Microservice Logical Coupling: A Preliminary Validation

Dario Amoroso d'Aragona[1], Luca Pascarella[2], Andrea Janes[3], Valentina Lenarduzzi[4], Davide Taibi[1,4]
[1]Tampere University — [2]ETH Zurich — [3]FHV Vorarlberg University of Applied Sciences — [4]University of Oulu
dario.amorosodaragona@tuni.fi; lpascarella@ethz.ch; andrea.janes@fhv.at;
valentina.lenarduzzi@oulu.fi; davide.taibi@oulu.fi

***Abstract*—Coupling is one of the most frequently mentioned metric in software systems. However, to measure logical coupling between microservices, runtime information is needed or the availability of service-log files to analyze the calls between services is required. This work presents our emerging results, in which we propose a metric to statically calculate logical coupling between microservices based on commits to versioning systems. We performed an initial validation of the proposed metric with a dataset containing 145 open-source microservices projects. The results illustrate how logical coupling affects every system and increases overtime. However, we did not find a correlation between the number of commits or the number of developers and the introduction of logical coupling. In future, we investigate *why*, *how*, and *when* logical coupling is introduced in a system.**

***Index Terms*—Microservices, Logical Coupling, Empirical Software Engineering**

I. Introduction

Coupling and cohesion are two fundamental metrics in software engineering: coupling measures the degree of interdependence between modules (low is preferred) and the cohesion of a module indicates the extent to which its individual components are needed to perform its task (high is preferred) [1]. Low coupling and high cohesion are often mentioned in connection with maintainable code: code with low coupling means that one module can be modified without impacting other modules; a highly cohesive module means that that module has a single, well-defined purpose.

In this paper, we study how well "low coupling" is respected for microservice-based systems. Microservices should be loosely coupled since coupling increases maintenance effort and—particularly in the context of microservices—increases the need for synchronization between teams. Service teams should need to know as little as possible about other services. Highly coupled services might require synchronizing with other services before deploying new features, reducing the benefits of microservices. Coupling not only slows down the development process but also impacts other qualities e.g., performance since communication between services is slow compared to communication within a service. Along the same lines, microservices should be highly cohesive, keeping all the related logic in one service instead of splitting it into multiple ones [2].

Practitioners agree on the importance of low coupling between microservices referring to "independence between teams" [3], "independent deployment" [4] or "no need to synchronize between teams before deploying" [5].

However, besides agreeing that coupling is one of the most important attributes affecting the quality of designs, there is no standard way to measure it [1]. Various metrics were proposed in research (e.g., [6], [7], [8]) and low coupling is often mentioned by practitioners as a main benefit of microservices [9], [10] but practitioners are still often using their gut feeling to structure a system into services, causing an uncontrolled degree of coupling. Possible reasons for not adopting the proposed metrics are over-optimism, inadvertence, or the unavailability of data. As an example, [8], [11] assume the availability of log files describing calls to software components, used to extract usage processes, which are then used to propose microservices. Such logs are not always available or might require major changes in the system under development. Moreover, the measurement of coupling proposed in the literature is based on the static [12] or dynamic analysis [13] of source code. Coupling between teams such as the need to wait or to synchronize with other service teams before committing is not captured by such metrics. Previous works addressed the problem of coupling in monolithic systems thoroughly. Also, the concept of *logical* coupling was introduced, i.e., coupling, which is not based on a dependency in source code, but on the implicit dependency between artifacts that are often changed together. In particular, [14] extends the logical coupling metric, originally proposed by [15] to capture whether changes made in a predefined time window are logically coupled.

In this work, we propose the *Microservice Logical Coupling* (MLC) as extension of the logical coupling metric [14], [15] to microservices, and we validate it on 145 open-source (OSS) projects. The advantage of measuring microservice logical coupling is that teams can calculate it by accessing their source code repositories, without accessing other systems or instrumenting their code. In this paper, we present our emerging results, which will be further extended and empirically validated.

Paper structure: Section V describes the related work. Section II introduces existing logical coupling metrics and the proposed approach, while Section III reports the preliminary validation. Section IV discusses the results, Section VI illustrates the future plans while Section VII draws conclusions.

