Analyzing Organizational Structure of Microservice Projects based on Contributor Collaboration

Xiaozhou Li
M3S; CloudSEA.AI
University of Oulu; Tampere University
Oulu, Finland; Tampere, Finland
xiaozhou.li@oulu.fi

Amr S. Abdelfattah
Department of Computer Science
Baylor University
Waco, Texas, USA
amr_elsayed1@baylor.edu

Jorge Yero
Department of Computer Science
Baylor University
Waco, Texas, USA
jorge_yero1@baylor.edu

Dario Amoroso d'Aragona
CloudSEA.AI
Tampere University
Tampere, Finland
dario.amorosodaragona@tuni.fi

Tomas Cerny
Systems & Industrial Engineering
University of Arizona
Tucson, Arizona, USA
tcerny@arizona.edu

Davide Taibi
M3S; CloudSEA.AI
University of Oulu; Tampere University
Oulu, Finland; Tampere, Finland
davide.taibi@oulu.fi

***Abstract*—Software system quality is strongly affected by the organizational structure and collaboration across developers. Effective and loosely coupled organization structures reflect the high quality of the system architecture and the efficiency with which this system can evolve. Especially for microservice-based systems, as the notion of "one-microservice-per-team" is highly recommended and advocated as one of the best practices in the industry, it is crucial for the companies to be aware of the status of their organizational structure and the critical contributors therein. To such an end, this paper proposes an approach to analyze the organizational structure of microservice-based software projects in terms of contributor collaboration and to identify the core contributors therein. Furthermore, we can also monitor the evolution of the project's organizational structure via the growing collaboration activities through different releases. The proposed method shall help the companies and organizations adopting microservices better understand their organizational structure and make more effective decisions in maintaining the quality of microservice architectures.**

***Index Terms*—Microservice; Organizational Structure; Collaboration; Social Network Analysis; Software Evolution; Mining Software Repositories**

I. Introduction

Microservice architecture is gaining increasing popularity when academia has been investigating its adoption strategies, best practices, and issues [1], [2]. Though both the technical and organizational aspects of microservice adoption need to be considered, the organizational side is often overlooked [3]. For large programming projects, the purpose of the organization is to reduce the amount of communication and enable effective collaboration, which is critical for the project's success [4]. Many metrics of project organization structure, such as the developer team size, contribution frequency, organization intersection factor, etc., can be seen as predictors for project failure [5]. Therefore, it is important for any software project and the critical stakeholders therein to be aware of the well-functioning of their organizational structure. Especially for large microservice projects, organizational challenges need to be addressed when the companies migrate from monolithic systems to microservices, where the structure and skills of the organization should also support the new architecture [6]. Regarding the relation between the organization structure and system architecture, Melvin E. Conway states that *any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure*, known publicly as the *Conway's Law* [7]. For microservice-based systems, Conway's Law is also contributing to the decentralized governance practice in the industry [8]. However, limited studies have contributed to investigating the decentralization of organizations corresponding to potentially similar microservice architecture and the best practices.

On the other hand, such decentralized organization of microservice projects also influences the work of individual software designers and developers in terms of their communication and collaboration [9]. Since responsibilities are assigned to microservices, a similar division in responsibilities and labor should exist for developers [10]. It is important for any software project managers to be aware of the developer network within their organization in terms of collaboration when such collaboration is latent from version control systems, e.g., Github [11]. For microservice-baed projects specifically, it is also important to effectively allocate the best-fitting developers and their suitable collaborators to specific microservices they're more familiar with. Such practice requires not only a clear understanding of the overall organizational structure and software architecture but also the traits and characteristics of each individual developer.

Meanwhile, accompanied with the inevitable evolution of the system, the microservice architecture evolves and, to some extent, degrades if not properly maintained [12]. The asynchronous changes in the architecture and organizational structure shall influence the performance of the maintenance team as well as the sustainable quality of the architecture itself. To cope with such changes, system evolution assessments

are often conducted focusing on the architectural changes across systems versions. However, the impact of such system evolution on the organizational structure, and vice versa, shall be taken into account and carefully addressed [13]. Therein, analyzing the evolution of developer collaboration networks as a reflection of the organizational structure could serve such a purpose.

Therefore, the goal of this paper is to investigate the methods to assess the organizational structure and identify the key contributors in terms of collaboration amongst the microservice projects. Furthermore, we also provide ways of tracking the changes in such collaboration and contributor rankings through the software evolution process.

