The Journal of Systems & Software 200 (2023) 111670

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

On measuring coupling between microservices

Chenxing Zhong [a,b,*], He Zhang [a,b,**], Chao Li [a,b], Huang Huang [c], Daniel Feitosa [d]

[a] *Software Institute, Nanjing University, Nanjing, China*
[b] *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China*
[c] *State Grid Nanjing Power Supply Company, Nanjing, China*
[d] *Faculty of Science and Engineering, University of Groningen, Groningen, The Netherlands*

ARTICLE INFO

Article history:
Received 15 August 2022
Received in revised form 26 December 2022
Accepted 27 February 2023
Available online 1 March 2023

Keywords:
Software quality
Microservices architecture
Coupling measure
Change impact

ABSTRACT

In software quality management, the selection strategy for proper metrics varies depending on the application scenarios and measurement objectives. MicroService Architecture (MSA), despite being commonly employed nowadays, still cannot be reliably measured and compared if the microservices in a system are independent. Software managers and architects need to understand whether their microservices are "decoupled enough", if not, which ones are over-coupled, and by how much. In this paper, we contribute a novel set of metrics – Microservice Coupling Index (MCI) – derived from the relative measurement theory. Instead of measuring coupling evidence with simple counts, we measure how dependent and coupled the microservices are relative to the possible couplings between them. We measured the MCI metrics for 15 open source projects that involve 113 distinct microservices. Empirical investigation confirmed that MCIs differ quite significantly from existing coupling measures and that they are more discriminative than existing ones for separating high and low degrees of microservice couplings and thus more useful in comparing design alternatives. A series of experimental studies were conducted, showing that the larger the MCIs, the less likely the bugs and changes can be localized and separated, and the less likely that the individual microservices in a system can be independently developed and evolved.

1. Introduction

Measuring software systems in terms of properly selected metrics is an integral step for software quality improvement (Galin, 2004). The strategy for selecting "appropriate" metrics varies differently, depending on the specific application scenarios and measurement objectives of the software project management activities. A significant scenario of software quality measurement is to quantify how the system specifies quality requirements with respect to relevant quality factors (Fu and Cai, 2019), such as maintainability. In addition, good software design is one of the most important activities in the system development lifecycle (Almugrin et al., 2016) and is crucial for the successful implementation of software systems that meet such quality requirements. Software metrics have been used for long to compare and choose design alternatives (Mo et al., 2016), monitor or predict architecture degradation (Kirbas et al., 2014; Perepletchikov et al., 2007), and indicate refactoring opportunities in software evolution.

Microservices architecture (MSA) (Jamshidi et al., 2018), is described as an alternative to monolithic architecture, and an approach for developing a single application as a suite of small and independent services, each running in its own process and communicating with lightweight mechanisms, such as RESTful API (Lewis and Fowler, 2014). As one of the most predominant architecture styles nowadays, the design of MSA also needs to be gauged, monitored, and improved throughout the whole software development lifecycle.

It has been widely acknowledged that *cohesion* and *coupling* are key internal quality attributes of a software design. Generally, a good software design is characterized by the principle of high cohesion and low coupling (Czibula et al., 2019), thereby associated with low maintenance effort, while the main symptom for badly-structured systems is considered as the violation of the rule *"Put together what belong together"*. The internal structure of a software changes frequently during its evolution. In the context of MSA, a typical changing scenario is service refactoring and decomposition, where microservices are frequently split and merged to find the most appropriate service granularity. For ensuring a facile and efficient microservice evolution, it is essential for software managers and architects to accurately estimate and

* Corresponding author at: Software Institute, Nanjing University, Nanjing, China.
** Corresponding author.
E-mail addresses: zhongcx@smail.nju.edu.cn (C. Zhong), hezhang@nju.edu.cn (H. Zhang), mf21320079@smail.nju.edu.cn (C. Li), sgcc.huang.huang@gmail.com (H. Huang), d.feitosa@rug.nl (D. Feitosa).

https://doi.org/10.1016/j.jss.2023.111670
0164-1212/© 2023 Elsevier Inc. All rights reserved.

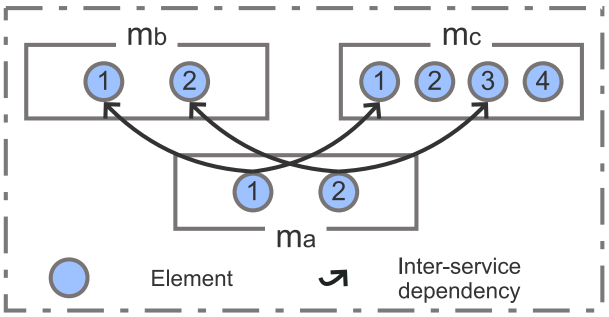


Fig. 1. An illustrative example for showing the limitation of using absolute metrics in comparing microservice coupling.

compare the quality of refactoring alternatives (Bogner et al., 2019).