II. Microservice Logical Coupling (MLC)

Complex software systems, due to a myriad of reasons, are not always developed by zealous practitioners. Developers constantly deal with project constraints and pressure. Consequently, developers risk committing software changes outside their domain of experience, task assignment, or—of interest to us—microservice. To recognize tangled changes in versioning systems, logical coupling exploits the development history of a software system to find change patterns among code units that are modified together. D'Ambros et al. [14] leveraged a metric introduced by Robbes et al. [15] that captures whether changes made in a predefined time window are logically coupled. Similarly to [14], but with a focus on microservices, we study how to capture logical coupling in a microservice-based development process. In particular, we aim to investigate changes involving multiple microservices committed atomically in the same working unit. By looking at atomic working units, we can identify two levels of granularity pull requests and commits. Since good practices in software engineering [16] discourage developers from submitting pull requests of untangled changes, we focus on the commit level. To this purpose, we extended the definition of logical coupling [15] to microservices to identify the logical coupling at the Microservice level. In other words, we consider two microservices logically coupled if at least one file of both services are modified in the same commit.

Therefore, we defined **Microservice Logical Coupling (MLC)** as follows:

Let's m_1, m_2 two microservices and f_1, f_2 two files related, respectively, to m_1 and m_2, then:

$$m_1 \xLeftrightarrow{MLC} m_2 \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } f_1 \text{ and } f_2 \text{ changed in same commit} \\ 0 & \text{otherwise} \end{cases}$$

Due to a different goal—we aim at detecting the logical coupling in microservices—our definition differs from the one proposed by [15] in terms of boundaries. In particular, while [15], for a less structured workflow, needed to identify a time window in which two consecutive changes are considered coupled, we relied on the definition of microservice to identify the natural boundaries in which two changes are coupled. In other words, we consider two microservices logically coupled if at least change in a file of both services is committed in the same commit. Indeed, in our definition, we consider m_1 and m_2 as microservices instead of files/components.

Finally, [15] advised, we rely on a threshold to select relevant changes and filter out occasional coupled ones that lie in, for example, massive refactoring. Specifically, we consider logically coupled two microservices that have been committed together for several times greater than or equal to the threshold; the threshold chosen is five times as suggested by [15] [17].

III. MLC at work: A Preliminary Validation

To allow the replication of our study, we published the raw data in the replication package[1].

[1]https://figshare.com/articles/software/MS_Logical_Coupling/21953456

A. Study Design

Our goal is to understand the applicability of MLC and to understand how frequent MLC is in OSS projects.

We defined the following research questions (RQ):

> **RQ$_1$** *How high is the logical coupling between microservices in OSS projects?*

More specifically, in RQ$_1$, we conducted a preliminary analysis aimed at understanding the distribution of logical coupling within the selected projects. An improved understanding of this aspect allowed us to start assessing whether logical coupling could be actually treated as an interesting phenomenon that could negatively impact maintainability, agility, and code understandability. Answering RQ$_1$ allowed us to determine if the magnitude of the phenomenon is such to justify the following part of the work.

> **RQ$_2$** *Does the degree of logical coupling change over time in OSS projects, and if so, how much?*

Once we had characterized MLC, we focused on its evolution over time and the related magnitude. Answering RQ$_2$ allowed us to understand if there are significant changes between the beginning of a project, or the introduction of a new microservice, and the logical coupling. We expected a microservice to have a high logical coupling while they are introduced, because they are often introduced as clones of other projects, and because we expected that the owner of a microservice might commit to multiple services together at the beginning of the service life for simplicity reasons. While we were aware that this should not be performed, this practice was commonly reported in our previous industrial survey [9]. However, we suspected that developers might also introduce logical coupling later when they are in hurry.

Context. We selected a manually validated dataset [18] in which the authors developed, validated, and released a tool to recognize microservices in a given project automatically. In addition, the authors provided a list of 145 projects that have been manually validated as non-toy projects using microservices. The set includes projects whose source code is publicly available on GitHub[2] and is augmented with additional information, such as the microservice list and their relative paths.

We selected all projects having at least 10 commits, 5 contributors, and not being forks (to reduce the chance of mining duplicated code). The minimum number of commits and contributors aims at discarding non-representative outliers such as single-user projects. Moreover, we did not exclude any programming languages because our analysis did not rely on a particular subset of them. All the 145 projects fulfilled our criteria, and therefore we considered them all.