In order to investigate our aforementioned goal, we formulated two Research Questions (RQs):

RQ1: *How to evaluate the organization's collaboration structure and identify the core contributors within?*

RQ2: *How to monitor the changes in the organization structure and the importance of contributors during system evolution?*

We propose adopting the social network analysis (SNA) methods to visualize and assess the organizational structure of microservice projects with the collaboration data crawled from GitHub repositories. Here, we apply a combination of mining software repository techniques together with network modularity analysis [14]. We can also evaluate the criticality of any contributors of a particular project by calculating their centralities [15] in the collaboration network, and further identify the core contributors with high centrality scores. Herein, we consider *core contributors* as the particular developers who are more central in the collaboration network. Therefore, the more collaborative or central is a developer, the higher impact on the organization it will be when such developers are available. This will answer RQ1. Furthermore, we also propose a method to track and compare the collaboration networks of the different releases of the project reflecting the evolution of their organizational structure. In addition, the changes in each contributor's centrality together with the evolution of the system shall also be monitored. This will answer RQ2. The main contribution of our work is to provide a comprehensive method to construct, visualize and monitor the organizational structure of microservice projects in terms of the collaboration network amongst contributors. It shall help the stakeholders better understand such structure and enable effective decision-making.

The remainder of this paper is structured as follows. Section II summarizes related works. Section III describes the research method applied. Section IV presents the results, while Section V discusses them. Finally, Section VI draws conclusions and highlights future works.

II. Related Work

Developer collaboration has been the subject of multiple studies in open-source projects from different perspectives. Researchers have used various methods to analyze the structure of the developers and their stability in software projects. One of the methods used is to build a collaboration network to investigate the interactions between developers. Wolf et al. [16] investigated the impact of developer collaboration on the outcome of code integration processes and found that collaboration plays a crucial role in determining the quality of software integration. They also developed a model that utilizes network measures to predict whether integration will fail based on the current communication structure of a development team. Singh studied the impact of macro-level properties of developer collaboration on open-source software success [17]. The study shows that a small-world network structure of the OSS community leads to the success of the projects in terms of code development and user acceptance. Surian et al. extracted detailed topological graph patterns from a large developer network [18]. Their results show that even for large developer communities, not all developers are connected to each individual other peer and collaboration clusters exist commonly. They also validated the existence of the small-world phenomenon where each developer can be reached within no more than six connections.

Meanwhile, many studies also proposed methods and techniques to analyze one or multiple attributes of the collaboration networks. For example, Peng et al. [19] applied network analysis techniques to analyze and measure the structural stability of the evolving developer collaboration network in open-source projects. Surian et al. [18] investigated the most common topological pattern in collaboration networks in open-source projects. They have proposed a method to mine this network in multiple projects. Furthermore, El Asri et al. [20] proposed a method to detect peripheral contributors, and applied it with an adaptive time-frame incremental approach to clustering and locating contributors in different temporal networks, and detected common temporal patterns.

Other studies have looked at developer collaboration and network analysis as a way to predict and detect failures in software projects. Meneely et al. [21] created a collaboration graph by tracking changes in files and releases, linking developers based on their collaboration on the same files and revisions. They used various methods of social network analysis to predict software defects. Zimmermann et al. [22] also identified defect-prone units using network analysis, but they focused on analyzing dependency graphs of systems. The aforementioned studies as well as many other works investigate the existing network structures in terms of the collaboration amongst developers. They focus more on unfolding the community structure than evaluating the overall quality and exploring the connection to the other latent relations of software projects. In this study, we evaluate the modularity of developer collaboration network through the system evolution and identify key developers within the system using centrality analysis.

In the context of microservices, which are known for their large distribution and use of multiple programming languages, there are few studies that investigate collaboration among developers in microservices-based systems. Gustafsson et al. [23] manually examined the collaboration practices of developers

in three microservices projects. They attempted to identify patterns in the organization of these projects and whether teams were adhering to Conway's Law recommendations. They also identified core developers based on the number of files they have committed to. However, their findings suggest that Conway's Law has a lower limit of applicability regarding project size and maturity, which calls for further research to validate these pilot results.

Other studies have used network analysis techniques to examine different aspects of microservice architecture. Gaidels et al. [24] used a 37-microservice online banking application to investigate the feasibility of applying network analysis on the service dependency graph, but the study only shows the feasibility of such application but fell short in contributing to solving issues in microservice architecture reconstruction, quality evaluation or anomaly detection. Gamage et al. [25] proposed a tool that uses dependency graph and graph theory algorithms to evaluate microservice architecture and to identify anti-patterns, but the tool is capable of tracking only synchronous systems which communicate in RESTful style.

