Understanding the Causes of Microservice Logical Coupling: an Exploratory Study

Dario Amoroso d'Aragona
Tampere University
Tampere, Finland
dario.amorosodaragaona@tuni.fi

Xiaozhou Li
University of Oulu
Oulu, Finland
xiaozhou.li@oulu.fi

Andrea Janes
FHV Vorarlberg University of Applied Sciences
Dornbirn, Austria
andrea.janes@fhv.at

ABSTRACT

When implementing a software system, two crucial metrics come into play: the degree of interdependence between modules, known as coupling, and the extent to which the components within a module collaborate to fulfill its task, referred to as cohesion. This paper studies the phenomenon of *apparent coupling* and explores the possibilities to exclude it from measurement when studying coupling between microservices.

ACM Reference Format:
Dario Amoroso d'Aragona, Xiaozhou Li, and Andrea Janes. 2024. Understanding the Causes of Microservice Logical Coupling: an Exploratory Study. In *2024 International Workshop New Trends in Software Architecture (SATrends '24), April 14, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3643657.3643912

1 INTRODUCTION

When implementing a software system, two crucial metrics come into play: the degree of interdependence between modules, known as coupling, and the extent to which the components within a module collaborate to fulfill its task, referred to as cohesion [5]. The pursuit of low coupling and high cohesion is often associated with the creation of maintainable code [15]. Code with low coupling allows for modifications in one module without affecting others, and a highly cohesive module possesses a singular, well-defined purpose.

Practitioners unanimously recognize the importance of low coupling between microservices, emphasizing aspects like "independence between teams" [8], "independent deployment" [6], or "no need to synchronize between teams before deploying" [13].

Amoroso et al. [4] have investigated the existence of coupling in an Open Source Open Source project involving microservices are affected by this problem. In this context, logical coupling *is calculated analyzing each commit, counting the occurrences of co-changes of two microservices, when two microservices are committed together for more than a given threshold n*. If such *n* co-changes are identified, these two microservices are considered as coupled.

However, using the above definition of microservice coupling, one quickly realizes that there are some situations that cause frequent co-commits but are not conceptually couplings: large refactorings, changes in the architecture, or the presence the anti-pattern "shotgun surgery" can lead to apparent coupling in which multiple microservices are often changed in the same commit. Such an "apparent coupling" has different causes (as said before, e.g., refactoring or the presence of antipatterns) and should not be identified as coupling.

This seems even more true if we consider that generally, a developer works on more than one microservices at the same time [2]. Moreover, if apparent coupling is identified but its causes are yet unknown, a deeper investigation into the root causes and the context where apparent coupling manifests can help to understand the cause and thus propose a solution to adopt to mitigate the problem.

To clarify the distinction between *coupling* and *apparent coupling*, we need to define both terms in the context of this paper: we refer to *coupling* as the degree of interdependence of two software components or connection between them [1]. It measures how closely these components depend on each other, with low coupling indicating a weaker connection and high coupling indicating a strong and tight connection, which can affect the maintainability and flexibility of the software [15]. Unlike this, we use the term *apparent coupling* to describe the situation in which the adopted measurement technique is not accurate and identifies a *false positive*: two components that are not interdependent are identified as such.

Paper structure: Section 2 describes the goal and the research questions. Section 3 explains the necessary steps performed in shi work to answer the research questions. Section 4 reports the results of our analysis. Section 5 discusses the results, Section 7 illustrates the future plans, and Section 6 draws conclusions, and Section 8 describes the threats to validity of this work.

2 GOAL AND RESEARCH QUESTIONS

The goal of our work is to investigate the reason beyond the existence of coupling between microservices in an Open Source context.

Thus, we depict the following research questions: RQ_1: When and why are microservices committed together? and RQ_2: Which situations lead to *apparent* coupling?