Coupling is considered an important indicator that negatively impacts software maintainability. Independent modules are easier to fix, change, and extend during their maintenance and evolution. In the MSA community, the coupling property has been considered as one of the minimum quality requirements to be evaluated during the design of microservices (Cojocaru et al., 2019; Waseem et al., 2021). Such a property is essential because it enables the individual microservices in a system to be independently developed, deployed, tested, and scaled. Therefore, many MSA researchers and practitioners consider the independent microservices as an essential trait of desirable architecture design (Li et al., 2019; Carvalho et al., 2020; Auer et al., 2021; Di Francesco et al., 2019).

Nevertheless, few coupling measurement in the SE literature has been proposed for describing the degree to which the microservices in a software system are related. Instead, some coupling metrics that were proposed for object-oriented or service-oriented paradigms have been adapted and applied now in MSA, such as Sellami et al. (2022), Carvalho et al. (2020), Mazlami et al. (2017) and Li et al. (2019). Most of these metrics are generally based on simple counts of coupling "evidence" in a system, such as the number of dependencies (e.g., static calls) between microservices and the number of software elements (e.g., classes) involved in the dependencies. These absolute scales can be effective tools for reflecting the magnitude of coupling between microservices. However, they fail to provide a comparative gauge on to what extent the microservices in a system are coupled (and less independent), and fail to help maintainers identify and refactor the least desirable dependencies which could possibly undermine the independence of individual microservices.

Take Fig. 1 as an example. Suppose there are three microservices m_a, m_b, and m_c in a system with their structural relationships depicted as in the figure. Using absolute metrics, we might find out that the coupling between m_a and m_b, and between m_a and m_c are equal, as there are two dependencies from m_a to m_b and from m_a and m_c (as well as two elements involved in the dependencies). However, given that the size of microservice m_b is only half that of m_c, we know that the coupling between microservices m_a and m_b is more severe and undesirable. After all, every time microservice m_b changes, microservice m_a might have to be changed as well, while this is less likely between microservice m_a and microservice m_c. In short, it is not fair enough to argue that two microservices are more detrimentally coupled than other two microservices using absolute coupling value.

The objective of this paper is to address the limitations of absolute coupling measurements among microservices. Nevertheless, we admit that coupling measurement is a complex task that covers diverse "connection" forms between software entities (Fregnan et al., 2019). Over the years, different coupling metrics have flourished in the literature, including *structural* (e.g., Mo et al., 2016; Almugrin et al., 2016; Czibula et al., 2019), *dynamic* (e.g., Fu and Cai, 2019), *semantic* (e.g., Czibula et al., 2019), and *logical* coupling (e.g., Alali et al., 2013). Additionally, in the MSA context, the style of inter-service communication (i.e. synchronous, asynchronous) can also affect how we model and measure microservices coupling. In this paper, we limit our research scope to structural coupling measurement since it is the most often used (Czibula et al., 2019) and the most fundamental in assessing coupling degree (Bavota et al., 2013). In particular, we focus on the coupling caused by synchronous inter-service dependencies (Engel et al., 2018), because synchronous communication is one of the most commonly used in service collaboration (Newman, 2021).

This paper contributes a novel suite of microservice-level coupling metrics: Microservice Coupling Index (MCI). Instead of measuring the absolute coupling evidence, we measure how dependent and coupled the microservices are relative to the possible couplings between them. Based on the relative measurement theory (Allen and Yen, 2001), these metrics are assumed to be, very likely, capable to reflect and compare microservices coupling degrees which cannot be captured using only the absolute measures. Second, inspired by Martin's seminal work in object-oriented packaging, this study on microservice coupling further explore the indicators of responsibility and dependence of individual microservices in a system. Finally, to better capture the dependencies across microservices, in MCIs we distinguish between interface classes that are intended to provide services for the outer world (e.g, other microservices), and inner classes for internal use within the microservice.

We measured the MCI metrics for 15 open sour projects from GitHub and GitLab that involve overall 113 distinct microservices and 10241 code files. Empirical investigation using statistical and principal component analysis was conducted, showing that the relative measures differ quite significantly from existing coupling measures and that they can capture multiple aspects of the interdependencies across microservices. In addition, we emphasized that the MCI metrics are more discriminative than classical coupling measures for measuring and separating high and low degrees of microservice couplings, and thus more useful in comparing and choosing design alternatives. To evaluate if microservices with higher MCI are less independent in terms of change impacts, we contributed a suite of change impact measures that quantify the affected scope and frequency in a microservice due to the changes of another microservice. A series of experimental studies were designed to collect these measures by simulating the revision history of 8 long-term projects. Our results show that the larger the MCIs, the less likely the bugs and changes can be localized and separated, and the less likely that the individual microservices can be independently developed and evolved.

This paper is organized as follows. We start by reviewing in Section 2 the existing measurements from the literature, especially those adapted to MSA. Our proposal for the relative microservice coupling measures MCIs is introduced in Section 3. In Section 4, the comparison between MCIs and other existing coupling measures are highlighted, especially their discriminative power in separating high- and low-level of couplings, and their capabilities in measuring microservices' independence. Section 5 discusses the practical implications of MCIs and the threats to validity that may affect the results of this study. Finally, Section 6 concludes our paper and outlines some further research directions.