The studies mentioned above use dependency graphs and network analysis on the connection relations between microservices and aim to validate the feasibility of the methods. Especially regarding the dependency graphs of microservices architecture, many studies contribute to the reconstruction of different architecture views using static and dynamic analysis. For example, Bushong et al. proposed a method to analyze microservice systems for the generation of data models and communication diagrams using static analysis on Java source code [26]. Al Maruf et al., proposed methods to use dynamic analysis on telemetry data to generate the service dependency graph of microservice-based systems which aims to facilitate the monitoring of such systems and to detect the potential anomalies and quality deterioration therein [27]. These studies focus on the reconstruction and visualization of the existing microservice architecture. But based on our knowledge, no studies have been conducted on the investigation of developer collaboration in microservice-based projects and the reconstruction of the developer organization structure.

Furthermore, community detection methods have been approved being useful in detecting the latent network clusters and patterns of software communities. Hou et al. proposed a community detection algorithm based on the cooperation intensity of developers to identify the many different types of developer networks in the GitHub software ecosystem [28]. Shen et al. proposed a multi-objective optimization model for community detection in software ecosystem network based on developer collaboration intensity and programming language similarity [29]. Meanwhile, network centrality and modularity are also often adopted as important metrics to assess the criticality of network nodes and the strength of network division. For example, He et al. applied centrality analysis to investigate the developers' behaviors in the OSS community and found different kinds of collaboration patterns and correlations amongst different centrality values [30]. Jermakovics et al. proposed an approach to construct and visualize the developer networks and a filtering method to improve the modularity of the network [11]. However, the application of such methods in investigating the developers' collaboration in microservice-based projects is still limited.

III. Methods

A. Data Pre-Processing

First, we crawl all the commits from the project repository and group them by releases. For each commit, we identify the related files changed, for each of which we identify the developers committing to it. We identify two developers' collaboration when they both commit to a file and use the amount of time they modify the same file as the weight of the collaboration relation. By doing so, we obtain the collaboration network with each node being a developer and each edge as the collaboration between two developers. On the other hand, as herein each microservice is organized as a separate folder in the repository, we consider any developer, who committed to a file in a microservice folder, to be a contributor to this microservice. In this way, we can also construct the collaboration network on the microservice level with the weight of each edge being the number of microservices to which the two developers both contribute.

B. Evaluate Organization Collaboration Structure

For any real-world graphs, e.g., social networks, it is also critical but difficult to detect the communities or clusters within. A network community is commonly seen as a set of vertices in a graph within which the connections amongst the nodes are denser than those towards the rest of the network [31]. Thus, community detection is necessary for the identification of such communities of a particular network so that the structure of it can be revealed. In general, two types of community detection methods are commonly adopted in practice: 1) to partition graphs by identifying and removing the "spanning" links between densely-connected regions, e.g., Girvan-Newman algorithm [32]; 2) to assemble nodes that are likely to belong to the same region, e.g., Louvain algorithm [33].

From the collaboration network identified in the preprocessing step, we adopt the Louvain community detection algorithm to extract the latent communities in the collaboration networks [33]. It is commonly used for the structure extraction of a large weighted network with optimized modularity value. As a key measure for network community quality, the modularity of a network is a measure indicating the strength of the division of a network into communities [14]. High-modularity networks have dense connections between the vertices within communities and comparatively sparse connections between vertices between different communities. Simply put, networks with higher modularity are stronger connected with the nodes therein.

Given any network which is defined by V and E, where V is the set of vertices and E is the set of edges of the network, we assume m is the number of communities by which the target

network is partitioned into. Therefore, the network modularity Q is then calculated as follows.

$$Q = \sum_{k=1}^{m} [\frac{l_k}{|E|} - (\frac{d_k}{2|E|})^2] \quad (1)$$

where l_k is the number of edges between any two nodes from the k-th community; and d_k is the sum of the degree of all those nodes.

Using the Louvain community detection method, each node is assigned to a community when Q is maximized. ΔQ indicating the increased value of Q when moving node i to community C, which is calculated as

$$\Delta Q = \frac{\sum_C + k_i^C}{2n} - (\frac{\sum_{\widehat{C}+k_i}}{2n})^2 - [\frac{\sum_C}{2n} - (\frac{\sum_{\widehat{C}}}{2n})^2 - \frac{k_i}{2n}] \quad (2)$$

where k_i is the sum of weighted edges incident to i; k_i^C is the sum of the edges from i to nodes in the community C; $\sum_C$ is the sum of the weighted edges in C; $\sum_{\widehat{C}}$ is the sum of the edges incident to nodes in C; and n is the sum of the weights of all the edges of the network.

The method is to detect the optimized community structure of a network by moving nodes from one community to another in order to detect the significantly improved ΔQ [34]. Considering the fact that the nature of a high-modularity network complies with the well-known "low coupling, high cohesion" in software engineering, it is reasonable to use the modularity value to represent the relevant quantification of the organization structure. According to Fortunato and Barthelemy, a value of Q larger than 0.3 – 0.4 is a clear indication that the subgraphs of the corresponding partition are modules [35]. Herein, such a value shall similarly indicate the project organization has decent "low coupling, high cohesion" quality in terms of contributor collaboration.

