I., Arcelli Fontana, F., Lenarduzzi, V., and Taibi, D. (2020). Towards microservice smells detection. In *Proceedings of the 3rd International Conference on Technical Debt (TechDebt 2020)*, page 92–97. ACM.

Ponce, F., Soldani, J., Astudillo, H., and Brogi, A. (2022a). Should microservice security smells stay or be refactored? towards a trade-off analysis. In Gerostathopoulos, I. et al., editors, *Software Architecture - 16th European Conference, ECSA 2022*, volume 13444 of *LNCS*, pages 131–139. Springer.

Ponce, F., Soldani, J., Astudillo, H., and Brogi, A. (2022b). Smells and refactorings for microservices security: A multivocal literature review. *Journal of Systems and Software*, 192:111393.

Ponce, F., Soldani, J., Taramasco, C., Astudillo, H., and Brogi, A. (2023). To security and beyond: On the impacts of microservice security smells and refactorings. In *XLIX Latin American Computer Conference (CLEI 2023)*, pages 1–10.

Sharma, T., Darraji, R., and Ghannouchi, F. (2016). Design methodology of high-efficiency contiguous mode harmonically tuned power amplifiers. In *2016 IEEE Radio and Wireless Symposium (RWS)*, pages 148–150.

Soldani, J., Muntoni, G., Neri, D., and Brogi, A. (2021). The μTOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures. *Software: Practice and Experience*, 51(7):1591–1621.

Soldani, J., Tamburri, D. A., and Van Den Heuvel, W. J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232.

Taibi, D. and Lenarduzzi, V. (2018). On the definition of microservice bad smells. *IEEE Software*, 35(3):56–62.

Vernon, V. (2016). *Domain-Driven Design Distilled*. Addison-Wesley.

Wizenty, P., Ponce, F., Rademacher, F., Soldani, J., Astudillo, H., Brogi, A., and Sachweh, S. (2023). Towards resolving security smells in microservices, model-driven. In Fill, H. G. et al., editors, *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, pages 15–26.

Zabardast, E., Gonzalez-Huerta, J., Palma, F., and Chatzipetrou, P. (2023). The impact of ownership and contribution alignment on code technical debt accumulation. *CoRR*, abs/2304.02140.

Zimmermann, O. (2017). Microservices tenets. *Computere Science: Research and Development*, 32(3–4):301–310.