2. Background and related work

Numerous metrics are well-known for measuring software code quality, such as Halstead metrics (Halstead, 1977), McCabe Cyclomatic complexity (McCabe, 1976), and Lines of Code. Moreover, metrics such as CK (Chidamber and Kemerer, 1994), LK (Lorenz and Kidd, 1994), and MOOD (Harrison et al., 1998) have been proposed for measuring object-oriented software. Utilizing code quality metrics to aid software maintenance and evolution has also been studied for a long-term. Oman and Hagemeister (Oman and Hagemeister, 1994) proposed one of the most classic approaches, namely Maintainability Index, which is a composite model for predicting maintainability based on multiple metrics, such as the average lines of code and cyclomatic complexity per module. Recently, Papamichail and Symeonidis (Papamichail and Symeonidis, 2020) built a generic identification framework of non-maintainable code components, by training Support Vector Machines classifiers based on the complexity, cohesion, coupling, and inheritance properties residing in software repositories.

Using only objects or classes to model a software system is insufficient for producing robust, stable, and maintainable designs, especially when the size of the system is large. Accordingly, a grouping of interrelated objects or classes has been acknowledged as a better unit of organization in software design than a single object or class. The design quality of such groupings, also known as modules (Mo et al., 2016), packages (Almugrin et al., 2016), components (Fu and Cai, 2019), subsystems (Rumbaugh et al., 1991), etc, is often measured using architecture metrics. Some example studies of architecture-level metrics are as follows. Mo et al. (2016) proposed an architecture maintainability metric that measures how the software is decoupled into small and independently replaceable modules. Almugrin et al. (2016) presented a package-level coupling metric and used it as indicators to predict maintainability and testability. Fu and Cai (2019) developed a suite of metrics for run-time communication structure complexity between distinct components and explored their relationships to six quality factors including maintainability.

Outside the classic metrics, some measurements (e.g., Kramer and Kaindl, 2004; Kazemi et al., 2011) have been developed for specific domains (Fregnan et al., 2019), such as Service-Oriented Architecture (SOA). Among the SOA quality researchers, Alahmari et al. (2011) proposed three metrics that cover service operation complexity, cohesion and coupling for indicating the optimal granularity of a service and their impacts on reusability, flexibility and maintainability. Athanasopoulos et al. (2014) provided three interface-level cohesion metrics for enabling the assessment of service cohesion only with the specification of a service. The authors validated these metrics (related to message, domain, and conversation-level similarity) using 22 real-world web services from Amazon and Yahoo. Perepletchikov et al. (2007) focused on quantifying the structural coupling of design artifacts in service-oriented systems, and using the results as early predictors of quality characteristics of maintainability. The coupling properties considered include extra-service and inter-service, incoming and outgoing coupling, which were evaluated using a controlled experiment with 10 participants and a number of corrective and perfective maintenance activities.

Compared to other architecture metrics, only a few studies have focused on microservices-level metrics, despite the popularity of MSA nowadays. Al-Debagy and Martinek (2020) proposed three metrics to measure the granularity, cohesion, and complexity of individual microservices. Santos and Silva (2020) introduced a complexity metric to estimate the effort of redesigning a functionality implemented in monolith into distributed microservices. Jin et al. (2019) evaluated microservice-level cohesion by adapting existing metrics from the SOA field. The quantification of modularization "quality" from monolithic applications to microservices has been studied in Kalia et al. (2021), Kalia et al. (2021), and Jin et al. (2019).

The idea behind MSA is to design a single large and complex application as a suite of small and autonomous microservices (Assunção et al., 2021). It means that the microservices should be designed in a way where a single service implements specific cohesive business capabilities within its design, and the interaction between services should be kept at a low level. Different from the SOA field where developers strive for the reusability of integrations and some level of component sharing, MSA is an area in which creating reusable components may result in inter-service dependencies that reduce agility and resilience. As a result, MSA developers generally prefer to reusing code and duplicating data for improving the decoupling and independence level of individual microservices (Valderas et al., 2020). However, inter-service dependencies are required when microservices need to cooperate with each other. The need of microservices cooperation indicates that some form of dependencies are reasonable while some are not. In terms of quantifying the dependencies among microservices and determining the reasonable level of coupling, MSA developers are more likely to be sensitive towards the measurement results. Therefore, the assessment of microservice-level coupling quality, by quantitatively describing microservices dependencies needs to be more accurate and rigorous. Many researchers (e.g., the authors from Li et al., 2019; Jin et al., 2019) have also noticed the specific needs in evaluating the coupling characteristics of microservices design.

Nevertheless, most metrics employed for microservice-level coupling were adapted from traditional disciplines, such as object-oriented and service-oriented paradigms. For example, Li et al. (2019) used the classic Afferent Coupling (Ca) and Efferent Coupling (Ce) metrics to quantify microservice coupling by calculating the number of classes in other services that depend upon the service itself, and the number of classes in other services that a service depends upon, respectively. Two more examples from the SOA, Absolute Importance of the Service (AIS) and Absolute Dependence of the Service (ADS), measure microservice coupling by counting the number of other services that rely on a service, and the number of other services that a service depends on, respectively (Vera-Rivera et al., 2021). The problem is that these metrics are only based on simple (absolute) count of coupling "evidence" in a system and have not considered the size of microservices. Working with our industrial partners, we observed cases where a single microservice contained very large number of code files. In these cases, even though the inter-dependencies among services appeared to be high, they may not be tightly coupled. In other cases, we observed that even though the number of files was not large in a microservice, it has a few dependents on others. In such cases, a microservice may experience maintenance issues, despite its low dependencies.