C. Identify the Critical Contributors in Organization

Two classic centrality measures are usually considered for such a purpose, including closeness centrality and betweenness centrality [36] Both closeness centrality and betweenness centrality can be used to evaluate the importance of any particular developer within the project collaboration network [36].

Closeness centrality is an important index based on the geodesic distances from the network vertices to all others. It is a metric used to identify how long it shall take for information to travel from a particular vertex to the others in the network, i.e., how close is it to them [37]. Let V be the set of vertices of a given network, where i is a particular vertex within. The geodesic distance between i and another vertex $j \in V$ is denoted as $d_G(i,j)$. Thus, as the closeness centrality is defined as the inverse of the average distance [**?**], the closeness centrality of i, $C_C(i)$, is calculated as follows.

$$C_C(i) = \frac{1}{\sum_{j \in V} D_G(i,j)} \quad (3)$$

Betweenness centrality, indicating the "brokering positions between others that provides an opportunity to intercept or influence their communication" [15], is based on the shortest paths through a particular node. Herein, the geodesic path of two individual nodes in the network is defined as the shortest path between them. For a particular node $i \in N$, the number of geodesic paths between another two nodes $h, j \in N$ via node i is denoted as g_{hij}. Meanwhile, the number of geodesic paths from h to j is denoted as g_{hj}. Then, the betweenness centrality of i, $C_B(i)$, is calculated as follows.

$$C_B(i) = \sum_{j,h \neq i} \frac{g_{hij}}{g_{hj}} \quad (4)$$

Both betweenness centrality and closeness centrality calculate the shortest paths of a vertex to the rest of the vertex pairs in a relatively large network [38]. Betweenness centrality measures the others' dependence on a particular node indicating which node has the most control [15]. Comparatively, closeness centrality measures the access efficiency of the specific node to the other nodes [39]. The developers of high betweenness centrality are the ones critically linking two sets of developers that tend to lose connection. Regarding the collaboration network of developers, the ones with the high closeness centrality are more closely collaborating with the others. Meanwhile, the developers of high betweenness centrality are the ones critically linking two sets of developers that tend to be separate from each other. If the high-betweenness developers are removed, the project then tends to have separate subgroups of developers with less communication across.

D. Monitor the Organization Structure Evolution

In order to monitor the changes in the organizational structure, we follow the changes in the collaboration networks; specifically, focusing on the changes in modularity and core contributors. We shall keep track of the changes through the different releases of the system, which is a common approach to analyzing the system evolution in terms of different performance metrics [40], [41]. Herein, we can observe the improvement or degradation of the collaboration relations and the changes in contributor ranking by using the collaborating data for different releases.

IV. Case Study

In this study, we demonstrate the applicability of the proposed method with a case study on an existing microservice-based system, *eShopOnContainer*[1], a sample .NET Core reference application, powered by Microsoft, based on a simplified microservices architecture and Docker containers. By adopting this case project as a proof-of-concept, we hereby present its collaboration network and its evolution through releases and identify the core contributors therein.