Data Collection. In order to analyze the selected projects, we created a Python script to collect development process metrics that systematically traverses all project commits. For each

[2]https://github.com

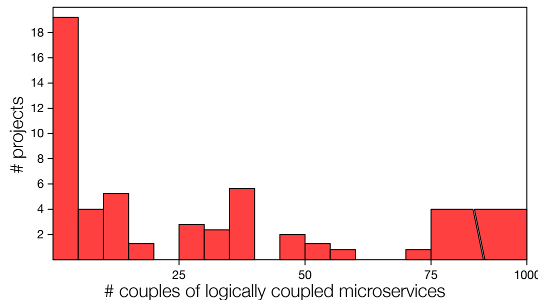


Fig. 1: Logically coupled microservices among projects (RQ_1)

commit, we collected the timestamp, the author identity, and the absolute change locations. The latter is used to associate a change to a microservice. In particular, we implemented a heuristic that matches if the path of the changed file falls into the project's microservice list. If yes, we updated the list of microservices changes in this commit by the given author.

Data Analysis. To answer RQ_1, we analyzed the Microservices logical coupling distribution over all the projects. We analyzed the number of microservices and the number of couples logically coupled in each project to have a clear idea about the magnitude of the problem in the projects analyzed.

Regarding RQ_2, we compared the first and the last commit to have an idea about how the logical coupling evolved in the project life. A hypothesis is that during the first month when developers start to commit, they introduce a high amount of logical coupling since they are creating new services as clones of existing ones; then, we analyzed the difference of microservices logically coupled in the first month and in the last commit; finally, we selected three interesting projects to compare the logical coupling introduction with the number of commits and developers per month.

B. Results

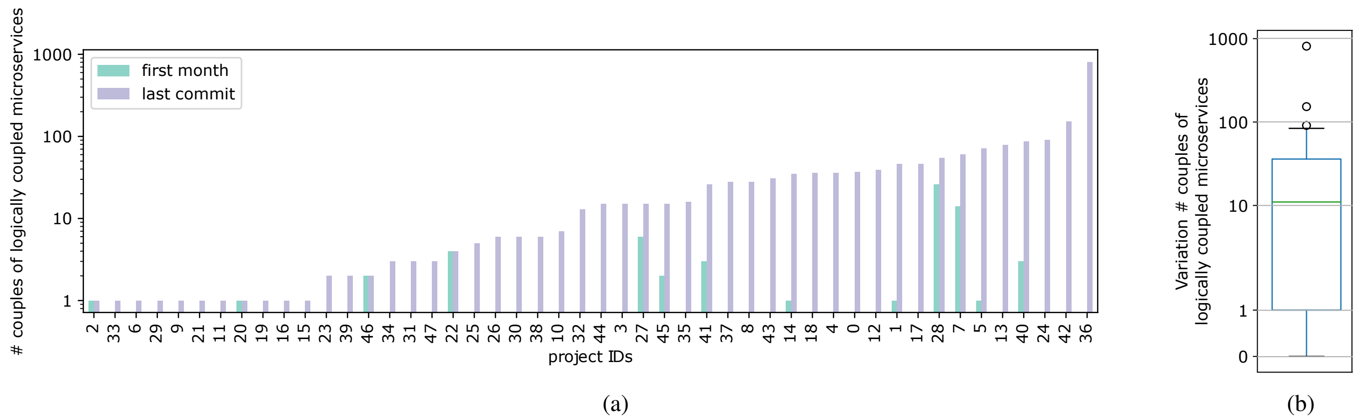
To answer RQ_1 we chose the number of couples of microservices logically coupled instead the number of microservices. For instance: let's A, B, C microservices and (A, B) and (B, C) couples logically coupled, the result is 3 microservices logically coupled and 2 couples of microservices logically coupled; now let's A, B, C microservices and (A, B),(B, C),(A, C) couples logically coupled, the result is 3 microservices logically coupled and 3 couples of microservices logically coupled, so the number of couples increased but the number of microservices (logically coupled) is still the same. The couples of microservices logically coupled indicate *how much coupling exists in a project*.

