Computing (2025) 107:102
https://doi.org/10.1007/s00607-025-01454-7

SPECIAL ISSUE ARTICLE

Exploring microservice ownership and organizational coupling in open-source projects: an empirical study

Xiaozhou Li[1] · Dario Amoroso d'Aragona[2] · Tomas Cerny[3] · Valentina Lenarduzzi[1] · Davide Taibi[1] · Andrea Janes[4]

Received: 20 April 2024 / Accepted: 3 March 2025
© The Author(s) 2025

Abstract

Together with the rising popularity of microservices, practitioners have started to pay more attention to the longevity and sustainability of microservices on the organizational level. Many argue—supported by Conway's Law—that each microservice should be owned by a specific team or individual developer, although each team or developer may only contribute to a maximum of one microservice. With such a "one microservice per team/developer" strategy, the architecture shall be better maintained in a fashion of "high cohesion, low coupling", which is a recommended setting for modular software systems. However, it is difficult to achieve such ideal circumstances when the coupling phenomena commonly exist therein, which are caused by cross-service calls and dependencies in microservice-based systems. Especially besides the couplings in functions and modules latent in the source code, it is also noticeable that the microservice project teams can suffer from high coupling issues regarding their cross-service contribution. Such an issue on the organizational level can inevitably result in technical debt and high managerial costs, which need to be detected and mitigated in time to prevent future losses. In this paper, we investigate the existence of the recommended "one microservice per team/developer" setting via an empirical study on 38 microservice-based open-sourced projects together with the different developer roles who are in charge of single or multiple microservices. Furthermore, by taking into account microservice ownership and cross-service contribution, we also investigate the organizational coupling of these projects and the developer roles that contribute to such couplings. The results show that it is rare that such "one microservice per team/developer" phenomenon exists in microservice projects, while the couplings are majorly caused by the leaders and major contributors of the microservices.

Keywords Microservice · Team · Coupling · Cross-service contribution · Service ownership

Extended author information available on the last page of the article

Published online: 01 April 2025

Mathematics Subject Classification 68-02

1 Introduction

Microservices are small and autonomous services deployed independently, with a single and clearly defined purpose [1, 2]. Their independent deployment allows each microservice to be scaled on its own. Some researchers argue that the advantage of microservices extends beyond technology to facilitate the scaling of development teams [3]. They point out that microservices enable the management of larger teams by reducing the need for inter-team communication, thanks to the decoupling of software components.

It is commonly recommended that each microservice should be under the stewardship of a single team [3–9]. This notion is supported by Conway's Law [10], which suggests that the structure of systems designed by an organization will mirror the organization's communication structures. Consequently, assigning more than one team to a microservice could induce the teams to split the microservice; also, assigning one team to more than one microservice could induce the team to introduce coupling between the two services [6, 7]. As such, the consensus is that each team should oversee one or several specific business functions [7, 9, 11]. While some believe a team could manage multiple microservices or subsystems, there is a prevailing view that a team should focus on a single service to avoid overwhelming its cognitive load, especially when dealing with significant business functions like customer or order management [8].

Following the guidance from experts and Conway's Law [7–11], it is recommended that developers be part of only one team, with each team contributing to a single microservice. This approach implies that each developer should work exclusively on one microservice. This paper explores the extent to which the "one microservice per team/developer" concept is applied in practice, particularly in open-source software (OSS) projects employing microservices.

In OSS projects, which often follow the bazaar-style development model described by [12], where contributors work on various parts of the system much like vendors in a bazaar, a decoupled software architecture is crucial. This paper posits that in such projects with a microservice architecture, developers typically commit to only one microservice within a specific timeframe.

The lack of "one microservice per team/developer" adoption results in an issue of coupling in terms of the software organizational structure. Similar to the concept of coupling within the source code and between the modules, the coupling between teams, especially for the teams that are supposed to be solely responsible for one microservice, is one of the obstacles to optimized organizational structure and the development and maintenance efficiency as well as the software quality [13]. As the principle of "high cohesion, low coupling" is well acknowledged across the software engineering domain, it is also required for microservice projects to follow such a principle [14]. After all, organization-related issues and technology issues are usually equally critical to the software quality [15]. There are a limited number of studies focusing on proposing metrics and assessing methods for coupling in general

for microservices [16, 17]. Regarding assessing the coupling on the organizational level, Li et al. propose a metric to assess the coupling on the organizational structure level for microservice projects, named *organizational coupling*, based on the developers' contribution across microservices. Despite the study being validated through a case study, it is still unknown what is the situation of organizational coupling existing in popular OSS microservice-based projects.

This study is an extension of our previous work [18]. Herein, we further collected the new data for the selected microservice-based OSS projects from Baresi et al.'s dataset [4] towards the reconfirmation of the results in the previous work. The work continues targeting the investigation on the adoption of the "one microservice per team/developer" strategy in OSS microservice projects, as well as the latent roles of the developers who are taking charge in or contributing to multiple services. Furthermore, we adopted the approach of measuring the organizational coupling proposed by Li et al. on the selected 38 projects investigating how strongly the services are coupled on the organizational level [19]. Also, we investigated the different technical roles of developers who contribute to the organizational coupling. The work further investigates the organizational coupling issues within microservice projects, as it is one critical factor causing the lack of the optimal "one microservice per team/developer." We also investigate the relation between the developers' technical roles and their tendency towards causing organizational coupling. The results, together with the approaches proposed, shall enable practitioners to be aware of such issues on the organizational level and to seek solutions with more insights.

The remainder of this paper is organized as follows. Section 2 introduces the related work on coupling in microservice and team collaboration. Section 3 presents the empirical study design for project selection and data analysis, as well as the approach for measuring organizational coupling. Section 4 presents the results that answer the research questions. Section 5 provides further discussion of the results. Section 6 discusses the threats to validity and potential ways to mitigate them. Section 7 concludes the article.

2 Related work

Developer interaction analysis has been approached from different perspectives in OSS communities. Given the large quantities of produced communication artifacts throughout the developer interaction in the development process, various automated approaches have been proposed. Common sources of input for such analysis include version control systems (performing mining source code repositories) [20], mailing lists, and issue trackers [21, 22], or developer online surveys [23]. Before the era of microservices, Bird et al. [22] considered social network communities and system modularity. They researched code artifact changes across modules and analyzed email archives to derive social networks and assess community alignment with modularity. The conclusions and research questions of Bird et al. [22] in the scope of microservices drive new perspectives. Microservices are self-contained, and according to Conway's Law, we can consider well-defined teams assigned to

particular microservice development. In addition, the remaining challenge related to crosscutting concerns cannot be simply negated in microservices.

With regards to microservices and well-defined separation boundaries by code repositories (or at least repository modules). It can thus be assumed that code artifacts modified by developers within the same community are placed in a related repository location.

Throughout OSS software development, it can be expected that developer assignments to subsystems remain stable (i.e., given expertise alignments, subsystem assignment, etc.). Ashraf and Panichella [24] analyzed a set of OSS projects to examine developer communities from the perspective of their subsystem assignment and interaction that emerging communities change considerably across a project's lifetime and do align with each other.

The microservices perspective, as suggested by Lenarduzzi et al. [25], enables teams to work independently, reducing cross-team communication. At the same time, upon microservice integration, issues are reported across teams, as suggested by Bogner et al. [26] who report on ripple effects. There are other underlying issues behind this relevant to system evolution, such as missing system-centered perspective and lack of tools to analyze coherence across microservices, perform modification trade-off analysis, or evaluate the conformance of the as-built and as-documented architectures.

Besides interaction analysis to understand communities, other interesting research directions took place. For instance, Marco et al. [27] analyzed GitHub commit comments regarding emotions and feelings expression showing that "one-commit" developers are more active and polite when posting comments as opposed to "multi-commit" developers, who are less active in posting comments, and when commenting, they are less polite.

In a timely thesis, Shi [28] looked into establishing contributor roles within software repositories by mining architectural information and proposed a metric to calculate it when detecting forward engineering, re-engineering, and management. In a case study on Apache Tomcat, they used the metric to deduce these roles and validate them with particular roles listed on the project website. Such a research direction aligns with the perspective of microservices with established separation of duty. Furthermore, the classification of experts responsible for re-engineering or management can lead to better insights into the applicability of Conway's Law across microservice developers.

It is also important to take into account that enterprise companies like Red Hat manage OSS projects [29] rather than projects based on volunteer contribution. This can influence role identification, contributor duty spread across modules, and also the community network. Spinellis et al. [29] considered the detection of OSS projects that are supported by enterprises. Such projects can serve as better benchmarks for practical case studies.

Concerning inter-project dependency identification, Blincoe et al. [30] considered reference coupling. The reference coupling method often identifies technical dependencies between projects that are untracked by developers. Understanding inter-project dependency is important for change impact analysis and coordination. In their study, they manually analyzed identified

dependencies and categorized and compared them to dependencies specified by the development team. They also assessed how the ecosystem structure compares with the social behavior of project contributors and owners. As a result, of socio-technical alignment analysis within the GitHub ecosystems, they found that the project owners' social behavior aligns well with the technical dependencies within the ecosystem. Still, the project contributors' social behavior does not align with these dependencies. In microservices, this could translate into system architects aware of consequences and microservice developers who operate in isolation as suggested by Lenarduzzi et al. [25] and unaware of such as inter-project dependency.

In a similar perspective, Scaliante Wiese et al. [31] researched co-change prediction. They use issues, developers' communication, and commit metadata to analyze change patterns for prediction models. They demonstrate that such models based on contextual information from software changes are accurate and can support software maintenance and evolution, warning developers when they miss relevant artifacts while performing a software change.

The organizational structure of software projects has long been a critical factor determining the projects' success [32]. Many studies have proposed approaches to analyze or improve software projects' organizational structure. Nagappan et al. propose a metric scheme to quantify organizational complexity regarding the product development process checking if the metrics impact failure-proneness where the level of organizational code ownership is a key metric [33]. Mockus studies the relationship between developer-centric measures of organizational change and the probability of customer-reported defects with the results showing organizational change is associated with lower software quality [34]. Isern et al. [35] investigate the popular agent-oriented methodologies in terms of their support and possibilities for modeling organizational structures with different levels of complexity.

Regarding the organizational structure of microservice projects, Li et al. propose an approach using social network analysis (SNA) to reconstruct the organizational structure of microservice-based software projects in terms of contributor collaboration [36]. Ashraf et al. conducted an empirical study and found that developer communities change considerably through projects' lifetime and that their alignment with the pre-defined microservice (or subsystem) teams is mostly low [37]. Comparatively, though many works have studied the approaches to extract the organizational structure and the changes therein through time, no studies have tried to investigate the potential optimization of such organizational structure. In the study we are extending [18], we validated that "one microservice per team/developer" optimization is not commonly respected. However, we did not further investigate possible implications such as its impact on coupling.