[1]eShop Container http://github.com/dotnet-architecture/eShopOnContainers

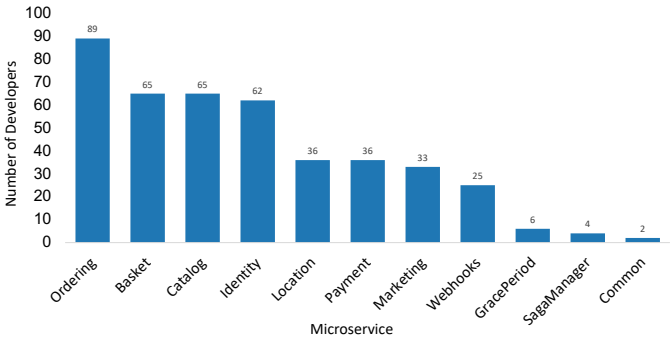

Fig. 1: Number of Developers for each Microservice

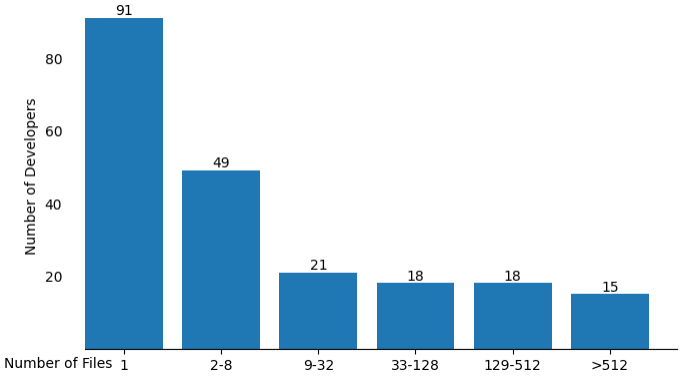

Fig. 2: Developers Distribution on File Coverage

A. Data Collection

We collect the 4 078 commits data of the project from 2016-09-06 to 2022-11-22 to establish latent collaboration relations. Therein, each commit data point contains the information of `committer id`, `date`, and `changed files`. 212 different committers contributed to all these commits with 59 321 changes to 6 403 different files. Furthermore, we also manually checked the architecture of the project and identified 11 microservices (by the time of the last collected commit), including *Ordering*, *Basket*, *Catalog*, *Identity*, *Location*, *Payment*, *Marketing*, *Webhooks*, *GracePeriod*, *SagaManager*, and *Common*. In order to investigate the collaboration relations on microservices, we identify the microservice on which each commit targets. As all the microservices are defined as individual sub-folders located in the *eShopOnContainers/src/Services/* folder, we can easily identify the service changed by each commit by locating the directory of the files changed.

The number of unique developers who contributed to each microservice is displayed in Fig. 1 while the number of files contributed by each developer is depicted in Fig. 2. We can observe the nature of this case project that for some microservices only very few developers have touched when some have been contributed by many. On the other hand, the developers' file coverages also vary greatly, as nearly half of the developers committed to only one file while 15 of them committed to more than 500 files.

Additionally, we also select five major releases of the project to observe its organizational structure evolution. The goal is to select the releases where the in-between timespans are similar so that the changes in terms of collaboration can be easily observed with more commits involved for each timespan.

B. Data Analysis

With the commit dataset, we construct and visualize the collaboration network establishing the collaboration relations amongst the developers with Gephi [42]. Herein, we define that a collaboration relationship is established when two contributors commit to the same file or microservice. Such a relation is depicted as an edge between the according nodes (i.e., contributors) in the constructed collaboration network. Furthermore, the weight of the edge is the number of files to which the two developers both contributed; because when the two developers co-contributed to more files, their connection is closer than that of the ones having fewer co-contributed files. To be emphasized, such an over-simplified collaboration definition is only used for this proof-of-concept. In real-life projects, the collaboration together with the social connection between developers is complex and of multi-perspectives.

TABLE I: Collaboration Network Info for Each Release

Release	Date	File Collaboration		MS Collaboration	
		Nodes	Edges	Nodes	Edges
2.0.5	2018-04-05	80	458	83	2967
2.0.8	2018-11-12	97	537	105	4517
2.2.0	2019-03-21	111	664	122	5972
3.0.0	2019-11-26	136	837	146	8455
5.0.0	2021-11-08	174	1037	197	15244

For the constructed collaboration network, we use the Louvain community detection algorithm to find the best communities and calculate the modularity [33]. In addition, by calculating the centrality of each contributor in the network, we can estimate their importance in terms of collaboration. Furthermore, by establishing the collaboration networks of the selected five major releases based on the commits pushed before the release time. Therefore, we can visualize and observe the changes in the collaboration network through these releases; meanwhile, together with the changes in contributor centrality, the core contributor status can also be tracked.

The number of nodes and edges of the collaboration networks (on files and microservices) for the selected releases are presented in TABLE I. We can observe the growth of the collaboration through the releases.

C. Results

By using the Louvain community detection method, in terms of file-level collaboration, the project (at Release 5.0.0) can be divided into three communities with modularity at 0.195 (in TABLE II). The modularity value is much lower than 0.3 which indicates a suboptimal partition [35]. Therefore, the result here shows that the communities are not well divided, which means the contributors are heavily collaborating across communities when it is likely that many are overburdened by

multiple unrelated modules of the systems. Similarly, considering collaboration on microservices, the modularity value is even lower (0.170). The visualized collaboration network is shown on the left of Fig. 3.

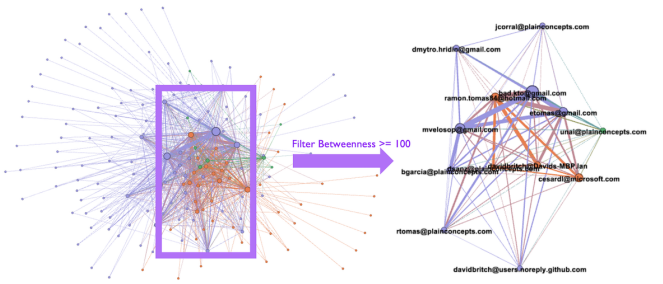

Fig. 3: The Collaboration Network and Core Contributors

Meanwhile, by calculating the betweenness centrality of each developer in the collaboration network, we can simply evaluate and compare their importance to the project in terms of their position in control. From the visualized network, by setting a threshold on centrality, we can simply extract the core contributors of the project. Shown in the right part of Fig. 3, by filtering out the developers with betweenness lower than 100 (an example threshold), we identify 13 core contributors. By choosing different thresholds, the circle of core contributors can certainly vary.