Different from these existing metrics, our MCI is the only metric suite that measures how *dependent* the microservices are relative to the possible dependencies between them, based on the relative measurement theory (Allen and Yen, 2001). Our objective is not for maintainability prediction, because of the fact that maintainability will be affected by many factors (Mo et al., 2016), such as the size of a microservice. Instead, we aim at providing a comparative gauge on to what extent the microservices in a system are coupled and less independent, and a timely indicator of architecture refactoring for decoupling them.

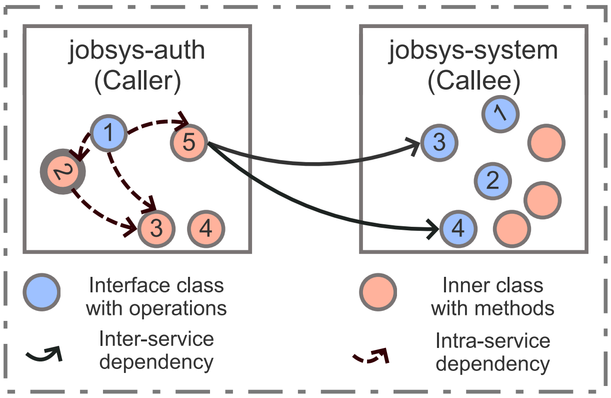


Fig. 2. An example of microservice-pair dependency graph.

3. Proposed microservice coupling measures

In this section, we introduce the formal definitions and rationale of Microservice Coupling Index (MCI). Based on the relative measurement theory, the MCI aims at measuring how dependent and coupled the microservices are relative to the possible couplings between them. Thus, we first introduce the constituents of a single microservice and the possible associations of inter-service that may undermine the independence of individual microservices, as the theoretical basis. Next, we explain and introduce the proposed relative microservice coupling measures, including the indicators for responsibility and dependence of a single microservice in a system. Please note that in this section, we use real examples from an open source microservice recruitment project to illustrate how MCI can manifest microservice couplings.

3.1. Basic definitions

A microservices-based software system is composed of a set of standalone microservices, each having multiple constituent elements inside. To measure the coupling between microservices, we need to accurately describe to what extent their constituents are associated, thereby a formal model of a single microservice is required. Fig. 2 illustrates an excerpted design view of two microservices from the example project: "jobsys-auth" and "jobsys-system". The "jobsys-auth" service is responsible for authentication (for all requests), while the "jobsys-system" service offers help for system login and registry. As present in Fig. 2, each of these microservices consists of two types of fundamental elements: (*i*) classes that are intended to expose public services to the outer world (e.g., other microservices, external clients), and (*ii*) classes for internal use within the microservice. In this paper, we distinguish these two types of elements mainly for better capturing the associations across microservices. In particular, we employ the concept of *interface* classes and *inner* classes (Sellami et al., 2022) to distinguish between the two aforementioned classes, respectively. In the meantime, we use the concept of *entity* to embody both of these classes within a microservice. Formally:

Definition 1 (*Microservice*)**.** A microservice M_i can be considered as a set of entities Ent_i, including interface and inner classes defined in the microservice. If $\mathbb{I}nt_i = \{int_1, int_2, \dots, int_n\}$, $\mathbb{I}nn_i = \{inn_1, inn_2, \dots, inn_p\}$, and $\mathbb{E}nt_i = \{ent_1, ent_2, \dots, ent_s\}$, where $int_k \in \mathbb{I}nt_i$ represents an interface published by microservice M_i, $inn_q \in \mathbb{I}nn_i$ denotes an inner class implemented in microservice M_i, and $ent_t \in \mathbb{E}nt_i$ denotes either a published interface or an inner class defined within the microservice, then it follows that $M_i = \mathbb{E}nt_i = \mathbb{I}nt_i \cup \mathbb{I}nn_i$.

When examined from a lower level, an interface class within a microservice is further constituted by a series of operations. Each operation is a fine-grained API that corresponds to an inter-service calling method and can be directly consumed by other microservices (Jin et al., 2018). The formal definition of an interface follows:

Definition 2 (*Interface Class*)**.**An interface class of a microservice, int_i, is characterized by a series of operations, which is denoted by $int_i = \mathbb{O}peration_i$. Specifically, $\mathbb{O}peration_i = \{op_1, op_2, \dots, op_n\}$, where each operation $op_k \in \mathbb{O}peration_i$ corresponds to an inter-service calling method that can be invoked by other microservices using inter-service communication techniques, e.g., RESTful APIs.