On the other hand, many studies have proposed methods to measure the coupling between software modules. Allen et al. propose related information theory-based measures of coupling and cohesion of a module based on the properties proposed by Briand et al. [38, 39]. Poshyvanyk and Marcus also propose a new set of coupling measures for object-oriented systems, named conceptual coupling, based on the semantic information shared between elements of the source code [40]. Other

methods are also proposed to measure the coupling between packages or classes [41, 42]. All these previous related work focus on the coupling metrics and measuring methods based on the dependency relations within the source code. None of them take into account the connections among developers or latent teams on the organizational level.

Regarding the coupling in microservice-based systems, Zhong et al. propose the Microservice Coupling Index (MCI) derived from the relative measurement theory, which measures how the coupled microservices are relative to the possible couplings between them [16]. Pedraza-Coello and Valdés-Souto propose a method to measure the coupling between microservices in early phases based on COSMIC method concepts regarding the data movements in functional processes [43]. d'Aragona et al. propose a metric to statically calculate logical coupling between microservices based on commits [17]. Though these previous works contribute to measuring the coupling issues in microservice-based projects, none of them investigate the coupling on the organizational level. Li et al. [19] propose an approach to measure the organizational coupling between microservices but does not investigate the connection of such phenomenon to other factors. In this paper, we will adopt the organizational coupling measuring method and investigate the collective relationship between developers' technical roles and their tendency to cause organizational coupling.

3 The empirical study

In this section, we describe the proposed research questions, the empirical study approach, the process of project selection, data collection and analysis, and the techniques utilized to facilitate the analysis. The conducted activities follow the guidelines defined by Wohlin et al. [44].

The goal of this paper is twofold. First, we evaluate to what extent the "one microservice per team/developer" strategy, recommended by practitioners [4–8] is respected in OSS projects. Second, we investigate how strong the organizational coupling between the selected projects can be by applying the organizational coupling assessment method proposed by Li et al. [19].

Toward such goals, we formulated three Research Questions (RQs).

RQ$_1$: *How well is the "one microservice per team/developer" strategy respected in OSS microservice-based projects?*

RQ$_2$: *What are the developer roles that (dis)respect the "one microservice per team/developer" strategy?*

RQ$_3$: *How strongly are the microservices coupled organizationally and who causes the organizational coupling?*

To answer **RQ$_1$**, we investigate the developers' committing behaviors within the selected projects, observing the number of microservices each individual developer covers using the statistics. To answer **RQ$_2$**, we analyze the roles of the developers based on the distribution of the programming languages they use by

adopting the technical role reference model proposed by Montandon et al. [45] and the Exploratory Factor Analysis (EFA) and the Kumar-Hassebrook (KH) similarity to map the developers with their roles. To answer $\mathbf{RQ}_3$, we use the organizational coupling analysis method proposed by Li et al. [19] and combine the results with the detected roles of developers.

3.1 Project selection and data collection

We considered the manually validated dataset including 145 microservice-based projects, proposed by Baresi et al. [4].

The selected dataset contains projects developed between 2018 and 2023 with at least 10 commits and 5 contributors. We would like to highlight that the contributors might have worked on any part of the project (e.g., documentation), and not only on the code thus, for instance, one project could have 5 contributors, but one developer. The aim of selecting the projects during this period is to filter out the projects with a history that is too short, which could result in a lack of sufficient commit data. Furthermore, due to the fact that by the time the data was first prepared, we can only have a five-full-year commit data from 2018 to 2023, in case the data is also needed for future works. The authors developed, validated, and released a tool to recognize the architecture (e.g., the microservices, the external services, and the databases used) in a given microservices-based project. In addition, the authors provided a list of 145 projects that have been manually validated as non-toy projects regularly using microservices, for which they also reported the list of built-in microservices in the form of relative paths and some other set-up information not used in our case. In particular, for our analysis, we leveraged the list of projects and the related list of microservices identified by a list of sub-project folders. The dataset consists of projects whose source code is accessible on GitHub and is complemented with further data, including the microservice list. Note that the projects analyzed are only projects that follow the mono-repo strategy, thus each microservice of each project is contained in a single git repository.

To select a relevant set of projects, we defined the following inclusion criteria:

1. **Projects with at least two microservices**: with this threshold, we aim to exclude the projects composed by single service, as no coupling between multiple services can be measured.
2. **Projects with at least two microservices committed in the last 12 months**: this rule is based on the assumption that — just as Conway describes for his law — it takes a certain amount of time for the communication structures to affect the software architecture. To enable us to observe the phenomenon we are investigating, we have defined a minimum period of 12 months, and, to increase relevance, we want to focus on projects modified in the *last* 12 months.

By requiring a minimum number of microservices and activities in the last year, non-representative outliers for our study can be excluded. It is important to note that we did not exclude projects based on their programming languages.

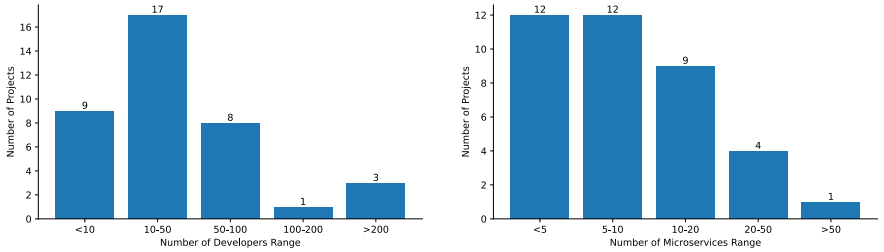

Fig. 1 Project distribution by #Developers and #Microservices

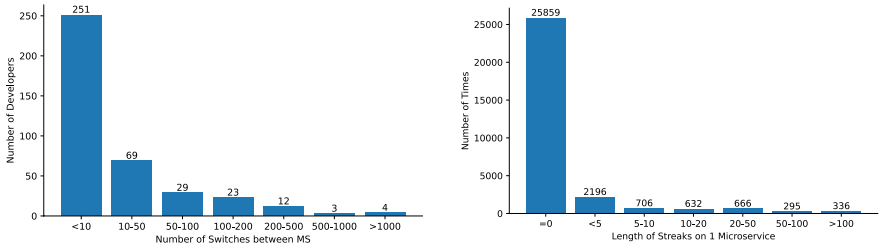

Fig. 2 Developer distribution by streak length and #Switches

As a result, we selected 38 microservice-based projects (listed in Appendix A) with a total of 379 microservices (10 microservices per project on average). We crawled each project's commit data up to 2023-12-31 using a customized Python crawler with GitHub APIs. The commit data features include each commit's author, the commit time, the change amount (lines of code), and the changed files. We specifically created a heuristic that matches if the paths of the modified files are contained in the project's list of microservices. In this way, we can identify the target microservice to which each commit was conducted. Especially, we identify all the files each commit touches. When the files touched by one commit are from different microservices, we count the corresponding developer commits on all the touched microservices, which creates logical couplings [17].

Among the selected projects, the distributions of the number of microservices and developers are shown in Fig. 1. We can observe that the majority of the projects have no more than 50 developers when only four projects have more than 100 developers. Meanwhile, there is only one project having more than 50 microservices when the majority of the projects have no more than 20 microservices. On the other hand, the large majority of the developers has no more than 10 times of commit switch between microservices (shown in Fig. 2). Only four developers achieved over 1,000 times of switches. Meanwhile, by checking the average time length of any developers continuously focusing on the same microservice (named streak), we find there are more than 25k times developers switching between microservice on the same day. There are also more

than 300 times, despite happening rarely, developers can concentrate on one microservice for more than 100 days. To be noted, there are less than 1% of the total developers who contributed to more than one project.

3.2 Data analysis

To answer $\mathbf{RQ}_1$, we investigated the relationship between the microservices structure and developers' commits by considering only the commits involving source code files. In this way, we excluded all the commits regarding documentation and setup files, which affect more than one microservice but without breaking the "one microservice per team/developer" strategy. We analyzed the distribution of commits over developers and microservices to understand how many microservices have developers in common, and thus how many developers work on more than one single microservice? We compared the number of microservices with no developers in common with the number of microservices that do, how often this happens, and the number of developers interested in this phenomenon.

However, the threat, in this case, is the situation where developers finish work on a microservice and get started to work on another microservice, or for some reason, they have just moved to another team of developers. From our point of view, this situation does not lead to a real violation of the "one microservice per team/developer" strategy. For this reason, we have defined a metric for counting how many times does a developer recommit to a microservice after starting work on another microservice; in other words, if a developer D_1 commits to microservice M_a, switches the team, and starts committing to a M_b microservice, then the result of our metric will be 0 because the developer never goes back to the previous microservice; otherwise, if after a while the developer commits back to M_a, our metric results will give 1, because the developer goes back to the previous microservice (M_b).

To better understand the allocation of developers in different microservices, we intuitively categorize the projects based on the number of developers and their coverage of microservices as follows.

- *One-Dev-ALL-MS projects*: Projects where only one individual contributor covers all microservices while all the others cover part of them;
- *Multi-Dev-ALL-MS projects*: Projects with multiple contributors covering all microservices;
- *Multi-dev-MULTI-MS projects*: Any projects with no contributors covering all microservices; nor do they adopt "One microservice per team/developer" strategy;
- *One-MS-per-developer projects*: Any projects with each contributor/team working only on one microservice;

To answer $\mathbf{RQ}_2$, we first needed to understand how to identify the role of each contributor in OSS microservice projects. Unlike industrial projects, within OSS projects on GitHub, the contributors are neither assigned roles by "project managers" or "product owners" nor obliged to focus on the tasks assigned to them

in the corresponding areas. Therefore, we shall only be able to understand the roles based on the domains each contributor has been contributing to.

Montandon et al. [45] proposed a machine-learning-based approach to automatically identify the technical roles of OSS project contributors. They used more than 100k developers' data from GitHub together with Stack Overflow data and studied five critical roles: Backend, Frontend, Data scientist, DevOps, and Mobile. Herein, we initially adopt the same settings. They concluded that the relevant features for each technical role and the adopted programming languages are closely relevant. For example, Data scientists are likely to contribute more using Jupyter Notebook when Backend contributors use PHP. However, in many situations, it is difficult to determine contributors' roles due to the fact they tend to contribute using multiple languages. For example, when a developer contributed 4000 LOC in Python and 5000 in HTML and CSS, we cannot arbitrarily determine whether they are Data scientists or Frontend contributors. To do some, we need to summarize the different language contributions of each contributor. From all the commit data collected in any given project, we can identify the commits conducted by any particular developer. For each commit therein, we can also get information on the detailed changes by LOC in each file that this commit touches while we identify the programming language for each file based on the extension; for example, *.py* is for Python, *.cpp* is for C++, etc. Then, we can summarize for the developers by LOC how much they contribute in all the languages they used to commit. By doing so, we can have a list of values representing each developer's contribution in terms of the pre-defined list of all languages. Then by comparing it to the reference model adapted from [45], we can determine the tech-role tendency for each developer.

To answer $\mathbf{RQ}_3$, by adopting the approach of assessing the organizational coupling proposed by Li et al. [19], we can calculate and observe the organizational coupling situation of the selected projects. Moreover, we can also observe the changes in the organizational coupling situations for each project through its lifecycle by calculating the coupling within each given consecutive time period.