RQ_1 wants to understand which type of operations (changes in the code) involves often more than two microservices. Starting from the results obtained in RQ_1, RQ_2 want to understand which could be possible situations to exclude when we are calculating dependency, in terms of Logical Coupling, between microservices.

3 METHODOLOGY

To answer the research questions, we will begin with RQ_1 and obtain a dataset of microservices that were committed together and further investigate about the timing and reason of such co-commits. Concretely, we will extract the commit message for each commit in

SATrends '24, April 14, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0560-1/24/04.
https://doi.org/10.1145/3643657.3643912

the the dataset analyzed in [4], identify the commits with more than one microservice, and apply natural language processing (NLP) to extract topics from those commit messages.

The assumption is that the topics of commits with more than one involved microservice help to identify the context in which *apparent coupling* occurs, i.e., in which microservices are committed together, even though there is no dependency among them.

Note that the used dataset [3] is the same used by the authors in [4], in order to have data to compare with.

To achieve the goal we pre-processed each commit message in order to prepare the text for algorithms. Due to the nature of the dataset and the unstructured nature of the commit message in the dataset, we prepared scripts to iteratively try different pre-processing strategies with each of the NLP algorithms selected.

Commit Message Clustering. This pre-processing strategies consist of tokenization, lemmatization/stemming, stop-words removal, cleaning (trimming, removing unnecessary spaces, removing punctuation, removing numbers, URLs, removing non-English words), but also of operations that are specific to the context of processing commit messages, in particular we defined the following strategies: S_1 extract *keywords* from each commit message (the keywords were selected taking only the VERB, the NOUN and the PRODUCT entity type in the commit messages); S_2 filter words based on position, thus discarding commits where—after non-english words removal—there is no verb in the first position (following the discovery of [10] that commit messages often begin with a verb followed by an direct object); S_3 substitute words to obtain more consistent commit message (in this case we only substitute "upgrade" with "update"); and S_4 deleting frequent domain-specific words (we noticed that some words, such as "docker", "docker-compose", were much more frequent on average due the fact that we are analyzing projects based on microservices; the fact that such words are present in the 90% of the commits—in our context—creates noise hindering to cluster the commit message correctly).

Note that depending on which strategy is used the final number of commits changes. In other words, commits can be discarded based on the strategy used. This means that we did not consider only the performance of clustering the processed commit message, but also how many commit messages were filtered out, privileging the combinations that give us the highest possible number of commit messages.

Furthermore, we defined different Natural Language Processing algorithms to try to process the commit message in the best way to obtain well-formed clusters.

In particular we tried the following combinations of algorithms: a) Word2Vec [12]; b) Word2Vec + Principal Component Analysis (PCA); c) Word2Vec + Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [11]; and d) BERTopic [7][1].

For each of these combinations, we tried, when applicable, to combine different preprocessing strategies, and for each combination, we used K-Means to cluster the commits message. For clustering, we created from 2 to 30 clusters and compared the results using Silhouette analysis [14].

[1]Please note that for BERTopic we did not use lemmatization/stemming because the machine learning algorithm has not been trained on lemmatized/stemmed words.

After obtaining well-formed clusters, we used GPT3.5 (model gpt-3.5-turbo-1106) to derive the topic for each cluster. We provided the top 10 most frequent words with their frequency in each cluster to derive the topic (following the approach in [9]).

Analyzing the type of commits responsible of Logical Coupling. For each pair of Logically Coupled microservices we selected the commits in which the two microservices were committed together and we extracted the clusters to which they belong. This allowed us to understand which are the most frequent type of commits that lead in Logical Coupling.

Recalculating Logical Coupling filtering the commits based on their type. In the context of determining when a developer is entitled to interact with more than one microservice within the same commits and when this should be avoided, we aim to refine the criteria for greater precision. To this extent, we analyzed the cluster obtained and we excluded the clusters that meet the following criteria: 1) Update of non-programming files (e.g., for example Docker files, configuration files.): the reason is that these types of changes often stem from an architectural change, where it is common to have an impact on all microservices [2]; and 2) Changes of versions of used third-party components: if a bug or a security issue is found in a shared library, is understandable (and desirable) that the affected component is updated as soon as possible everywhere. The same applies to other cases (e.g., deprecation of third party components, migration to another system, etc.).