TABLE II: Network Modularity and Communities per Release

Release	File Collaboration		MS Collaboration	
	Modularity	Community	Modularity	Community
2.0.5	0.139	2	0.113	2
2.0.8	0.145	3	0.144	2
2.2.0	0.167	3	0.148	2
3.0.0	0.184	3	0.156	2
5.0.0	0.195	3	0.170	2

Furthermore, by visualizing the collaboration networks of different releases, we can easily observe the growth of the collaboration in terms of both files and microservices (shown in Fig. 4 and Fig. 5). Meanwhile, by tracking the evolution of the collaboration through the releases, we can observe that the modularity is increasing (shown in TABLE II). It indicates the organizational structure is gradually improving though the value is still not optimal. On the other hand, the collaboration network on microservice shows similar results. The modularity is also slightly increasing; though many developers are covering multiple microservices with their duties heavily coupled.

Furthermore, by comparing the centrality of the core contributors in different releases, we can also clearly observe their gaining or losing connections (shown in TABLE III). Herein, we only use the betweenness centrality value of each contributor as an example, as it is only to demonstrate the feasibility of the method. For example, the developer *mvelosop* start at a centrality of 7.93 at Release 2.0.5 and became the 2nd core contributor of the project with a centrality of 2222.74. Another two developers, *bad.kto* and *cesardl* also

TABLE III: Core Contributors Centrality Evolution

Contributor	2.0.5	2.0.8	2.2.0	3.0.0	5.0.0
bad.kto	695.19	1339.95	1925.43	3150.56	6925.14
mvelosop	7.93	55.33	259.62	1164.68	2222.74
cesardl	730.16	1111.82	1230.81	1441.75	1759.89
etomas	593.61	710.60	844.21	1353.30	1574.85
ramon.tomas84	408.13	514.60	732.44	709.44	802.14
bgarcia	0.00	0.00	0.00	0.00	508.70
davidbritch@D	163.17	194.78	226.83	268.91	365.16
dmytro.hridin	0.00	0.00	0.55	206.89	221.78
dsanz	69.15	73.15	83.98	94.50	163.41
jcorral	0.00	0.00	0.00	108.22	117.39
unai	20.76	115.84	92.99	98.91	115.55
davidbritch@u	91.50	81.66	71.96	97.36	108.16
rtomas	18.94	92.86	89.87	89.98	100.30

grew steadily and remained top contributors. There are also developers who lose collaboration within some periods, e.g., *unai* and *davidbritch@u*.

V. Discussion

Compared to the previous studies, our work focuses on the application of social network analysis methods, especially the network centrality and modularity analysis as well as the community detection, to the investigation of developer collaboration and organizational structure of microservice architecture. Compared to traditional monolithic systems, microservice projects are more demanding in terms of the quality of their organizational structure, as advocated by many practitioners, for microservice projects, it would be recommended that a team should own exactly one service unless there is a proven need to have multiple [43], [44]. Therefore, it is critical to investigate the internal structure of the microservice project organization in terms of developer collaboration before detecting the potential harming collaboration coupling and constructing the according solutions.

In general, the coupling is a critical factor for microservice architecture anti-patterns in terms of shared libraries, inappropriate service intimacy, and shared persistence [45]. Regarding microservice systems, logically coupled microservices can cause issues as multiple files will be changed with one single commit when, consequently, the risk of developers committing to services out of their domain expertise is high [46], [47]. On the contrary, loosely coupled services allow the developers to make changes without modifying other services [48]. Therefore, investigating the evolving coupling between services is critical for increasing the independence between teams and reducing the level of hazardous dependencies. Correspondingly, it is also critical to investigate the toxic coupling in the organization structure of microservice projects as it can be copied by the system structure [7]. Towards such an end, a synthesized study on the different system views together with the view of the organization structure shall contribute to solving the issues [26].

The results show that by mining the commit data of any given microservice project, we can evaluate its latent organizational structure via collaboration network construction (considering either file level or microservice level collaboration).