Then we can also identify the developers that cause the organizational coupling between any microservices by conducting cross-service contributions. While answering the previous RQs, we identified each developer's technical roles from the selected projects. Therefore, we can summarize and compare the common technical roles of the ones who cause and do not cause the organizational coupling in the projects.

3.2.1 Exploratory factor analysis (EFA)

In general, factor analysis is adopted to detect the latent variables sharing a common variance and being unable to be observed [46]. These variables are obtained from a larger set of measurable and observable ones. These detected factors are not directly measured but are essentially hypothetical constructs that are used to represent variables [47]. Especially, the Exploratory Factor Analysis (EFA) [48] aims to discover not only the number of factors but also what measurable variables together

influence which individual factors [49]. With EFA, we can reduce the complexity of the data and also explain the observations with a smaller set of latent factors. Importantly, by doing so we can also discover the relations among the variables.

Herein, we follow these steps to conduct EFA on our commit dataset and determine the profile of each developer:

Step 1: Preprocessing Firstly, based on the obtained dataset, we shall group the data based on individual developers with the features (i.e., variables) being their different behaviors. Furthermore, the dataset is normalized by each column.

Step 2: Data Verification To investigate whether the obtained data is suitable for factor analysis, we need to verify its sampling adequacy and statistical significance. For example, we can use the commonly used Bartlett's Test of Sphericity [50] and Kaiser-Meyer-Olkin (KMO) Test [51] for such a purpose.

Step 3: Determining Factor Number After verifying the dataset, to facilitate EFA in terms of finding the number of factors, parallel analysis (PA) [52] is commonly applied. Herein, we employ the Monte Carlo simulation technique to simulate random samples consisting of uncorrelated variables that parallel the number of samples and variables in the observed data. We extract the eigenvalues of the correlation matrix of the simulated data within each simulation and calculate the mean across several simulations [52]. We then compare the extracted eigenvalues that are ordered by magnitude to the average simulated eigenvalues that are also ordered by magnitude. The decision criteria are that the factors with observed eigenvalues higher than the corresponding simulated eigenvalues are considered significant.

Step 4: Factor Extraction and Interpretation With the number of factors determined, we conduct the EFA on the dataset. To simplify the interpretation of the factor analysis result, we employ the varimax rotation technique [53] to maximize the variance of each factor loading. Thereafter, a reasonable threshold for the determination of variable correlation shall be set, with which the factor-variable reference table will be determined.

Step 5: Determining Individual Developer Role Allocation To apply the factor-variable relation to individual contributors, we shall calculate the similarity between the developers' contributions in terms of the languages and each detected factor. By comparing the contributor's similarity to each role-factor, we shall understand more intuitively which role(s) they lean to. Such results can be visualized in a radar chart.

For this study, as a result of the EFA, we shall have a set of factors, each of which is closely related to a set of latent variables, i.e., programming languages. To be noted, since the original data are collected from projects of different programming languages, contributors working on different languages likely lean toward similar roles. For example, contributors working on CSS and VUE can both be *Frontend* contributors. Therefore, we shall observe the loadings of the EFA and manually merge factors related to only closely connected languages into the unified roles.

3.2.2 Developer technical role identification

Here, we identify the technical role of each developer by calculating the similarity of his/her technical workload (in terms of programming language) and the distribution of the languages in the factors. Such similarity can be calculated as follows.

One way of calculating the similarity between the technical role factors and the developer's language workload is to use The Jaccard Similarity Coefficient. However, Jaccard similarity does not consider the different workloads for each programming language; therefore, we herein adopt the Kumar-Hassebrook (KH) similarity [54], which also incorporates the inner product of the assigned values of the variables. Moreover, using the KH similarity, we can evaluate each contributor's effort level in each pre-detected role-factor, respectively. The KH similarity is calculated as follows.

$$KH(L_C, L_F) = \frac{\sum_{x=1}^{k} c_x \cdot f_x}{\sum_{x=1}^{k} c_x^2 + \sum_{x=1}^{k} f_x^2 - \sum_{x=1}^{k} c_x \cdot f_x} \tag{1}$$

where:

- k is the total number of elements considered in each set.
- c_x and f_x represent the values or weights assigned to the x-th element in the sets L_C and F, respectively. These could represent the presence, frequency, or importance of elements in each set.
- $\sum_{x=1}^{k} c_x \cdot f_x$ computes the sum of the product of corresponding values or weights from each set, analogous to a dot product, reflecting the degree of match between the elements across the sets.
- $\sum_{x=1}^{k} c_x^2$ and $\sum_{x=1}^{k} f_x^2$ calculate the sum of the squares of the values or weights for elements within each set, which can be seen as a measure of the set's magnitude or diversity.
- The denominator acts to normalize the measure, taking into account the diversity within and across the sets, ensuring the similarity score is scaled and interpretable.
- $KH(L_C, L_F)$ yields a similarity score that quantifies how similar the sets L_C and L_F are, based on the overlap and the distribution of values or weights assigned to the elements in each set.

For example, we define the reference list of programming language including *Python*, *C*, *Html* and *Kotlin* and the reference vector of "Frontend developer" is (0, 0, 1, 0). A developer's contribution to each language after normalization is (0.2, 0.5, 0.8, 0.7). Then, the developer's similarity to the role of "Frontend developer" can be calculated as follows.

$$KH = \frac{(0 \times 0.2) + (0 \times 0.5) + (1 \times 0.8) + (0 \times 0.7)}{(0^2 + 0^2 + 1^2 + 0^2) + (0.2^2 + 0.5^2 + 0.8^2 + 0.7^2) - (0 \times 0.2) + (0 \times 0.5) + (1 \times 0.8) + (0 \times 0.7)} \approx 0.494 \tag{2}$$

Using the KH similarity, we shall be able to evaluate each contributor's effort level in each pre-detected role-factor respectively.

3.2.3 Identify microservice teams

To assess the organizational coupling between microservices, we first need to identify the teams for each microservice, i.e., which developers are working mainly on which microservices and what the proportion of their contribution is to that of the whole service. For such a purpose, we adapt the method proposed by Bird et al. [55]. This method of microservice team identification is also adopted by Li et al. [19].

For each microservice in a project, we can identify the list of files in the corresponding folder or repository. Therefore, by examining all the commits of the project history, we shall know which developers have contributed to which microservices as well as how much the contribution is via the volume of changes in those commits. By doing so, we can identify the team of each microservice to which all developers therein have contributed. Herein, we use lines of code (LOC) as a developer contribution metric [56]. Despite the fact that LOC is limited in reflecting code productivity or quality, it is still a widely commonly adopted method and suits this scenario well.

Furthermore, by comparing the proportion of the developers' contributions to the whole service, we can identify the leader, major contributors, and minor contributors of the microservice. Such a proportion is denoted as the ownership or ownership proportion of the microservice, which is the concept used in Bird et al.'s study [55]. Herein, adopted the criteria proposed by [55], we see the developer(s) who has the highest proportion of ownership for the microservice as its *Teamleader(s)*. We also define *Major* contributors as the developers whose contribution reaches at least 5% proportion level, and *Minor* contributors as the developers whose contribution does not reach 5% proportion level. To be noted, herein, we use the number of commit changes instead of the number of commits due to the consideration that commits vary largely between one and another in terms of the exerted effort from the developers.

3.2.4 Contribution switch as weight

Herein, by adopting the method proposed by Li et al. [19], we also consider the developer's contribution switch as a weight influencing the organizational coupling measure. Here we consider a "contribution switch" occurs when a particular developer commits to two different services sequentially. Provided each commit carries the same amount of contribution when particular developers switch between two microservices more frequently, it is more likely they create functional or logical coupling between them. We understand the concept of "gradual transition" as situations where particular developers switch less frequently between two microservices when, at a certain point, they will focus on the second microservice and do not commit back to the first one. This also means his/her influence between these two target

microservices on the organizational level is decreasing gradually. Certainly, the more contributions the developers make with these commits, the more likely they will create coupling. Admittedly, the observation time period is also important when determining the influence of the "switch", which is partially why we choose to use the annual average organizational coupling value to demonstrate the results.

We define the *contribution switch weight* between these two microservices M_a and M_b as $S_D(M_a, M_b)$ when we assume a developer D from the team of M_a (T_{M_a}) or the team of M_b (T_{M_b}). Figure 3 shows an example of the contribution switch of one developer between microservice M_a and M_b. Whenever D commits to M_a and then commits to M_b afterward (or from M_b to M_a), we consider a "switch" is performed. To be noted, we do not consider the time intervals between two consecutive commits but only consider their sequential relation. Precisely, in Fig. 3 developer 1 mainly works on microservice M_a but switches to M_b once; developer 2 switches back and forth between both microservices M_a and M_b.

Therefore, given the sequence of commits of D in terms of M_a and M_b, the contribution switch value is the number of switches performed by D. Especially, regarding the situation of logically coupled commits [17] where both microservices are changed in a single commit, two contribution switches are counted for this situation.

To generalize, given the previously described situation where k contribution switches are performed by D between M_a and M_b while D has in total n commits for both microservices, the contribution switch weight can be calculated as follows.

$$S_D(M_a, M_b) = \frac{k}{2 \times (n-1)} \tag{3}$$

where:

- $S_D(M_a, M_b)$ represents the contribution switch weight from microservice M_a to microservice M_b by developer D.
- k denotes the total number of contribution switches performed by D between M_a and M_b. A "contribution switch" occurs each time the developer shifts their focus from working on one microservice to the other.
- n signifies the total number of commits made by D to both microservices, M_a and M_b, capturing the overall engagement of the developer with these projects.
- The denominator, $2 \times (n-1)$, normalizes the switch count by considering the maximum possible number of switches in n commits, which is $n-1$, and then adjusting for the "bidirectional nature" of switches between M_a and M_b, The reason to double $n-1$ value here is that any switch must happen between two microservices with the influence to both when each additional commit should be counted twice.
- This formula yields a value between 0 and 1, where 0 indicates no switching between microservices (i.e., all contributions are to a single microservice) and 1 represents maximum switching behavior (i.e., the developer alternates between microservices with every commit, which is theoretically possible if $k = n-1$).