Figure 1 shows the number of microservices couples logically coupled among the projects. The $x - axis$ represents the number of couples logically coupled and the $y - axis$ the number of projects. In 48 projects (50%), out of 82 projects analyzed, there is a logical coupling between microservices. As it possible to see many projects has a number of couples logically coupled between one and five. There are 4 outliers that has more than 85 couples logically coupled.

To answer RQ_2 we, first of all, measured the difference between the first month of commits and the last commits (at the time of writing). The results are summarized in Figure 2a. As with RQ_1, we used the number of logically coupled pairs of microservices. In 48 projects the couples of microservices logically coupled increase during the evolution of the project, and in four projects the number of couples logically coupled is constant. Among the 45 projects that present logical coupling, in nine projects the logical coupling has been introduced during the first month of commits, instead in the remaining projects the logical coupling has been introduced after the first month during the life of the project. In Figure 2b we can observe the difference of microservices logical coupled in the first month and in the last commit over the projects. We used a symmetric log scale to handle the 0-values. The result is that in all but four projects the logical coupling increases, in some projects it increases by about tenfold, and in most projects, it increases by a value between 0 and 10. We selected three projects with different progressions of logical coupling and analyze the introduction of the logical coupling per month in relation to the number of commits and the number of developers per month (Figure 3). The introduction of a pair (e.g, *(A, B)*) of microservices logically coupled in a specific month means that in that month the two microservices (belonging to the couple, thus: *A, B*) has been committed for the fifth time in the same commit. Analyzing Figure 3, is it possible to see that the number of developers and the number of commits are not correlated with the number of introduced logically coupled couples. For example, in Figure 3a the number of couples logically coupled is around zero also when the number of commits is high. In the three projects example analyzed in Figure 3 we can observe three different situations: in Figure 3a the logical coupling is introduced constantly over the life of projects; in Figure 3b logical coupling is introduced in the first months, but after ten months is constantly zero (even if the number of commits increases); finally, Figure 3c shows a very different situation where the logical coupling is introduced after some months and then increases also when the number of commits decreases.

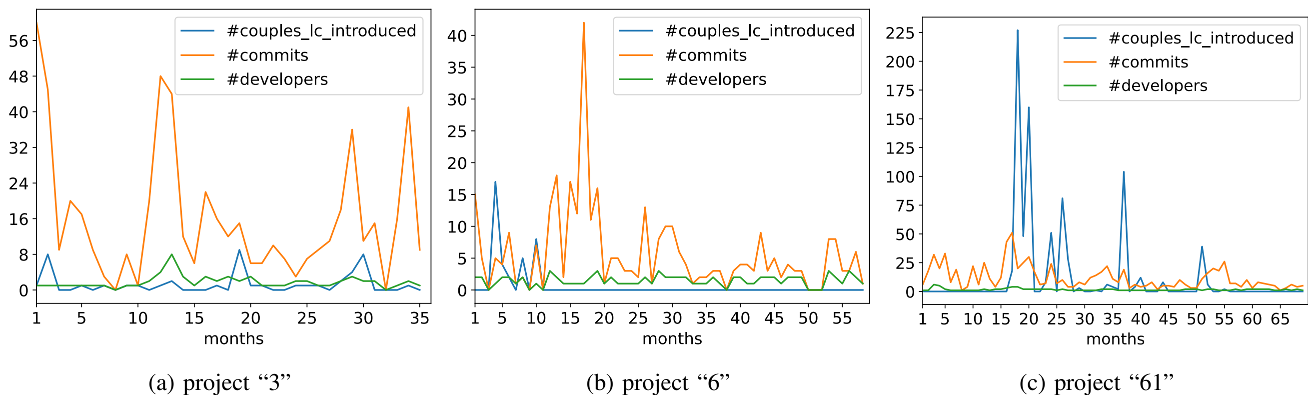
IV. Discussion

MLC exists and can be observed when developers commit multiple microservices at the same time. On one hand, the causes of MLC and its correlation between the numbers of commits or developers are not yet clear. On the other hand, it is clear that MLC often increases over time, but *how, when*, and *why* this occurs, needs more explanation. Our first hypothesis was that, when developers start developing a new project, particularly in OSS projects, the developers make a lot of *copy-and-paste* that produce in a short time a huge amount of MLC, but observing our results this hypothesis has been unconfirmed. This is also confirmed by [19] where code clones between services seem very high in the early life of microservices. It is surprising that sometimes MLC is introduced at the end of the life of the project, an explanation could be that at a certain point, the developers make a refactoring of the system and this leads to the introduction of MLC. Is interesting to note that, the number of commits has