On the other hand, when developed using object-oriented design, an inner class is typically defined by multiple attributes and methods. Nevertheless, this study focuses on measuring the extent of microservice couplings due to their structural interdependencies, in which the attributes of a class often do not directly participate (discussed later in Section 3.2). We thus denote an inner class as a series of methods that can directly invoke other methods, e.g., public methods of other classes within the same microservice, exposed operations from other microservices, etc. Formally:

Definition 3 (*Inner Class*)**.** An inner class in a microservice, inn_i, can be considered as a set of methods declared in the class itself, denoted by $inn_i = Method_i$. To be specific, $Method_i = \{md_1, md_2, \dots, md_p\}$, where $md_q \in Method_i$ represents a method defined in inner class inn_i that could invoke or be invoked by other possible methods.

As aforementioned, we employ the *entity* concept to generally represent the constituents elements of a microservice: *interface* classes and *inner* classes. To be specific:

Definition 4 (*Entity*)**.** An entity is a generalization of both an interface class and an inner class. That is, an entity can be either an interface class or an inner class, and every interface class or inner class is also an entity. The entities within a microservice can be considered as the union of the interface class set and inner class set inside the microservice, formally $\mathbb{E}nt_i = \mathbb{I}nt_i \cup \mathbb{I}nn_i$.

Dependency analysis is one of the most studied core techniques for quantifying the coupling relations among software components (Fregnan et al., 2019). The more dependencies between two components, the greater the degree of coupling between the two components. To measure the coupling degree between microservices, in the following we thus discuss the interdependence among microservices.

In addition, the dependencies within a microservice can sometimes lead to a ripple effect, i.e. when a change to one part of a system has consequences to other parts of the same system (Arvanitou et al., 2015). If there is ripple between two microservices, the dependencies within a microservice may increase this effect by propagating it to more classes inside a microservice (for more details, see Section 3.2). Therefore, the intra-service dependencies are also discussed as follows.

Dependence analysis across microservices. *Given two microservices M_i and M_j in a microservices-based system, microservice M_i depends on microservice M_j, which corresponds to a "Caller-Callee" relationship between them (as Fig. 2), if and only if:*

- There exists an entity *ent* in microservice M_i that depends on a public interface class *int* of microservice M_j, which further requires:

- There exists a method *md* in entity *ent* that directly invokes at least one published operation in the interface class *int*.

To that end, the operation to be invoked has to be publicly published in interface class *int*, and the method *md* in entity *ent* has to explicitly subscribe the operation. This is determined by the inter-service communication mechanisms in MSA, which requires that microservices access each other's resources only through the published interfaces. In addition, due to the fact that each microservice is running in its own process, the cases where entities from distinct microservices directly call each other can be avoided (Lewis and Fowler, 2014).

Dependence analysis within a microservice. *Given several entities (i.e. interface and inner classes) and their methods in a microservice, there exists the following kinds of dependencies:*

- A method md_i directly depends on another method md_j, if md_i calls md_j, where these methods can be from the same entity or different entities in the microservice.
- A method md_i indirectly depends on another method md_j, if md_i depends on another method md_k calling md_j, no matter these methods are from the same entity or different entities in the microservice.
- An entity ent_i directly depends on another entity ent_j, if a method in ent_i directly depends on one method of ent_j.
- An entity ent_i indirectly depends on another entity ent_j, if a method in ent_i indirectly depends on one method of ent_j and no method in ent_i directly depends on any method of ent_j.

Please note that we did not discuss the indirect dependencies between microservices, because, as mentioned earlier, for supporting the loose-coupling microservices principle, the inter-service dependencies occur through service interfaces, which are generally expected to be designed to be stable and robust over evolution and to prepare for future changes (Bogner et al., 2019; Zhong et al., 2022). As a result, the cases where the interface changes in one microservice, lead to the contract changes of a second microservice that further propagate to a third microservice could be very few. Considering that the indirect dependencies between microservices are likely to have much less influences than direct ones, for the time being of this article they are not considered.

3.2. Microservice coupling index

This section first introduces the coupling measure between two microservices, then extends the definition to indicate the coupling degree between one and multiple microservices. Specifically, inspired by Martin's work (Martin, 2003), we further define two additional coupling measures that estimate how much a microservice in a software system influences (responsibility) and is influenced by (dependence) other microservices from the system.

Suppose there are two microservices in a project, and each of them have multiple entities inside, including interface and inner classes. If none of the entities in a microservice invoke the published interfaces of another, it means that these two microservices are likely to be independent and loose-coupled, and it is less likely that the developers of these services need to expend effort communicating with each other for possible functionality changes (Bogner et al., 2019). On the other extreme, if all the entities of a microservice, say Caller, depend on the interfaces of another microservice, say Callee, and all the interfaces of Callee microservice are depended upon by the Caller, then, it seems that these two services are over-coupled, and it is highly likely that every time the interfaces of microservice Callee are to be modified, the developers have to communicate to resolve possible conflicts among these two services. It is obvious that the coupling degree between these two microservices, in the second case, is much higher than that in the first case.