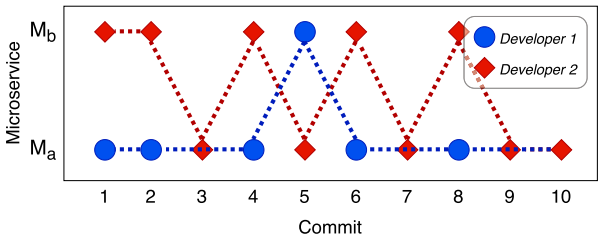

Fig. 3 Contribution switch between microservices

3.2.5 Measure organizational coupling

For the measurement of microservice organizational coupling, we also adopt the method proposed by Li et al. [19]. Firstly, we identify the teams of the two microservices, M_a and M_b, using the previously proposed method. The teams are denoted as T_{M_a} and T_{M_b}. Considering there are p developers who contributed to both M_a and M_b, denoted as $T_{(M_a \cap M_b)} = \{D_1, D_2, ...D_p\}$. For any particular developer, $D_i \in T_{(M_a \cap M_b)}$, we denote the temporal sequence of his/her commits as C_{D_i}. For each commit, $c \in C_{D_i}$, by checking the file(s) changed by this commit, we can identify which microservice(s) this commit is for. Therefore, by finding the commits that are conducted to M_a or M_b or both, we obtain a sub-sequence, denoted as $C_{D_i}(M_a, M_b)$. As the commit sequence is identified, we can also identify the contribution switches therein and calculate the switch weight, $S_{D_i}(M_a, M_b)$, based on the method described above.

For the coupled contribution of D_i on M_a and M_b, we adopt the harmonic mean of D_i's contribution, because the more equally any developer commits to multiple microservices, the more organizationally coupled the two microservices are, regarding this developer's contribution. Let $\{ca_1, ca_2, ...ca_m\}$ be the corresponding contribution value sequence for the m commits in $C_{D_i}(M_a)$ while $\{cb_1, cb_2, ...cb_n\}$ be that for the n commits in $C_{D_i}(M_b)$. Herein, the contribution value of each commit is calculated by the sum of all the number of changes to each file in the target microservices. Let $OC(D_i, M_a, M_b)$ be the organizational coupling (OC) caused by developer D_i's cross-service contribution on microservices M_a and M_b, we can calculate $OC(D_i, M_a, M_b)$ as follows.

$$OC(D_i, M_a, M_b) = \left(\frac{2 \sum_{j=1}^{m} ca_j \sum_{k=1}^{n} cb_k}{\sum_{j=1}^{m} ca_j + \sum_{k=1}^{n} cb_k} \right) \times S_{D_i}(M_a, M_b) \tag{4}$$

where:

- $\{ca_1, ca_2, ...ca_m\}$ represents the sequence of contribution values for m commits made by D_i to microservice M_a. Each contribution value ca_j is the sum of changes across all files in a single commit to M_a.

- $\{cb_1, cb_2, ...cb_n\}$ denotes the sequence of contribution values for n commits made by D_i to microservice M_b, with each cb_k calculated similarly as a sum of changes in a single commit to M_b.
- $OC(D_i, M_a, M_b)$ defines the organizational coupling caused by D_i's contributions across the two microservices, factoring in both the volume and significance of changes to M_a and M_b.
- The numerator, $2\sum_{j=1}^{m} ca_j \sum_{k=1}^{n} cb_k$, calculates a weighted interaction term between the contributions to M_a and M_b, emphasizing the combined impact of work across both services.
- The denominator, $\sum_{j=1}^{m} ca_j + \sum_{k=1}^{n} cb_k$, aggregates the total contribution value to both microservices, ensuring the OC value is normalized by the overall contribution effort.
- $S_{D_i}(M_a, M_b)$ is the contribution switch weight for D_i between M_a and M_b, reflecting the frequency of shifts in focus between these microservices.

Thus, the overall organizational coupling between M_a and M_b, denoted as $OC(M_a, M_b)$, can be calculated as follows.

$$OC(M_a, M_b) = \sum_{i=1}^{p} OC(D_i, M_a, M_b) = \sum_{i=1}^{p} \left(\frac{2\sum_{j=1}^{m} ca_j \sum_{k=1}^{n} cb_k}{\sum_{j=1}^{m} ca_j + \sum_{k=1}^{n} cb_k} \right) \times S_{D_i}(M_a, M_b) \tag{5}$$

where:

- p denotes the total number of developers contributing to either microservice M_a or M_b. It is the cardinality of the union of the set of developers to M_a and the set of developers to M_b.
- For each developer D_i, m_i and n_i represent the number of commits they have made to M_a and M_b, respectively.
- ca_{ij} and cb_{ik} are the contribution values for the j-th commit to M_a and the k-th commit to M_b by developer D_i, respectively. These values quantify the magnitude of changes (e.g., number of lines added, modified, or deleted) in each commit.
- The term $OC(D_i, M_a, M_b)$ calculates the individual organizational coupling caused by D_i's cross-service contributions, taking into account both the volume of contributions and the frequency of switching between services.
- The formula aggregates these individual OC values to determine the overall organizational coupling, $OC(M_a, M_b)$, which quantifies the combined effect of all developers' contributions on the relationship between the two microservices. Simply put, according to the formula, the total organizational coupling between M_a and M_b is the sum of the organizational coupling caused by all the developers who "switch" between M_a and M_b.
- The value of $OC(M_a, M_b)$ can range from 0 to infinity, as there is no limit on either the volume of contribution between services or the number of developers therein.

3.3 Verifiability and replicability

To allow verifiability and replicability, we published the raw data in the replication package.[1]

4 Results

In this section, we report the results, organized to address the proposed research questions (RQs).

4.1 One microservice per team/developer

We investigated the single developer, assuming that a single developer does not belong to more than one team at the same time. Figure 4 compares the number of microservices with shared developers (*MSs with Shared Dev*) with the number of microservices where all developers committed only to the same microservice (*MSs without Shared Dev*).

Unexpectedly, only two projects always respected the "one microservice per team/developer" strategy, while the remaining projects shared among services.

Since in the vast majority of the projects (Fig. 6) a developer works on more than one microservice, we continue our analysis to understand if developers are simply switching teams, or are working on more microservices at the same time.

Figure 5 shows the result of the number of times developers commit back on a microservice after moving to another one among the projects. In two projects out of 38, developers never commit back to the previous microservice. In most of the projects (**53%**), the median is 0, in **34%** of the projects the median is between 1 and 10, and finally in **13%** of the projects is more than 10. However, analyzing the figure, we can see that the boxplots are very stretched, thus in the same project some developers do not return after changing microservice (or never changing microservice), and some developers instead, commit to the previous microservice.

As a result, we conclude that in an OSS context, the "one microservice per team/developer" strategy is not respected as in most cases, developers work on more than one microservice in parallel.

4.2 Technical roles across multi-microservices

We first investigated the strategies of different contributor-microservice effort allocations. Figure 6 shows the distribution of the "microservices per developer" of each selected project. To be noted, the *light-example-4j*, which contains 154 different microservices, is not shown in Fig. 6. Because showing this project in the chart will make the details of all other projects invisible, nonetheless, this project was certainly included in the analysis process.

[1] https://dx.doi.org/10.6084/m9.figshare.27153711.

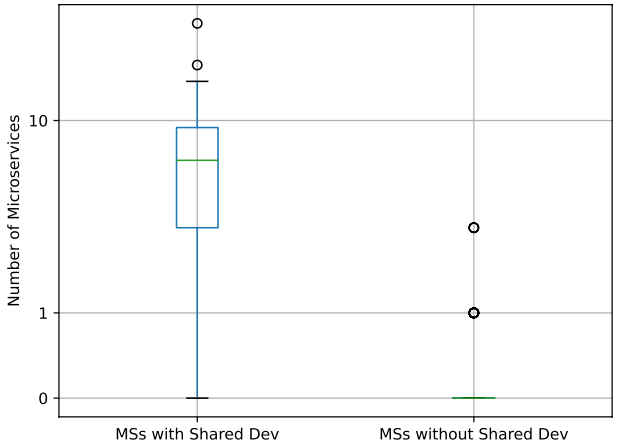

Fig. 4 Number of microservices with shared and not shared developers (RQ_1)

From Fig. 6, we can easily find that for all projects, except for two, the "one microservice per team/developer" strategy has not been respected. For the selected projects, the majority of medians range from one to seven. For most projects, developers are always committed to multiple microservices. Furthermore, Fig. 8 shows the difference in the number of developers committing to microservices and the number of microservices committed. Meanwhile, Fig. 7 shows the comparison between the projects in terms of 1) the number of developers and the number of developers committing to any microservices, and 2) the number of microservices and the number of microservices being committed (Fig. 8).

On the contrary, many projects that contain various numbers of microservices have one individual contributor who contributes to all the microservices. We name such a strategy *One-Dev-ALL-MS*. For example, in project *geoserver-cloud*, contributor `gabriel.roldan` committed in all the 11 microservices, and in project *eShopOnContainers*, contributor `mvelosop` covers all the 17 microservices. Furthermore, many projects even have multiple contributors that cover all the microservices. We name such a strategy *Multi-Dev-All-MS*. For example, in project *loopback4-microservice-catalog*, there are eight contributors covering all 18 microservices; and in project *DeathStarBench* six contributors cover all three microservices. Such a phenomenon is likely irrelevant to either the microservice number or several contributors.

We find that 16 out of the 38 projects have only one individual contributor covering all microservices while all the others cover part of them (i.e., One-Dev-ALL-MS). Meanwhile, another 10 projects have multiple contributors covering all microservices (i.e., Multi-Dev-ALL-MS). There are also 12 projects having no developers covering all microservices but they have multiple developers working on more than one microservices (Multi-Dev-MULTI-MS).

To investigate the potential roles of the contributors that cover all microservices and the other common contributors further, we used EFA to detect the latent factors.

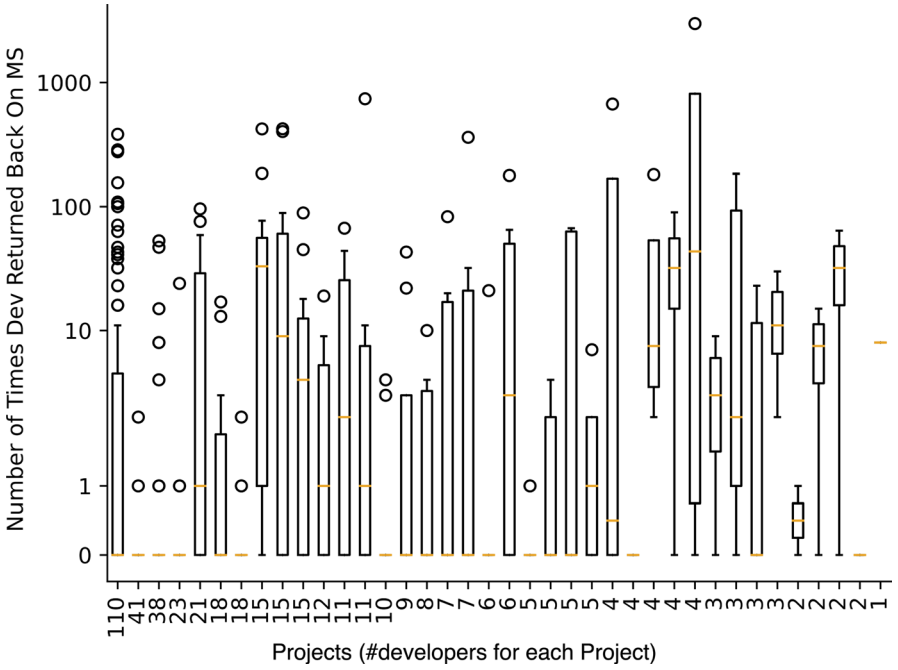

Fig. 5 Frequency that developers have committed back (RQ_1)