Basing on this considerations we recalculated coupling, filtering out the commits belonging to clusters that depict such admissible situations. In this way we can understand if there are some changes in the results of the logistic coupling when more qualitative factors are considered. Due the subjective nature of this step, we want clarify that the goal of this paper is to explore how logical coupling changes if the type of the commit is taken into consideration. A more in-depth discussion is needed, also involving developers, to understand which categories of commits are to be excluded when calculating the Logical Coupling between microservices.

4 RESULTS

It turned out that the algorithm that performed best was Word2Vec + UMAP + K-means with the strategies S_1, S_3, S_4. Using these strategies allowed us to have the highest Silhouette value and to discard only the 30% of the commit messages. As depicted in figure 1, analyzing the silhouette score, 9 is the best number of clusters (silhouette score 0.92).

The topics that were identified from the resulting clusters are: Feature Implementation, Version Update and Project Maintenance, Bug Fixing and Issue Resolution, Dependency Management and Code Maintenance, Project Configuration and Dependency Bumping, Bug Fixes and Dependency Updates, Feature Integration, Code Quality and Maintenance, and Project Updates and Additions.

Analyzing the commits where microservices have been committed together, we see how the majority of the commits belong to the cluster *Feature Implementation* and *Version Update and Project Maintenance* as shown in figure 2.

Based on the above considerations, we propose a heuristic, based on the exclusion of common topics identified within commit messages. As an example, we explore the exclusion of version updates.

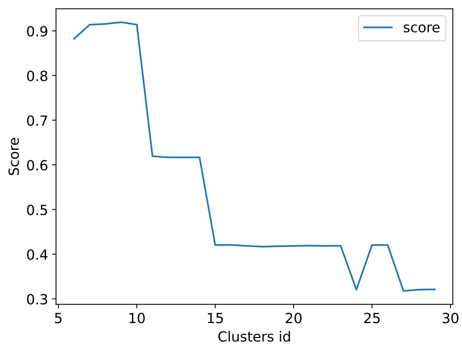

Figure 1: Silhouette score for Word2Vec+UMAP+K-means

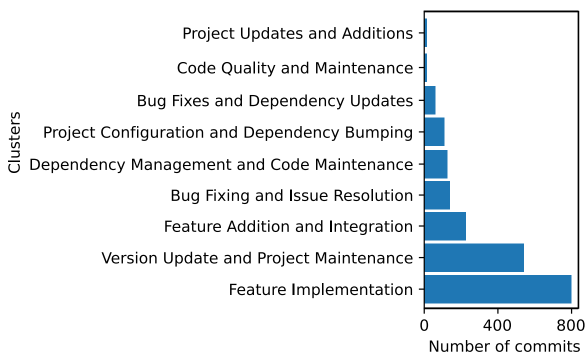

Figure 2: Most frequent type of commits leading to Logical Coupling

Version updates are one of those situations where changing more than one microservice at the same time can be considered acceptable: e.g., consider the update of a common library, an change of the used database technology or a used API. We could consider these situations as not relevant if we want to understand if there are couplings between microservices that hinder the regular maintenance of these services. The logical coupling between microservices tries to highlight whether there are dependencies between microservices that are not created by explicit design decisions. The decision to change the version of a tool (e.g. because a vulnerability was found in the currently used version) is indeed a design decision. Based on these considerations, we recalculated the logical coupling and discarded the commits belonging to the *version update and project maintenance* cluster.

The result is depicted in Figure 3. The figure shows the number of couples of microservices logically coupled among the projects analyzed, in particular the boxplots shown the number of couples of microservices logically coupled when all the commits are considered (*all_commits*) and when are considered all the commits but the commits belonging to the *Version Update and Project Maintenance* cluster (*filtered_commits*).