However, these are not all the cases that make differences in microservice couplings. Even if all the entities of Caller microservice rely on all the interfaces provided by the Callee, as in the second case, to what extent the entities of the Caller depend on those interfaces of Callee, e.g., directly or indirectly, can still make the coupling between them vary in degree. To accurately quantify the coupling level between two microservices, we compute a ranked list of the potential couplings based on the following assumptions:

- For a Caller microservice, the Callee microservices with more interfaces consumed by the Caller, are more likely to impose their influences on it, and thus more coupled with it.
- For a Callee microservice, the Caller microservices with higher number of entities that rely on the interfaces of the Callee are more probably to be affected by its changes, and thus more coupled with it.
- In a Caller microservice with multiple entities that depend on microservice Callee, the entities further away from the Callee are relatively less vulnerable to changes made to the Callee and thus less coupled by it.

The first assumption tells us that for all the Callee microservices that a microservice (Caller) depends on, the Callee microservices having more interfaces being depended on by the Caller, are more likely to propagate their changes to the Caller and thus have larger impacts on it. That is, the coupling degree between the Caller microservice and such Callee microservices are more detrimental.

The second assumption considers the number of possibly affected entities due to inter-service dependencies, which implies that the degree of microservice Caller impacted by microservice Callee increases as more entities within the Caller depend on the Callee, because more entities might have to change due to the changes made to the Callee. The more the possibly affected entities in Caller, the higher the microservice Caller is coupled with the Callee.

Finally, the third assumption is about the distance on the dependencies between microservices. Specifically, the possibility of an entity in microservice Caller that can be impacted by changes to microservice Callee decreases as the distance from the entity to the depended interfaces in Callee becomes farther. Similar arguments about dependency distance (Li et al., 2013) were observed in the change impact area. Take a special example, for two entities i and j in the Caller microservice that both depend on the same Callee microservice. Suppose entity i relies on the Callee microservice directly, and entity j indirectly, e.g., entity j depends on the Callee through its dependence on entity i. We can take the 5th and 1st entity in the "jobsys-auth" service in Fig. 2 as an reference. Then we can state that entity i is more likely to be impacted by the Callee, since its distance to the interfaces of Callee is closer than that of entity j. This assumption considers the relations among the possibly impacted entities in Caller microservice and the interfaces of the Callee microservice that are depended upon by them, rather than the arbitrary entities and interfaces among microservices.

According to these three assumptions, we thus define a metric, *Microservice Coupling Index*, to quantify the coupling strength between two services. Specifically, the coupling index in the direction from microservice M_i to microservice M_j, which is denoted by $MCI(i, j)$, measures the likelihood that the entities in microservise M_i are impacted by microservise M_j. A higher MCI value indicates that the corresponding Caller microservice is more tightly coupled to the Callee. The $MCI(i,j)$ metric is defined as follow:

$$\frac{|Dep_I(i,j)|}{|Int_j|} \times \left[\frac{|Dep_E(i,j)|}{|Ent_i|} + \frac{|Dep_E(i,j)|}{\sum_{ent \in Dep_E(i,j)} min(dist(ent, Int_j))}\right] \quad (1)$$

In this formula, $Dep_I(i,j)$ is the set of multiple depended interfaces in microservice M_j that are reachable from the entities in microservice M_i. A higher number of depended interfaces means that the Callee microservice is more coupled with the Caller. $Dep_E(i,j)$ is the set of multiple dependent entities in microservice M_i that are possibly impacted by changes made to microservice M_j. The more the dependent entities, the higher coupled the Caller microservice is with the Callee. The $min(dist(ent, Int_j))$ is the least number of edges needing to traverse downward (opposite the dependency direction) from any of the depended interfaces in microservice M_j to the dependent entity ent in microservice M_i. We note that the $min(dist(ent, Int_j))$ for entity ent is not zero only when ent is downward reachable from interfaces in microservice M_j (i.e. Int_j), meaning that the actually accumulated entities are equal to the possibly impacted entities in microservice M_i by microservice M_j, namely $Dep_E(i,j)$. This formula is clearly in line with the aforementioned three assumptions: $Dep_I(i,j)$ represents the first assumption, $Dep_E(i,j)$ expresses the second assumption, and $\sum_{ent \in Dep_E(i,j)} min(dist(ent, Int_j))$ is related to the third assumption.

In the meantime, the number of interfaces in Callee microservice, denoted by $|Int_j|$, and the total number of entities in Caller, expressed as $|Ent_i|$, are employed in this formula, following the relative measurement theory (Allen and Yen, 2001). This indicates that the more the interfaces in a microservice, the more likely it will influence more entities in other microservices, and hence lower MCI; the larger a microservice, the more opportunities for it to invoke interfaces of other microservices, and thus lower MCI.