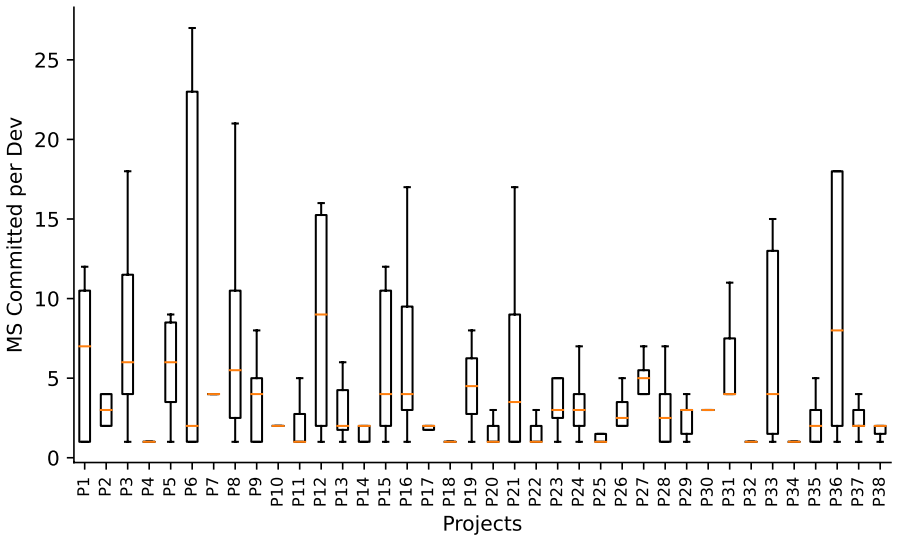

Fig. 6 MS Per developer sorted by #Microservice (with #Developers)

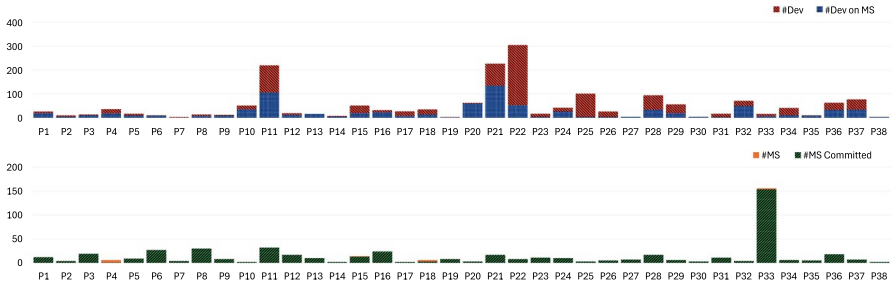

Fig. 7 MS Per developer sorted by #Microservice (with #Developers)

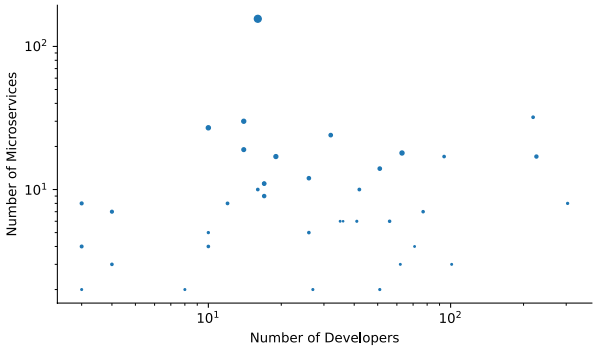

Fig. 8 Scatter Plot for Number of Microservices and Number of Developers

1. Preprocessing. Firstly, we grouped the original dataset by the contributors for the preprocessing. For each contributor, we synthesized their contribution in every language by checking the extensions of the committed files. We crawl each project's languages using GitHub API. By grouping the data, we obtained the 1 536 contributors' dataset with their contribution to the 33 languages. And we further normalized the dataset into values between zero and one.

2. Data Verification. Herein, the KMO score for this dataset is 0.585. It shows that the sampling is adequate and applying factor analysis is useful for this dataset. When applying PA to the dataset, we detected 13 factors as there are 13 out of 33 observed eigenvalues greater than 1.0. The corresponding factor loadings are shown in the replication package.

3. Determining Factor Number. Based on the result of the PA, the turning point can be found easily by examining the differences between observed eigenvalues and simulated eigenvalues. Since the simulated eigenvalue becomes greater than the observed eigenvalue in the 14th factor (1.00049 and 0.90517, respectively), the first 13 factors are retained. The number of factors is therefore 13. According to Guadagnoli and Velicer [57], scores greater than 0.4 are considered stable, especially when all variables are not cross-loaded strongly.

4. Factor Extraction and Interpretation. The initially detected factors and the correlated variables are reported in the replication package. Herein, we adapted Montandon et al.'s role-language relevance results [45] as the reference to analyze the interpretation of each factor. To be noted, we added several languages that are not listed in Montandon et al's study based on common knowledge and experts' opinions. The list of the languages and their reference to all the pre-defined roles can be found in Table 1.

For example, we adopted Miranda et al's PHP TechSpace reasoning on *PLSQL*, which is connected to *PHP* as an extension [58]. Meanwhile, *Vue* and *TypeScript* belong to frontend when *Go* belongs to backend. Because we took into account only microservice-based applications, we see *C#* and *Java* as backend. On the other hand, for the same reason, we took into account *YAML* and *Batchfile* as DevOps. *FreeMarker* is a Java library to generate text output for HTML web pages, which is also connected to frontend. In addition, we considered *Markdown* related to the role of *Documentation*, as it is often used for "ReadMe" in Github. To be consistent with the reference scoring, we assigned 0.6 to the clear relations which is the highest in [45]. The relevance values are directly adopted from Montandon et al.'s model.

Meanwhile, we also considered the other contributors that are not related to any specific roles above as *Others*. By calculating the KH similarity between the role factors in the factor table obtained previously and the reference table [45]. Here we assigned the role with the highest similarity score to each factor.

Furthermore, we combined the factors with the same roles and obtained the final role-factor reference model.

5. Determining Individual Developer Role Allocation. By using this role-factor relevance model, we simply calculated the factor similarities of any contributor, given his/her contribution allocation in terms of the 33 languages. Furthermore, we investigated the difference in terms of the contributor roles of the project strategies mentioned above.

Figure 9 shows the average behavior patterns of the different types of contributors in terms of the technical roles. From Fig. 9, we can easily observe that the individual contributors who cover all microservices (i.e., One-Dev-ALL-MS) of the projects contribute largely as Documentation+. They are also strongly involved in Frontend,

Table 1 Updated role-language relevance results (RQ_2)

Role	Related languages
Backend	'Go': 0.06, 'C#': 0.06, 'java': 0.06, 'PHP': 0.011, 'HTML': 0.01, 'JavaScript': 0.01, 'Java': 0.01, 'Ruby': 0.01, 'Python': 0.008, 'PLSQL': 0.008,
Frontend	'Vue': 0.06, 'TypeScript': 0.06, 'JavaScript':0.036, 'Python': 0.018, 'HTML':0.017, 'CSS': 0.016, 'SCSS': 0.015, 'Java': 0.01, 'Jupyter Notebook': 0.01, 'FreeMarker': 0.001, 'Smarty': 0.06
Data scientist	'Jupyter Notebook': 0.1, 'Python': 0.03, 'JavaScript': 0.02
DevOps	'Shell': 0.06, 'JavaScript': 0.01, 'Python': 0.005, 'Batchfile': 0.06, 'YAML': 0.06, 'Makefile':0.06
Documentation	'Markdown': 0.06
Others	*if related to NONE above*

when slightly less in Backend and DevOps roles. To be noted, they also contribute as Fullstack but are nearly non-existent in the other aspects. In addition, the One-Dev-ALL-MS also contributes to the Data Scientist role more than the others. On the other hand, for the multiple contributors that cover all the microservices (i.e., Multi-Dev-ALL-Ms), these contributors, on average, contribute less than the One-Dev-ALL-MS mentioned above. However, they contribute slightly more as Frontend than the other roles. They cover the Fullstack role a little less but surprisingly at a similar level compared to One-Dev-ALL-MS. Furthermore, they contribute more in other languages that are not role-related than that from the One-Dev-ALL-MS. The Multi-Dev-ALL-MS also contributes to Backend, Documentation, and Data Scientist, but much less than the other aspects. Regarding all the Multi-Dev-MULTI-MS contributors, they contribute much less in terms of all working roles than the One-Dev-ALL-Ms and Multi-Dev-ALL-Ms.

The majority of the microservice projects have one or multiple contributors who commit to all microservices. The single contributor who covers all microservices (One-Dev-ALL-MS) contributes much more than the multiple contributors covering all microservices (Multi-Dev-ALL-MS) in all roles, except that Multi-Dev-ALL-MS contributes more in non-role-related languages. Multi-Dev-MULTI-MS contributes much less in all roles.

4.3 Organizational coupling

By adopting the organizational coupling measurement, we first assessed the overall organizational coupling of the selected 38 projects. Because the projects were created in different years while having various longevity, we calculated and compared the annual average organizational coupling. The aim of adopting an annual average of organizational coupling is to moderate the difference in the growth rate of the selected projects. Several projects were initiated years before the others when their organizational structure might have been optimized. On the other hand, the monthly activities of any projects can also be unstable; e.g., the average

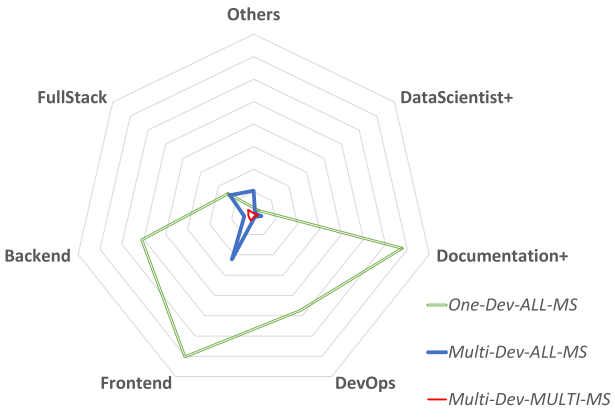

Fig. 9 Average Role-Factor Distribution of Each Strategy (RQ_2)

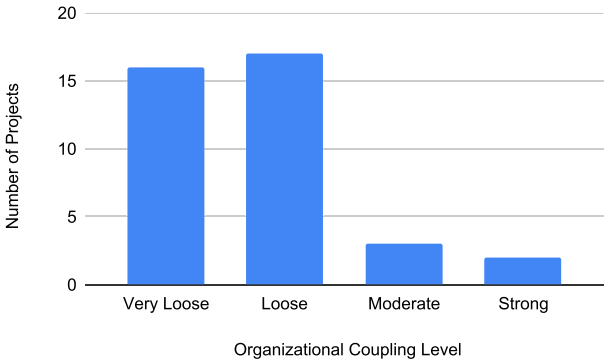

Fig. 10 Project Organizational Coupling Levels

activities in holiday months (e.g., December) are likely to be lower than those of other months. Therefore, it is hard to show any meaningful visualization of the evolution in the metric.