Fig. 4: Developer Collaboration Evolution on Files

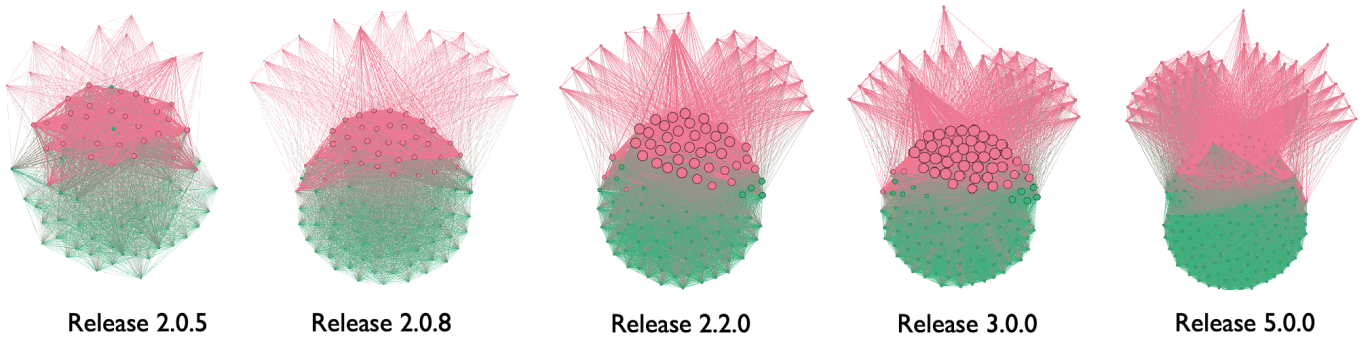

Fig. 5: Developer Collaboration Evolution on Microservices

Meanwhile, we can also identify the core contributors of the project by evaluating their centrality in the network. This shall answer RQ1. Furthermore, by tracking the evolution of such collaboration through releases, we can clearly observe the evolution of such structures and the growth of any individual contributor. This shall answer RQ2. To be noted, when adopting closeness centrality, the outcomes are similar. When considering the collaboration networks on the microservice level, the majority (11) of the high-centrality core contributors remain. In addition, these core contributors also contributed a largely above-average number of commits.

To be emphasized, the proposed method is limited concerning the ways of obtaining sufficient collaboration data. In the case study, the collaboration relations between developers (either on files or microservices) are largely simplified for the purpose of proof-of-concept. The collaboration data can be enhanced by mining issues and communications via chat or emails [49]. On the other hand, the scope of the project organization structure can be also enlarged by including more stakeholders [50]. For future work, we shall continue to investigate the adaptation of network modularity as a metric to measure the quality of microservice-based project organization in terms of contributor collaboration. We would also explore the approaches to identify and assess the organizational coupling issue in microservice projects and its relation to other types of couplings. Furthermore, we shall also investigate the potential solutions to fix the identified coupling issues in order to improve the overall quality of the project organization.

VI. Conclusion

In this study, we propose the use of social network analysis to evaluate the microservice project organizational structure in terms of contributor collaboration, identify the core contributors therein, and monitor the evolution of collaboration and contributors' centrality through releases. The approach can be largely enhanced by considering the multi-perspectives of contributor collaboration in the future. This work also contributes to the future investigation towards solving the organizational coupling in microservice projects, and furthermore, effectively maintaining the quality of microservice architecture.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and Grant No. 2245287, a grant from Red Hat Research https://research.redhat.com, a grant from the Ulla Tuominen Foundation (Finland), a grant from Climate Neutral Energy Systems and Society (CNESS), Tampere University, Finland, and a grant from the Academy of Finland (grant n. 349488 - MuFAno).

References

[1] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: a systematic mapping study," in *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress, 2018.

[2] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in *CLOSER 2019 - Proceedings of the 9th International Conference on Cloud Computing and Services Science*, 2019, pp. 153–164.

[3] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, vol. 60, no. 5, pp. 428–436, 2020.

[4] F. P. Brooks Jr, *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[5] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in *International Conference on Software Engineering*. IEEE, 2008.

[6] S. Newman, *Building microservices*. O'Reilly Media, Inc., 2021.

[7] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[8] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1651–1681, 2020.

[9] R. Damaševičius, "On the human, organizational, and technical aspects of software development and analysis," *Information Systems Development: Towards a Service Provision Society*, pp. 11–19, 09 2009.

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, p. 1053–1058, dec 1972.

[11] A. Jermakovics, A. Sillitti, and G. Succi, "Mining and visualizing developer networks from version control systems," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2011, pp. 24–31.

[12] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice architecture reconstruction and visualization techniques: A review," in *SOSE 2022*. IEEE, 2022, pp. 39–48.

[13] A. Mukherji, "The evolution of information systems: their impact on organizations and structures," *Management Decision*, 2002.

[14] M. E. Newman, "Modularity and community structure in networks," *The national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[15] U. Brandes, S. P. Borgatti, and L. C. Freeman, "Maintaining the duality of closeness and betweenness centrality," *Social Networks*, vol. 44, pp. 153–159, 2016.

[16] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Int. Conference on Software Engineering*, 2009.

[17] P. V. Singh, "The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 2, pp. 1–27, 2010.