Note that the total number of commits (leading in Logical Coupling) in the dataset is 2254, we filtered out 598 commits (36, 1%)

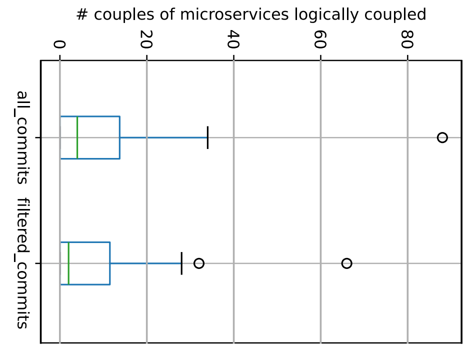

Figure 3: Comparison of couples number of microservices logically coupled considering all commits and only the selected commit types

belonging to *Version Update and Project Maintenance* cluster, obtaining 1565 commits.

5 DISCUSSION

Analyzing the type of commits responsible of Logical Coupling (RQ_1), the results show how the majority of commits where logical coupling was detected based on co-committed microservices belongs to the *Feature Implementation* cluster. A possible explanation is that logical coupling really exists in this project and that the implementation of the new feature requires changing more than one microservice. Another possibility is that developers did not follow the practice to commit each task separately and that these commits represent more than one change is that more features are added in different microservices and then all these changes are committed together. To better understand which of these possibilities applies, we need to perform a more detailed analysis, such as manually labeling the commit messages and involving the developers through a questionnaire or interviews.

When recalculating Logical Coupling filtering the commits based on their type (RQ_2), the first assumption we had was that filtering out commits causes a decrease in terms of Logical Coupling. However, even if the amount of commits we filtered is considerable, the impact on logical coupling seems to be quite low. We assume that the reason is that commits contribute to logical coupling width different weights (e.g., if one commit changes three microservices, we measure three different couples (ms_1,ms_2),(ms_1,ms_3),(ms_2,ms_3)). Taking this aspect into consideration, we can derive how not only is the number of commits is important but also the number of pairs of logically coupled microservices committed together.

If we look at the absolute numbers of pairs of logically coupled microservices, looking at all commits yields 628 pairs of logically coupled microservices among all projects, in fact filtering the commits yielded 427 pairs of logically coupled microservices. Thus, about 600 ($\tilde{3}6$%) commits are responsible for 201 (30%) pairs of logically coupled microservices, compared to the 1656 ($\tilde{6}4$%) commits responsible for 70% of the logical coupling (628 pairs of logically coupled microservices). This situation is shown in Figure 4.

We can therefore see that the excluded commits have an impact on the number of logically coupled pairs of microservices and

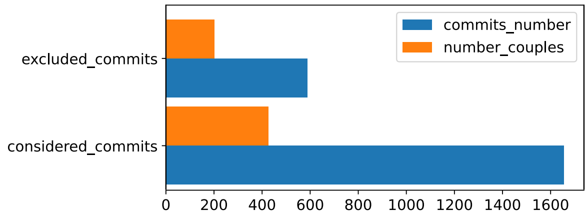

Figure 4: Incidence of the number of commits on the number of couples logically coupled.

therefore it is important to further investigate what kind of commits (and therefore changes in the code) need to be discarded when we calculate the dependency between microservices in terms of logical coupling.

6 CONCLUSION

When analyzing what kind of commits cause the logical coupling in the open source microservices context, we found that most of the commits belong to the "Feature Implementation" operations. There may be several reasons for this, but a deeper investigation involving the developers is required to understand the reason.

The second most frequent type of operation responsible for logical coupling in our work is *Version Update*. We think that this is one of the possible types of changes that can be excluded when calculating the dependencies between microservices, especially when we calculate the logical coupling, which by its definition is not interested in the dependencies resulting from design decisions.