By adopting the evaluation criteria from Li et al. [19], we classify the projects based on the organizational coupling level as follows: 1) strongly Coupled: $OC \geq 10,000$; 2) Moderately Coupled: $1,000 \leq OC < 10,000$; 3) Loosely Coupled: $100 \leq OC < 1,000$; 4) Very Loosely Coupled: $OC < 100$. Fig. 10 shows the distribution of projects in terms of different levels of organizational coupling.

Figures 11, 12, 13, and 14 show the evolution of organizational coupling for each selected project from 2018 to 2023. Herein, for each project, we calculate its average organizational coupling of each year by selecting the corresponding commit data conducted within that year. We group the projects based on the different coupling levels to observe potential trends. In the figure, each line represents one individual project. The figures exemplify the trend of the various projects. Herein, we show the diagrams to illustrate the organizational coupling magnitude in each group through the years. We can observe that there are no clear patterns in any project group, but apparently, several projects have an increase in such coupling followed by a decrease, regardless of the sharpness of such changes. For the majority of these projects, the coupling value in the latest year is lower than the average value, implying that, intentionally or unintentionally, the organizational coupling is being mitigated.

Amongst the selected 38 projects, two projects, *delimitrou/DeathStarBench* and *sourcefuse/loopback4-microservice-catalog* have strong organizational coupling. The two projects have been active for respectively four and five years. Project *delimitrou/DeathStarBench* has the highest organizational coupling of 13430.87 with *sourcefuse/loopback4-microservice-catalog* running second with 10926.64. The three projects with moderate organizational coupling are *TaitoUnited/full-stack-template*, *1-Platform/one-platform* and *dotnet-architecture/eShopOnContainers* with respectively 3192.81, 7782.27 and 1044.48. On the other hand, 17 projects have loose organizational coupling while the other 16 have very loose coupling.

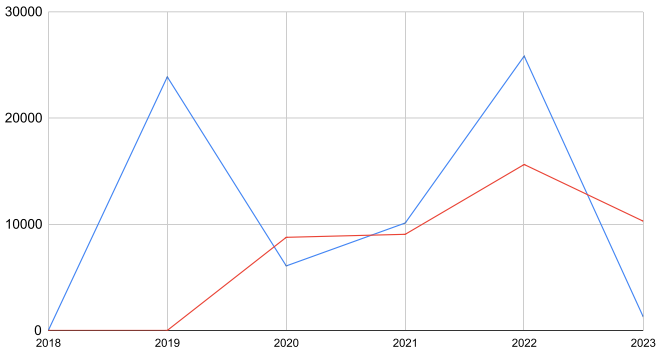

Fig. 11 Organizational coupling evolution of strongly coupled projects

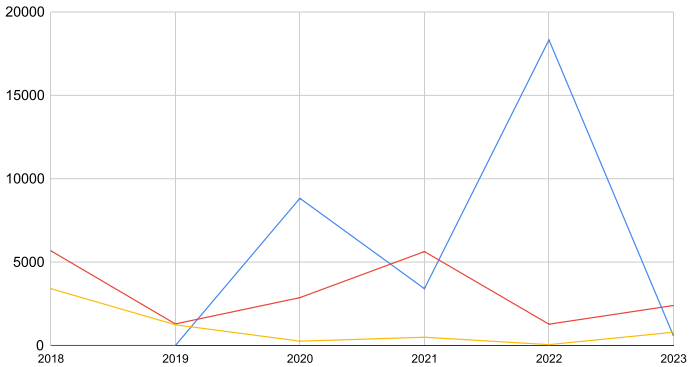

Fig. 12 Organizational coupling evolution of moderately coupled projects

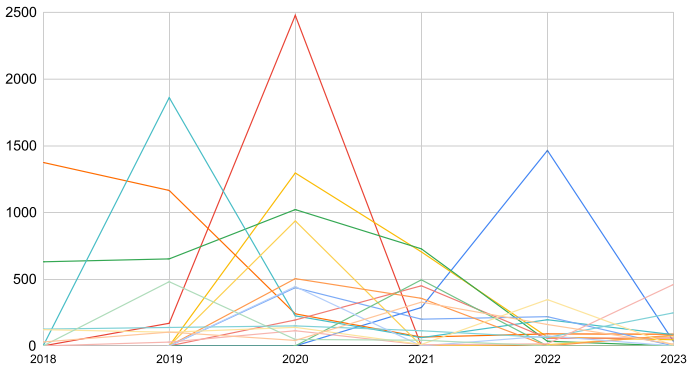

Fig. 13 Organizational coupling evolution of loosely coupled projects

Amongst these projects, we can observe that their active duration is irrelevant to the coupling levels. Some projects, e.g., *kanisterio/kanister* and *MozillaSecurity/orion*, as projects that have been active for over six years, have very low organizational coupling (1.28 and 13.89). On the contrary, there are also projects over six years with moderate or strong coupling, such as the ones mentioned previously. Furthermore, we also investigated the potential correlation between the organizational coupling and the other factors, e.g., the number of microservices in the projects, the number of different contributors, the number of commits, and the size of the project. Surprisingly, we found that the organizational coupling only has a very low correlation with the number of different contributors in the project (0.236), but no correlation with the number of some microservices (−0.052), the number of commits (0.132), or project size (0.075). Furthermore, the correlation situations remain when considering the accumulated organizational coupling instead of the average.

Moreover, we also investigate the changes in the organizational coupling of these projects through the years. We illustrate the changes in organizational coupling from 2018 to 2023 grouped by previously defined level criteria (Figs. 11, 12, 13, 14). It is noticeable that for strongly coupled projects, the organizational coupling is decreasing in the latest year, despite the increase in previous years, as well as at the beginning of the project. A similar trend can also be found in the moderately coupled *1-Platform/one-platform* project and many in the other groups, e.g., loosely coupled *FraunhoferISST/diva* project and the very loosely coupled *MozillaSecurity/orion* project.

It is noticeable that such fluctuation in organizational coupling is common amongst the projects through their life cycles regardless of the coupling levels. Therein, the difference can be enormous. For example, the organizational coupling of the project *1-Platform/one-platform* increased from 3406.872 to 18320.587 (by 437.8%) from 2021 to 2022 and then decreased to 577.286 (by 96.8%) in 2023. Meanwhile, several projects have an overall trend of decreasing the organizational coupling through these years, e.g., project *OpenLiberty/guide-getting-started*, though minor setbacks, such as a small increase from 2021 to 2022 (66.187 to 90.051), is, to a large extent, reasonable.

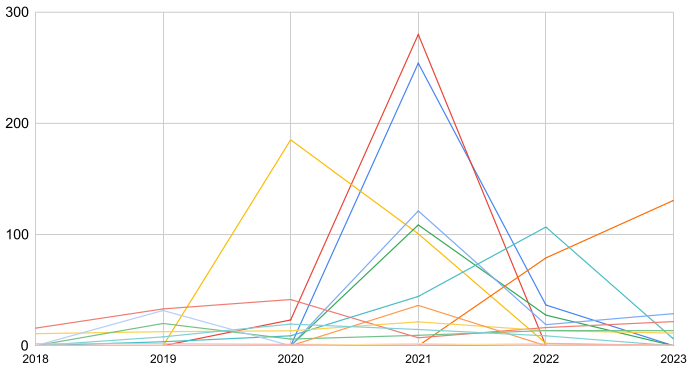

Fig. 14 Organizational coupling evolution of very loosely coupled projects

Regarding the different roles, we first identify the ownership roles, i.e., leader, major contributor, or minor contributor, of each microservice in each selected project. Together with the previous results of EFA, we can compare the different technical roles of the contributors of different service ownership levels. As shown in Fig. 15, despite the different ownership levels, the overall technical roles are, to some extent, similar. It is noticeable that contributors of all ownership levels are more leaning toward the "FullStack" role, "Frontend", "Backend", "DevOps" and "Documentation+" roles on the second tier, with "DataScientist+" and "Others" as the last. Proportionally, the service leaders focus more on "FullStack" roles than the others, while the major contributors focus less on "DevOps" than the service leaders. As shown in the figure, we can observe such a difference is minor. In addition, it is also obvious that the level of contribution decreases accompanied by the degrades in ownership levels.

Furthermore, we assess the technical roles of the developers that contributed to the organizational coupling. As shown in Fig. 16, we can simply observe that the developers who create organizational coupling contribute much more than the ones who do not. Proportionally, the developers that create organizational coupling cover more in the "Others" dimension when the other party does not. Meanwhile, the developers who do not create organizational coupling focus more on the "Backend" role compared to the ones who do. We also group the developers creating organizational coupling based on their service ownership levels. Therein, the service leaders focus more on the roles of "Documentation", "Frontend" and "Fullstack". The minor contributors, on the other hand, focus more on "DevOps" and "Fullstack" roles compared to their contribution in the rest. Similarly, the level of contribution also decreases with the degrades in ownership levels for the developers creating organizational coupling.

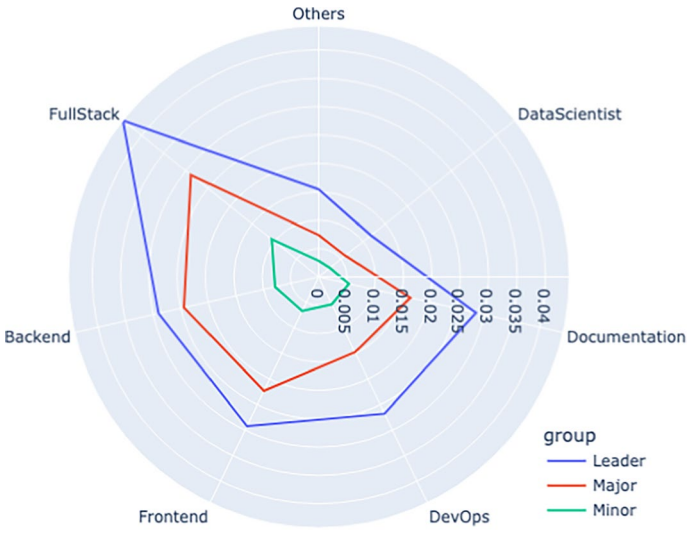

Fig. 15 Technical Roles in Organizational Coupling

Furthermore, we investigate the potential correlation between the traits of the 38 selected projects and the organizational coupling caused by these project teams. Interestingly, we find that there is a very poor correlation between the average inter-microservice organizational coupling and the number of developers (0.236) or between that and the number of commits (0.132). For example, the project *emissary-ingress/emissary* with 50 developers and over 1.1k commits has an average yearly organizational coupling of 28.28 when another project *1-Platform/one-platform* with 19 developers and around 750 commits has the value of 7782.27. Such a result can imply that the organizational coupling is independent of the project properties, which requires special awareness from practitioners. Meanwhile, nearly no correlation is detected between the average inter-microservice organizational coupling and the project size (0.07) or between that and the number of microservices (−0.05).