We note that the MCI values are always smaller than (or equal to) 2, and it equals to 2, if and only if, all the interfaces of microservice M_j influence all the entities in microservice M_i, and, all these influences are directly imposed, meaning that the $min(dist(ent, Int_j))$ of all entities in the formula are minimal to 1. To make a complete definition, if there is no dependency from microservice M_i to M_j, the corresponding MCI value is 0. Formally:

$$MCI(i,j) = 0, if \ Dep_I(i,j) = \varnothing \ or \ Dep_E(i,j) = \varnothing \quad (2)$$

Finally, we note that the MCI values of $MCI(i,j)$ and $MCI(j,i)$ are not equal according to the definition. It means that in this study, we distinguish the direction of coupling between two microservices in terms of how much two microservices depend upon each other. Such differences can be made because of the fact that the extent one service depends on (influences) another service is not equal to that dependences (influences) from the opposite direction. For example, in Fig. 2, the coupling value from service "jobsys-system" to "jobsys-auth" is 0, smaller than that from service "jobsys-auth" to "jobsys-system", which is $\frac{2}{4} \times (\frac{3}{5} + \frac{3}{1+2+3}) = \frac{11}{20}$, because the "jobsys-system" service does not depend on the "jobsys-auth" service. In the following, we define two additional measures to estimate the coupling level between one and multiple microservices.

Afferent Microservice Coupling Index (aMCI) : This coupling metric is designed by measuring how much a microservice can influence other microservices in a system. A higher degree of a microservice's influences on other microservices, means that it has more dependencies from others, and thus the more afferent couplings with others. Almugrin et al. (2016) stated that the more dependencies from others on a software module, the more responsibilities it has in the system. Thus, the higher the aMCI, the more responsibilities the microservice has for others, and thus a higher prestige of it in the context of the whole system. A microservice would have a high aMCI if the summation of its influences on other microservices is high. Considering the MCI metric introduced in Formulae (1), we propose below a measure to quantify the overall influences microservice M_i can impose on others.

$$\frac{\left|\bigcup_{j\neq i} Dep_I(j,i)\right|}{|Int_i|} \times \left[\frac{|Dep_{EA}(i)|}{\left|\bigcup_{j\neq i} Ent_j\right|} + \frac{|Dep_{EA}(i)|}{\sum_{ent \in Dep_{EA}(i)} min(dist(ent, Int_i))}\right] \quad (3)$$

Where $Dep_{EA}(i) = \bigcup_{j\neq i} Dep_E(j,i)$, denoting the set of overall entities in other microservices that depend on M_i. If a microservice influences none of the other microservices, its aMCI value is 0. This indicates a maximally irresponsible microservice, meaning that it is very likely that this microservice can be changed independently and with little effects on other microservices in the system. While $aMCI = 2$ indicates a maximally responsible microservice, implying that this microservice is very important in the system.

Efferent Microservice Coupling Index (eMCI) : This coupling metric aims to quantify how much a microservice can be influenced by others in the system. The higher the degree a microservice can be influenced, means that the more dependencies it has on others, and thus the more efferent couplings with others. Since software dependencies are the means for transferring changes from one artifact to another (Kretsou et al., 2021), changes applied to many of its depended microservices may propagate to it and cause it to change too. Therefore, this measure reflects the vulnerability of a microservice to the changes of other microservices in the whole system. A microservice will have high eMCI if the summation of those influences that are imposed on it is high. Based on the defined MCI metric, the overall influences that are imposed on microservice M_i by others in the system can be calculated as follows.

$$\frac{\left|\bigcup_{j\neq i} Dep_I(i,j)\right|}{\left|\bigcup_{j\neq i} Int_j\right|} \times \left[\frac{|Dep_{EE}(i)|}{|Ent_i|} + \frac{|Dep_{EE}(i)|}{\sum_{ent \in Dep_{EE}(i)} min\left(dist\left(ent, \bigcup_{j\neq i} Int_j\right)\right)}\right] \quad (4)$$

Where $Dep_{EE}(i) = \bigcup_{j\neq i} Dep_E(i,j)$, indicating the set of overall entities in microservice M_i that depend on other microservices. If a microservice does not rely on any others, its eMCI value is 0. This indicates a maximally independent microservice, meaning that it has little risk to be changed due to the effects from other microservices. In addition, $eMCI = 2$ indicates a maximally dependent microservice, suggesting that it has many efferent couplings and is very unstable in the context of the whole system.

3.3. Tool support

We have built a program to calculate *Microservice Coupling Index*, which is currently programmed using Java language. Our MCI program takes the structural information within and across microservices as its inputs. The structural information of each microservice, is contained in a single XLS file, including the interfaces it declared, inner classes it implemented, and the dependencies between them. The inter-dependencies across microservices in a project are stored with another single XLS file. Given these inputs, our tool (accessible on GitHub[1]) calculates the *Microservice Coupling Index* values between every pair of microservices, as well as the aMCI and eMCI in the microservice level.

1 https://github.com/lemonheroine/MicroserviceMeasure

Please note that the tool's input is actually a microservice architecture obtained from reverse engineering. That is, our coupling measurement tool does not support reverse engineering. This design is more portable because architectures derived from various reverse engineering techniques can be used as input. Nevertheless, considering the popularity of the Spring Cloud framework, we also built another tool called "MicroParser" to automatically recover the microservice architecture developed with Java language and OpenFeign technology (see Section 4.1). "MicroParser" is also available on the GitHub.