[18] D. Surian, D. Lo, and E.-P. Lim, "Mining collaboration patterns from a large developer network," in *2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 269–273.

[19] L. Peng, M. Jianan, and L. Wenjun, "Structural stability of the evolving developer collaboration network in the oss community," *PLOS ONE*, vol. 17, no. 7, pp. 1–19, 07 2022.

[20] I. El Asri, N. Kerzazi, L. Benhiba, and M. Janati, "From periphery to core: a temporal analysis of github contributors' collaboration network," in *Working Conference on Virtual Enterprises*, 2017.

[21] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *International Symposium on Foundations of software engineering*, 2008.

[22] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 531–540.

[23] T. Gustafsson and A. Saltan, "Managing open-source microservices projects," in *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution*, 2019, pp. 30–36.

[24] E. Gaidels and M. Kirikova, "Service dependency graph analysis in microservice architecture," in *International Conference on Business Informatics Research*. Springer, 2020, pp. 128–139.

[25] I. U. P. Gamage and I. Perera, "Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach," in *2021 Moratuwa Engineering Research Conference (MERCon)*. IEEE, 2021, pp. 699–704.

[26] V. Bushong, D. Das, and T. Cernỳ, "Reconstructing the holistic architecture of microservice systems using static analysis." in *International Conference on Cloud Computing and Services Science (CLOSER)*, 2022.

[27] A. Al Maruf, A. Bakhtin, T. Cerny, and D. Taibi, "Using microservice telemetry data for system dynamic analysis," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 29–38.

[28] T. Hou, X. Yao, and D. Gong, "Community detection in software ecosystem by comprehensively evaluating developer cooperation intensity," *Information and Software Technology*, vol. 130, p. 106451, 2021.

[29] X. Shen, J. Du, D. Gong, and X. Yao, "Developer cooperation relationship and attribute similarity based community detection in software ecosystem," *Chinese Journal of Electronics*, vol. 32, no. 1, pp. 39–50, 2023.

[30] P. He, B. Li, and Y. Huang, "Applying centrality measures to the behavior analysis of developers in open source software community," in *2012 Second International Conference on Cloud and Green Computing*. IEEE, 2012, pp. 418–423.

[31] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proceedings of the national academy of sciences*, vol. 101, no. 9, pp. 2658–2663, 2004.

[32] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[33] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[34] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Generalized louvain method for community detection in large networks," in *Int. Conference on Intelligent Systems Design and Applications*, 2011.

[35] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the national academy of sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[36] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1978.

[37] K. Okamoto, W. Chen, and X.-Y. Li, "Ranking of closeness centrality for large-scale social networks," in *Frontiers in Algorithmics*, F. P. Preparata, X. Wu, and J. Yin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 186–195.

[38] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[39] J. Zhao, J. C. Lui, D. Towsley, and X. Guan, "Measuring and maximizing group closeness centrality over disk-resident graphs," in *Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 689–694.

[40] X. Li, Z. Zhang, and K. Stefanidis, "Mobile app evolution analysis based on user reviews," in *New Trends in Intelligent Software Methodologies, Tools and Techniques*. IOS Press, 2018, pp. 773–786.

[41] ——, "A data-driven approach for video game playability analysis based on players' reviews," *Information*, vol. 12, no. 3, p. 129, 2021.

[42] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Int. AAAI conference on web and social media*, vol. 3, no. 1, 2009, pp. 361–362.

[43] C. Carneiro Jr and T. Schmelmer, *Microservices from day one: build robust and scalable software from the start*. Springer, 2016.

[44] C. Richardson, "Dark energy, dark matter and the microservices patterns?!" https://qconsf.com/presentation/oct2022/dark-energy-dark-matter-and-microservices-patterns, 2022.

[45] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices anti-patterns: A taxonomy," *Microservices: Science and Engineering*, pp. 111–128, 2020.

[46] D. A. d'Aragona, L. Pascarella, A. Janes, V. Lenarduzzi, and D. Taibi, "Microservice logical coupling: A preliminary validation," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2023, pp. 81–85.

[47] D. A. Aragona, L. Pascarella, A. Janes, V. Lenarduzzi, R. Penaloza, and D. Taibi, "On the empirical evidence of microservice logical coupling. a registered report," *arXiv preprint arXiv:2306.02036*, 2023.

[48] J. Lewis and M. Fowlor, "Microservices," https://www.martinfowler.com/articles/microservices.html, 2014.

[49] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *Int. Conference on Software Maintenance and Evolution*, 2014.

[50] A. Nguyen Duc, D. S. Cruzes, C. Ayala, and R. Conradi, "Impact of stakeholder type and collaboration on issue resolution time in oss projects," in *Int. Conference on Open Source Systems*, 2011.