5 Discussion

Using a large established dataset of 145 microservice projects [4], we selected 38 projects with a sufficient number of contributors and commits as a representative OSS sample. This project sample did not adopt the same strategy as suggested for proprietary software (closed-source) projects. In the sample analyzed, we identified that developers typically work on multiple microservices and focus on various features, often in parallel. These conclusions are also confirmed by the vast majority of projects when considering different developer roles.

One of the explanations might be the dynamics of OSS projects. In OSS projects, developers commit their time voluntarily at random, non-fixed hours and schedules, oftentimes driven by feature priority requests or error reports. In particular, none of the selected projects is directly sponsored by a company that allocates developers to the project. Therefore, developers commonly select a set of issues to be implemented (either new features or bug fixing) and work on them independently rather than adopting the specific microservice that they maintain. Another explanation might be that, despite the decentralized nature of OSS and the microservice architecture, OSS

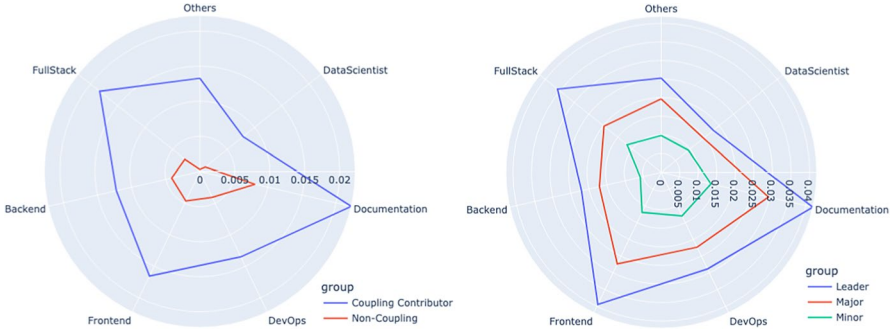

Fig. 16 Technical Roles for Developers Creating Couplings

projects might not yet have assimilated this strategy. Another reason might be the lack of clear teams in OSS projects (i.e., each developer does not belong to a specific team), and therefore the "one microservice per team/developer" strategy might not be perceived as an issue.

It must also be recognized that additional effort and overhead are related to the "one microservice per team/developer" strategy. However, this might not be the proper fit for the OSS environment and context. OSS projects are often driven by small development teams or individuals who stand behind the entire project; occasionally OSS projects have professional teams behind them (e.g., Red Hat); however, we did not include these projects in the study.

The microservice architecture is the main architecture for native cloud systems. However, not necessarily all microservice systems are cloud-native. In a similar parallel, the decentralized development model connected with cloud-native systems might collide with the OSS development model. Perhaps the main driver for the microservice architecture in these OSS projects is scalability, and the decentralized development aspect goes away with the OSS model.

As practitioners often suggest [4–8], if the development team is too small to be split into multiple teams, and there are multiple microservices, to respect the "one microservice per team/developer" strategy, the system should rather remain monolithic. The reasons for OSS might be prioritized system scalability for the price of this strategy violation. Perhaps some projects might have decided to split their systems into multiple microservices for maintainability reasons, to increase the separation of concerns, or to better identify different business domains, independently from the team that is working on the same services.

Another explanation might be given by Mariusz [59], who investigated whether Conway's Law applies to OSS projects and concluded that teams "organize themselves spontaneously around tasks, and since those tasks concern software modules, teams naturally follow Conway's Law."

An additional aspect revealed by our analysis relates to the influence of documentation on project structure and developer contributions. As observed in Figs. 6, 12, and 13, documentation commits appear to have a significant influence on the results, and many developers contribute to this role. This raises the question of whether including documentation-related commits affects the adherence to the "one microservice per team/developer" principle. It may not be entirely reasonable to consider documentation commits as a violation of this principle. However, it is worth noting that such commits often result in organizational coupling within a project, as developers from different microservices contribute to the same documentation. This is an interesting finding and provides flexibility for users of our metric. Based on their specific needs, they can choose to either include or exclude documentation-related commits when evaluating adherence to the "one microservice per team/developer" principle.

The result of this study will serve the practitioners' community in understanding how OSS microservice projects are being developed. Moreover, it will help researchers to further investigate the "one microservice per team/developer" strategy.

6 Threats to validity

In this section, we discuss the threats that might affect the validity of our multiple empirical studies according to Yin [60]: *Construct*, *Internal*, *External*, and *Reliability*.

Construct Validity. Replying to $\mathbf{RQ}_1$ we tried to understand if the "one microservice per team/developer" strategy is adopted. However, we measure how well this strategy is adopted in an OSS context by analyzing individual developer behavior and assuming that a single developer belongs to one team at a time. We recognized that this assumption could lead to some threats. We plan to expand our work in the future by adding information (such as developer communications, and issue/pull request comments) to extract teams to fine-grain our analysis. Additionally, we acknowledge the study's lack of ground truth regarding team organization within the analyzed projects. While we reviewed available information from the project repositories, we did not contact the developers directly. We agree that this is a limitation and recognize that this may restrict our ability to fully answer the research questions based on the available data. Such threats shall be addressed in the future with industrial cases studies as well as interviews and questionnaires with the experienced developers.

Internal Validity. The dataset used is one of the most recent in the context of microservices and open-source projects. However, the dataset is very heterogeneous (for the number of microservices, the age of the projects, and the number of developers), and we could only analyze a subset of the projects. We want to extend the dataset to get a better picture of the real state of the art. Such threats are can be addressed when the existing projects grow more mature. On the other hand, other types of developer behavior data, e.g., issue handling, communication, etc. will also be taken into account to address such threats.

External Validity. The findings of this paper can be simply extended when more microservice projects are taken into account. It is reasonable that all the currently included projects shall also inevitably evolve when the proposed method is replicated with the results updated. Especially for $\mathbf{RQ}_2$, the findings can also be generalized to projects that are not specifically microservice-based if we use modules or features to functionally separate the projects instead of using microservices. In this way, such extended findings shall provide insights into the collective contributor profiles for any given OSS project scope. In addition, when the language-role relations can be further defined (e.g., new roles defined, new languages assigned to different roles, etc.), the findings can also be updated accordingly with the changes conducted in the reference table. When microservice architecture is more widely adopted and the sample number of microservice-based projects grows, especially when more industrial cases are involved. Furthermore, the influence of other external factors shall be explored, e.g., the developers' personality, which can also contribute to the potential decoupling strategy towards organizational structure optimization.

Reliability. Using the dataset we provided in the replication package with the same approach, the practitioners and scholars can easily obtain the same results as described above. Only when any changes are introduced in the data itself or when the interpretation of the obtained factors varies based on different expertise,

do the findings differ accordingly. Such threat can be addressed when the dataset is continuously maintained and enhanced with new data added. In addition, the different tech roles shall also be maintained and validated by continuously retraining models with the new inputs.

7 Conclusion

Based on the suggestion of practitioners that "a developer should have exactly one service unless there is a proven need to have multiple services" and the assumption that open source software developed using a bazaar-style software development model would encourage a "one microservice per team/developer" strategy, we learned in this study that OSS projects do not comply with this strategy. Oftentimes, we could identify projects with a greater number of microservices than project contributors, and the OSS development model with a few main contributors dominated the proprietary software strategy. Still, we must assume that the contributor dedication to OSS has a very different dynamics than fully-funded organization projects that can afford multiple developers with regular commitments to contribution. One might question if Conway's Law collides with the OSS development model, and the results of this study add weight to the doubts. In this work, we showed that OSS microservice projects rarely follow the "one microservice per team/developer" strategy.

We have demonstrated this by analyzing the OSS project source code repositories of an established microservices project dataset. We further supported this result by analyzing the different developers' roles in contributing to these projects.

As future work, we aim to further study if "one microservice per team/developer" holds in OSS projects by trying to observe emerging or stable developer-like collaborations between developers. To do so, we plan to analyze the commits of source code repositories of microservice projects, also parsing the actual code modifications to understand if a collaboration took place. Also, following the suggestion by Mariusz [59], which states that "developers organize themselves spontaneously around tasks," we plan to study issue-tracking systems in combination with source code repositories to investigate if we can detect such spontaneous developers acting on single microservices. Moreover, we aim to investigate the developers' team composition to classify the developers who contribute to the same code.

Appendix A

See Table 2.

Table 2 The list of selected projects

No	Project URL
P1	http://github.com/1-Platform/one-platform
P2	http://github.com/Azure-Samples/container-apps-store-api-microservice
P3	http://github.com/CDLUC3/merritt-docker
P4	http://github.com/EYBlockchain/nightfall_3
P5	http://github.com/EdwinVW/pitstop
P6	http://github.com/FraunhoferISST/diva
P7	http://github.com/MongkonEiadon/VehicleTracker
P8	http://github.com/MozillaSecurity/orion
P9	http://github.com/OpenCodeFoundation/eSchool
P10	http://github.com/OpenLiberty/guide-getting-started
P11	http://github.com/SeldonIO/seldon-core
P12	http://github.com/SuperMap/GAF
P13	http://github.com/TaitoUnited/full-stack-template
P14	http://github.com/WilliamXu96/ABP-MicroService
P15	http://github.com/abpframework/abp-samples
P16	http://github.com/aliyun/alibabacloud-microservice-demo
P17	http://github.com/apache/juneau
P18	http://github.com/aws-containers/eks-app-mesh-polyglot-demo
P19	http://github.com/chayxana/Restaurant-App
P20	http://github.com/delimitrou/DeathStarBench
P21	http://github.com/dotnet-architecture/eShopOnContainers
P22	http://github.com/emissary-ingress/emissary
P23	http://github.com/geoserver/geoserver-cloud
P24	http://github.com/hyperledger-labs/blockchain-carbon-accounting
P25	http://github.com/imgproxy/imgproxy
P26	http://github.com/ivanpaulovich/clean-architecture-manga
P27	http://github.com/jbw/TooBigToFailBurgerShop
P28	http://github.com/kanisterio/kanister
P29	http://github.com/koderover/zadig
P30	http://github.com/kriswep/graphql-microservices
P31	http://github.com/learningOrchestra/learningOrchestra
P32	http://github.com/microservices-demo/microservices-demo
P33	http://github.com/networknt/light-example-4j
P34	http://github.com/pace/bricks
P35	http://github.com/rodrigorodrigues/microservices-design-patterns
P36	http://github.com/sourcefuse/loopback4-microservice-catalog
P37	http://github.com/wso2/product-microgateway
P38	http://github.com/xjantoth/microservice

Acknowledgements This work was supported by a grant from the Academy of Finland (grant n. 349488 - "MuFAno") and a grant from Business Finland ("6 G-Bridge 6GSoft").