By excluding this cluster from the analyzed commits, we were able to determine that the impact of a type of operation on the logical coupling depends not only on the number of commits, but also on the number of microservices that are committed together in this type of operation. However, the excluded cluster was responsible for 1/3 of the logical coupling found in the analyzed projects, which confirms that it is necessary to investigate which type of operation must be excluded when we investigate coupling (especially logical coupling) between microservices. Understanding the causes of coupling between microservices is fundamental to propose possible solutions to mitigate the problem.

7 FUTURE WORK

Clustering commit messages is a difficult task. In this work, we propose a solution, but in most cases, especially for small projects, the commit messages do not follow any rules, which makes their categorization difficult. As future work, we want to try to use other techniques to preprocess the commit messages, such as TF-IDF vectorization, to alleviate the problem of words present in most commit messages causing noise in the data. To obtain more consistent and structured commit messages, we will try to generate the commit messages using these techniques and then perform clustering on the generated commit messages. Other aspects that need to be investigated in the future are the reason why feature implementation is the most responsible operation for logical coupling and what kind of operations need to be discarded in this kind of analysis. For the two aspects mentioned above, we would like to involve the developers to better understand the issues they face during development, which can lead to situations responsible for logical coupling and which are in fact the situation where changing more than one microservice at the same time is not a problem.

8 THREATS TO VALIDITY

In terms of external validity, we are aware to the fact that we analyzed only a subset of microservices-based projects, and we know that may not represent the state of the art. However, the goal of this work was to understand if a qualitative investigation on the type of commits can lead in different results from the ones obtained by the authors in [4]. In addition, we used the only dataset currently available containing validated open-source microservices-based projects [3]. In terms of internal validity, we are aware that a different text preprocessing can have a significance impact, resulting in different clusters and thus in different results. However, we tried different combination of preprocessing strategies in order to respect as much as possible the nature of text, without changing the context ans thus the meaning.

REFERENCES

[1] 2010. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)* (2010).
[2] Dario Amoroso d'Aragona, Xiaozhou Li, Tomas Cerny, Andrea Janes, Valentina Lenarduzzi, and Davide Taibi. 2023. One Microservice per Developer: Is This the Trend in OSS?. In *Service-Oriented and Cloud Computing*, George A. Papadopoulos, Florian Rademacher, and Jacopo Soldani (Eds.). Springer Nature Switzerland, Cham.
[3] Luciano Baresi, Giovanni Quattrocchi, and Damian Andrew Tamburri. 2022. Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files.
[4] Dario Amoroso d'Aragona, Luca Pascarella, Andrea Janes, Valentina Lenarduzzi, and Davide Taibi. 2023. Microservice Logical Coupling: A Preliminary Validation. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*.
[5] Norman Fenton and Shari Lawrence Pfleeger. 1997. *Software Metrics: A Rigorous and Practical Approach* (2nd ed.). International Thomson Computer Press, London, UK.
[6] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani. 2021. *Software Architecture: the Hard Parts: Modern Trade-Off Analyses for Distributed Architectures.* O'Reilly Media.
[7] Maarten Grootendorst. 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794* (2022).
[8] Shane Hastie. 2022. Kent Beck: Software Design is an Exercise in Human Relationships.
[9] Andrea Janes, Xiaozhou Li, and Valentina Lenarduzzi. 2023. Open tracing tools: Overview and critical comparison. *Journal of Systems and Software* 204 (2023).
[10] Siyuan Jiang and Collin McMillan. 2017. Towards Automatic Generation of Short Summaries of Commits. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*.
[11] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426 [stat.ML]
[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]
[13] Chris Richardson. 2022. Minimizing Design Time Coupling a Microservice Architecture.
[14] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987).
[15] W. P. Stevens, G. J. Myers, and L. L. Constantine. 1974. Structured design. *IBM Systems Journal* 13, 2 (1974).