(a) (b)

Fig. 2: a) Number of couples of logically coupled microservices between the first month and the last commit in log scale (RQ_2), b) distribution of the overall variation of the number of couples of logically coupled microservices between the first month and the last commit in symlog scale (RQ_2)



(a) project "3" (b) project "6" (c) project "61"

Fig. 3: Logical coupling per month in three projects with different progressions of logical coupling (RQ_2). The line *#couples_lc_introduced* describes the number of couples of logically coupled microservices that were introduced per month.

an implicit impact on MLC, simply because without commits it is not possible to introduce it. However, the results show that MLC is not related to the number of developers (Figure 3c).

Another interesting aspect that needs more clarification and work is understanding when we could consider not coupled anymore two microservices.

We are aware that our work is subject to threats to validity. First, although the selected projects might not represent the complete panorama of OSS microservices, we selected one of the most heterogeneous and recent datasets (for number of microservices, age of the projects, and number of developers). Second, industrial projects might consider coupling differently, especially because of the important message on low coupling reported by practitioner talks. Nonetheless, understanding the project's MLC could reveal inefficient teams and workflow organizations, allowing for an organization's renewal. Last, the MLC is only considering microservice co-changes in the same commit. Consequent changes due to the need for synchronizing services are not captured. A possible solution could be the introduction of a time window to consider co-changes. Another possible solution is to consider issues reported in the issue trackers, to verify if developers are requesting other services to synchronize with their change.

The MLC definition considers only the coupling between service and not its order of magnitude. As an example, if a large number of files for two microservices are modified in the same commit, the MLC will consider them as coupled as if only one file per microservice is modified. Moreover, we are aware that results can be biased by false-positive coupling scenarios. As an example, if two microservices that are not actually coupled are committed together more than three times, then we will consider them as Logically Coupled, obtaining a false positive coupling.

V. Related Work

A number of coupling metrics have been proposed for monolithic systems (see e.g., [1]), some of them have been highlighted for service-based systems, and especially for microservices. Bogner et al. [6] conducted a systematic literature review regarding maintenance metrics of microservices focusing on service-based systems instead of metrics designed for object-oriented systems. The results show that the majority of metrics explicitly designed for monolithic systems and for Service Oriented Architectures are also applicable in the

microservices context. Based on this work, which we consider to be the first one focusing on microservices, other similar works investigated the topic adding knowledge on top of their results. Apolinaro et al. [20] presented a theoretical use case proposing a roadmap to apply four metrics defined by [6]:

- Absolute Importance of the Service (AIS): number of consumers invoking at least one operation from a service
- Absolute Dependence of the Service (ADS): number of services on which the a service depends
- Service Coupling Factor (SCF) as density of a graph's connectivity. SCF=SC /N^2-N where SC is the sum of calls between services, and N is the total number of services.
- Average Number of Directly Connected Services(ADCS): the average of ADS metric of all services.

Taibi et al. [8] proposed four measures (coupling between microservices, number of classes per microservice, number of duplicated classes, and frequency of external calls) to be considered when decomposing an object-oriented monolithic system into microservices. Panichella et al. [21] proposed a structural coupling metric and validated it within 17 OSS microservice-based projects [22]. The metric is based on the inbound and outbound calls between services and measures the coupling of services at runtime.

VI. Future Plan

We plan to extend our work into these directions:
1. Expiration of coupling: We intend to explore if there are events that trigger a decoupling or a decay of the MLC. This would also alleviate the problem of false-positive coupling scenarios since over time services will not be coupled anymore.
2. Metric validity: This step is composed by two tasks:

a) *Number of considered commits*: We want to investigate if the used threshold of five commits respects the representational theory of measurement [1] in contemporary projects, i.e., if MLC corresponds to what we want to measure.

b) *Code changes:* We want to investigate how to include the actual code changes to understand the evolution of coupling.
3. Identification of coupling patterns: Fig. 3 shows that we observed different progressions of logical coupling over time in different project. We intend to study if we can observe common patterns in such progression.