Author contributions Xiaozhou Li: Software, Formal analysis, Investigation, Writing - original draft, Visualization. Dario Amoroso d'Aragona: Software, Formal analysis, Investigation, Writing - original draft, Visualization. Tomas Cerny: Conceptualization, Investigation, Writing - review and editing. Valentina Lenarduzzi: Conceptualization, Investigation, Writing - review and editing. Davide Taibi: Conceptualization, Methodology, Funding acquisition, Writing - review and editing, Supervision. Andrea Janes: Conceptualization, Investigation, Writing - review and editing.

Funding Open Access funding provided by University of Oulu (including Oulu University Hospital). This work was supported by a grant from the Academy of Finland (grant n. 349488 - "MuFAno") and a grant from Business Finland ("6G-Bridge 6GSoft"). This material is also based upon work supported by the National Science Foundation (NSF) under Grant No. 2409933.

Data availability To allow verifiability and replicability, we published the raw data in the replication package(https://dx.doi.org/10.6084/m9.figshare.27153711).

Declarations

Conflict of interest No Conflict of interest.

Ethical approval Not applicable.

Consent for publication All the authors have approved the publication of this work.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Fowler M (2006) CodeSmell. https://martinfowler.com/bliki/CodeSmell.html
2. Newman S (2015) Building microservices, 1st edn. Oreilly & Associates Inc., USA
3. Reinfurt M (2021) The horror of microservices in small teams - and why you shouldn't build them. https://manuelreinfurt.medium.com/the-horror-of-microservices-in-small-teams-208f6660dd5
4. Baresi L, Quattrocchi G, Tamburri DA (2022) Microservice architecture practices and experience: a focused look on docker configuration files. https://doi.org/10.48550/ARXIV.2212.03107 . https://arxiv.org/abs/2212.03107
5. Balalaie A, Heydarnoori A, Jamshidi P (2016) Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Softw 33(3):42
6. Amazon: service per team pattern (2023) https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/service-per-team.html
7. Richardson C (2021) A pattern language for microservices. https://microservices.io/patterns/decomposition/service-per-team.html
8. Qcon2022 (2022) Dark energy, dark matter and the microservices patterns?! https://qconsf.com/presentation/oct2022/dark-energy-dark-matter-and-microservices-patterns
9. Wolff E (2021) Microservices primer: a short overview. Leanpub, Canada
10. Conway ME (1968) How do committees invent. Datamation 14(4):28–31
11. Carneiro C, Schmelmer T (2016) Microservices from day one: build robust and scalable software from the start. Apress, USA

12. Raymond ES, O'Reilly T (1999) The cathedral and the bazaar, 1st edn. O'Reilly & Associates Inc, USA
13. Parnas DL (1972) On the criteria to be used in decomposing systems into modules. Commun ACM 15(12):1053–1058
14. Walpita P (2020) Coupling and cohesion in microservices. https://priyalwalpita.medium.com/coupling-and-cohesion-in-microservices-235ed9203843
15. Perry DE, Staudenmayer NA, Votta LG (1994) People, organizations, and process improvement. IEEE Softw 11(4):36–45
16. Zhong C, Zhang H, Li C, Huang H, Feitosa D (2023) On measuring coupling between microservices. J Syst Softw 200:111670
17. Amoroso d'Aragona D, Pascarella L, Janes A, Lenarduzzi V, Taibi D (2023) Microservice logical coupling: a preliminary validation. In: 2023 IEEE 20th international conference on software architecture companion (ICSA-C), IEEE
18. d'Aragona DA, Li X, Cerny T, Janes A, Lenarduzzi V, Taibi D (2023) One microservice per developer: Is this the trend in OSS? In: 10th European conference on service-oriented and cloud computing (ESOCC)
19. Li X, d'Aragona DA, Taibi D (2023) Evaluating microservice organizational coupling based on cross-service contribution. In: International conference on product-focused software process improvement, Springer
20. Joblin M, Mauerer W, Apel S, Siegmund J, Riehle D (2015) From developer networks to verified communities: a fine-grained approach. In: Int. Conf. on software engineering
21. Panichella S, Bavota G, Penta MD, Canfora G, Antoniol G (2014) How developers' collaborations identified from different sources tell us about code changes. In: Int. Conf. on software maintenance and evolution
22. Bird C, Pattison D, D'Souza R, Filkov V, Devanbu P (2008) Latent social structure in open source projects. In: Int. symposium on foundations of software engineering
23. Meneely A, Williams L (2011) Socio-technical developer networks: Should we trust our measurements? In: Int. Conf. on software engineering
24. Ashraf U, Mayr-Dorn C, Mashkoor A, Egyed A, Panichella S (2021) Do communities in developer interaction networks align with subsystem developer teams? An empirical study of open source systems. In: Int. Conf. on software and system processes
25. Lenarduzzi V, Sievi-Korte O (2018) On the negative impact of team independence in microservices software development. In: International conference on agile software development (XP)
26. Bogner J, Fritzsch J, Wagner S, Zimmermann A (2021) Industry practices and challenges for the evolvability assurance of microservices: an interview study and systematic grey literature review. Empir Softw Eng 26(5):1–39
27. Ortu M, Hall T, Marchesi M, Tonelli R, Bowes D, Destefanis G (2018) Mining communication patterns in software development: a github analysis. In: Int. Conf. on predictive models and data analytics in software engineering, PROMISE'18
28. Shi K (2021) Establishing contributor roles within software repositories by mining architectural information. https://fse.studenttheses.ub.rug.nl/25608/1/bCS_2021_ShiK.pdf
29. Spinellis D, Kotti Z, Kravvaritis K, Theodorou G, Louridas P (2020) A dataset of enterprise-driven open source software. In: Int. Conf. on mining software repositories
30. Blincoe K, Harrison F, Kaur N, Damian D (2019) Reference coupling: an exploration of inter-project technical dependencies and their characteristics within large software ecosystems. Inf Softw Technol 110:174
31. Wiese IS, Ré R, Steinmacher I, Kuroda RT, Oliva GA, Treude C, Gerosa MA (2017) Using contextual information to predict co-changes. J Syst Softw 128:220
32. Brooks FP Jr (1995) The mythical man-month: essays on software engineering. Pearson Education, England
33. Nagappan N, Murphy B, Basili V (2008) The influence of organizational structure on software quality: an empirical case study. In: Proceedings of the 30th international conference on software engineering, pp 521–530
34. Mockus A (2010) Organizational volatility and its effects on software defects. In: Proceedings of the Eighteenth ACM SIGSOFT international symposium on foundations of software engineering, pp 117–126
35. Isern D, Sánchez D, Moreno A (2011) Organizational structures supported by agent-oriented methodologies. J Syst Softw 84(2):169–184

36. Li X, Abdelfattah AS, Yero J, d'Aragona DA, Cerny T, Taibi D (2023) Analyzing organizational structure of microservice projects based on contributor collaboration. In: 2023 IEEE international conference on service-oriented system engineering (SOSE), IEEE, pp 1–8
37. Ashraf U, Mayr-Dorn C, Mashkoor A, Egyed A, Panichella S (2021) Do communities in developer interaction networks align with subsystem developer teams? an empirical study of open source systems. In: 2021 ICSSP and ICGSE, IEEE, pp 61–71
38. Briand LC, Morasca S, Basili VR (1996) Property-based software engineering measurement. IEEE Trans Softw Eng 22(1):68–86
39. Allen EB, Khoshgoftaar TM, Chen Y (2001) Measuring coupling and cohesion of software modules: an information-theory approach. In: Proceedings seventh international software metrics symposium, IEEE, pp 124–134
40. Poshyvanyk D, Marcus A (2006) The conceptual coupling metrics for object-oriented systems. In: 2006 22nd IEEE international conference on software maintenance, IEEE, pp 469–478
41. Gupta V, Kumar Chhabra J (2009) Package coupling measurement in object-oriented software. J Comput Sci Technol 24:273–283
42. Hammad M, Rawashdeh A (2014) A framework to measure and visualize class coupling. Int J Softw Eng Appl 8(4):137–146
43. Pedraza-Coello R, Valdés-Souto F (2021) Measuring coupling in microservices using cosmic measurement method. ICSEA 2021:26
44. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Germany
45. Montandon JE, Valente MT, Silva LL (2021) Mining the technical roles of GitHub users. Inf Softw Technol 131:106485
46. Bartholomew DJ, Knott M, Moustaki I (2011) Latent variable models and factor analysis: a unified approach, vol 904. Wiley, United Kingdom
47. Cattell RB (1988). In: Nesselroade JR, Cattell RB (eds) The meaning and strategic use of factor analysis. Springer, Boston, MA
48. Hair JF, Black WC, Babin BJ, Anderson RE, Tatham RL et al (2006) Multivariate data analysis (Vol. 6). Upper Saddle River, NJ: Pearson Prentice Hall
49. DeCoster J (1998) Overview of factor analysis. http://www.stat-help.com/notes.html
50. Snedecor GW, Cochran WG (1989) Statistical methods. Ames Iowa State Univ Press Iowa 54:71
51. Kaiser HF (1974) An index of factorial simplicity. Psychometrika 39(1):31
52. Horn JL (1965) A rationale and test for the number of factors in factor analysis. Psychometrika 30(2):179
53. Kaiser HF (1958) The varimax criterion for analytic rotation in factor analysis. Psychometrika 23(3):187
54. Kumar BV, Hassebrook L (1990) Performance measures for correlation filters. Appl Opt 29(20):2997
55. Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code! examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on foundations of software engineering
56. Lima J, Treude C, Filho FF, Kulesza U (2015) Assessing developer contribution with repository mining-based metrics. In: 2015 IEEE international conference on software maintenance and evolution (ICSME), pp 536–540. https://doi.org/10.1109/ICSM.2015.7332509
57. Guadagnoli E, Velicer WF (1988) Relation of sample size to the stability of component patterns. Psychol Bull 103(2):265
58. Miranda G, Montandon JE, Valente MT (2022) Techspaces: Identifying and clustering popular programming technologies. In: Brazilian symposium on software components, architectures, and reuse
59. Kamola M (2019) How to verify Conway's law for open source projects. IEEE Access 18:38469
60. Yin RK (2014) Case study research: design and methods, 5th edn. SAGE Publications Inc, USA

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Xiaozhou Li[1] · Dario Amoroso d'Aragona[2] · Tomas Cerny[3] · Valentina Lenarduzzi[1] · Davide Taibi[1] · Andrea Janes[4]

✉ Valentina Lenarduzzi
valentina.lenarduzzi@oulu.fi

Xiaozhou Li
xiaozhou.li@oulu.fi

Dario Amoroso d'Aragona
dario.amorosodaragona@tuni.fi

Tomas Cerny
tcerny@arizona.edu

Davide Taibi
davide.taibi@oulu.fi

Andrea Janes
andrea.janes@unibz.it

[1] University of Oulu, Oulu, Finland

[2] Tampere University, Tampere, Finland

[3] University of Arizona, Tucson, USA

[4] Free University of Bozen-Bolzano, Bozen-Bolzano, Italy