VII. Conclusion

In this paper, we propose the metric MLC, to identify coupling between microservices based on their co-occurring changes. We performed a preliminary validation with 145 OSS projects developed with a microservices-based architecture, to demonstrate its applicability, and to understand how common MLC is in OSS projects. The results show that MLC affects all systems and increases over time. The introduction of MLC seems not to be correlated with the number of developers and the number of commits, thus the causes of the introduction of MLC need more in-depth investigations. The application of the here proposed metric will enable companies to early discover possible issues in their team allocation, or in their microservice decomposition.

Acknowledgement

This work was supported by a grant from the Ulla Tuominen Foundation (Finland), a grant from the Academy of Finland (grant n. 349488 - MuFAno).

References

[1] N. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. London, UK: International Thomson Computer Press, 1997.
[2] S. Newman, *Building Microservices*. O'Reilly Media, 2021.
[3] S. Hastie, "Kent Beck: Software Design is an Exercise in Human Relationships," 2022, https://www.infoq.com/news/2022/10/beck-design-human-relationships.
[4] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani, *Software Architecture: the Hard Parts: Modern Trade-Off Analyses for Distributed Architectures*. O'Reilly Media, 2021.
[5] C. Richardson, "Minimizing Design Time Coupling a Microservice Architecture," 2022, https://www.youtube.com/watch?v=EGOYRuuf2nQ.
[6] J. Bogner, S. Wagner, and A. Zimmermann, "Automatically Measuring the Maintainability of Service and Microservice-Based Systems: A Literature Review," in *Int. Workshop on Software Measurement*, 2017.
[7] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of Microservice Architectures: A Metric and Tool-Based Approach," in *Information Systems in the Big Data Era*, 2018, pp. 74–89.
[8] D. Taibi and K. Systä, "A Decomposition and Metric-Based Evaluation Framework for Microservices," in *CLOSER*, 2020, pp. 133–149.
[9] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.
[10] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A Systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.
[11] D. Gadler, M. Mairegger, A. Janes, and B. Russo, "Mining logs to model the use of a system," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 334–343.
[12] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice architecture reconstruction and visualization techniques: A review," in *International Conference on Service-Oriented System Engineering (SOSE)*, 2022, pp. 39–48.
[13] M. E. Gortney, P. E. Harris, T. Cerny, A. A. Maruf, M. Bures, D. Taibi, and P. Tisnovsky, "Visualizing microservice architecture in the dynamic perspective: A systematic mapping study," *IEEE Access*, vol. 10, pp. 119 999–120 012, 2022.
[14] M. D'Ambros, M. Lanza, and R. Robbes, "On the Relationship Between Change Coupling and Software Defects," in *Working Conference on Reverse Engineering*, 2009, pp. 135–144.
[15] R. Robbes, D. Pollet, and M. Lanza, "Logical coupling based on fine-grained change information," in *Working Conference on Reverse Engineering*, 2008, pp. 42–46.
[16] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *International Conference on Software Engineering (ICSE)*, 2016.
[17] D. Zhou, Y. Wu, L. Xiao, Y. Cai, X. Peng, J. Fan, L. Huang, and H. Chen, "Understanding evolutionary coupling by fine-grained co-change relationship analysis," in *International Conference on Program Comprehension (ICPC)*, 2019, pp. 271–282.
[18] L. Baresi, G. Quattrocchi, and D. A. Tamburri, "Microservice architecture practices and experience: a focused look on docker configuration files," 2022. [Online]. Available: https://arxiv.org/abs/2212.03107
[19] R. Mo, Y. Zhao, Q. Feng, and Z. Li, "The Existence and Co-Modifications of Code Clones within or across Microservices," in *Int. Symp. on Empirical Software Engineering and Measurement*, 2021.
[20] D. R. de Freitas Apolinário and B. B. N. de França, "Towards a method for monitoring the coupling evolution of microservice-based architectures," in *Brazilian Symposium on Software Components, Architectures, and Reuse*, 2020, p. 71–80.
[21] S. Panichella, M. I. Rahman, and D. Taibi, "Structural Coupling for Microservices," in *CLOSER*, 2020.
[22] M. Imranur, S. Panichella, and D. Taibi, "A curated Dataset of Microservices-Based Systems," in *CEUR-WS*, 09 2019.

View publication stats