4. Evaluation

The objective of this evaluation is to investigate how the MCI metrics behave when measuring microservice couplings, which is three-fold. First, we aim at studying the relations between MCIs and the existing coupling metrics in MSA, in order to demonstrate the practicality of measuring microservice-level coupling using MCIs. Second, we aim at assessing the discriminative power of the MCI metrics, in order to demonstrate their superiority over existing metrics in separating high-coupled and low-coupled microservices. Third, we aim at investigating the implications of microservice-level coupling to the independence of individual services in terms of change impacts, in order to demonstrate the practical usefulness of measuring MCIs for understanding, comparing, and even improving microservices' independence. This section focuses on clarifying the guiding research questions, and then presenting the objects of study, data analysis and results of each of the questions.

RQ1: Are the MCI measurements correlated with but significantly different from the existing absolute coupling values when assessing microservices?

– Positive answers to this question imply that the MCI metrics are practical in measuring microservice-level coupling, because the relative coupling level calculated by them are reasonable for their associations with the absolute measurements of coupling. More importantly, the proposed metrics differ quite significantly from existing measurements because they capture the ratio scales uniquely.

RQ2: Is the discriminative power of MCI metrics in separating high and low coupled microservices higher than that of existing absolute measurements?

– Positive results of this question indicate that the MCI metrics can distinguish the microservices with the same absolute coupling measurements but with different relative coupling levels, so that a manager can reliably employ them to determine which microservices in a system are relatively more coupled, and thus possibly need to be decoupled.

RQ3: Does higher MCI values of two microservices mean that these microservices are less independent than other two microservices with lower MCI values?

– A positive answer to this question implies that it is possible to quantitatively compare the extent of independence, as well as the induced change impacts among different microservices or design alternatives for the same microservices. Thus, if we measure the MCI metrics for a large number of microservices, a maintainer could consult this dataset and determine whether the most coupled microservices in the system is in the "healthy" range or need to be improved by refactoring.

Since several coupling metrics have been adapted and applied for measuring microservice coupling, we would like to know whether the MCI metrics are more reliable than the existing ones. Therefore, we will answer the three questions using MCIs and these metrics comparatively. Specifically, for measuring the afferent coupling of a microservice (aMCI), we used the Ca and AIS metrics (see Section 2). To estimate a microservice's efferent coupling, we employed the Ce and ADS metrics as baselines.

To the best of our knowledge, there are few metrics for measuring the coupling degree between two specific microservices. Thus, we extended the definitions of Ca, Ce, AIS, and ADS and derived the following ones for evaluation purposes.

- Afferent Coupling between Two services (CaT): number of classes in another microservice that depend upon (interface) classes within the microservice itself.
- Efferent Coupling between Two services (CeT): number of (interface) classes in another microservice that the classes in the microservice depend upon.
- Absolute Coupling between Two services (ACT): whether a given microservice depends upon another microservice.

It can be seen that the definitions of CaT and CeT are adapted from that of Ca and Ce, respectively. For the coupling metrics AIS and ADS, only one metric namely ACT is adapted, because the basic definitions of these two metrics are the same, meaning that adapting two metrics here may lead to information redundancy. Therefore, in the following sections, the MCI for coupling between two microservices will be compared with these three metrics.

4.1. Subjects selection and preparation

To minimize the possible bias caused by project selection, we randomly searched our subjects in GitHub and GitLab. At first, the projects that are marked as "microservice" or its synonyms (e.g., "micro service"), or described as microservices-based software systems were considered as the candidates. With this criterion, we first derived dozens of code repositories, but most of them are microservices frameworks or development kits with little business functionalities in their modules (microservices). To better analyze the dependency relationships between microservices and measure their coupling, we further chose the candidates by filtering out those projects with little business functions (the second criterion). Third, driven by the popularity of the Spring Cloud framework, most of the microservice projects we found were implemented in Java and used the Spring Cloud OpenFeign technology for inter-service communication. In order to consistently automate the extraction of inter-service dependencies, we focus on Java and Spring Cloud projects in this paper. Finally, we collected 15 microservice projects with various domains, sizes and ages, as shown in Table 1.

To calculate the microservice coupling measures in the subject projects, we chose the latest version of each project (in early June 2022), downloaded its source code, and then reversed-engineered it. In our study, the recovery of microservices architecture is majorly about identifying microservices and their interdependencies (as discussed in Section 3.3). To this end, we first extracted microservices based on the documentation descriptions of a project online, including architecture diagrams and textual descriptions. If a project does not describe its microservices, we then manually analyzed its package structure for identifying microservices, e.g., those modules with YAML configuration files. That is, we considered a microservice as a component that can be independently deployed (according to the system configuration files), which is consistent with the microservices characteristics (Lewis

Table 1
Studied projects.

Subjects	#Services	History length	#Files	LOC	#Commits	Description
Recruit[a]	8	3 months	273	5k	43	Intelligent recruitment system
Enarx[b]	7	1 month	386	1k	34	Human resource management system
Cangjingge[c]	7	21 months	114	2k	66	Online novel platform
Mall-swarm[d]	7	50 months	780	66k	439	Microservice mall system
Xavius[e]	5	30 months	120	2k	44	A solution to add/remove services
Madao-service[f]	6	39 months	776	14k	1431	Blog platform
Mall-cloud[g]	13	15 months	1010	106k	71	Microservice mall system
RuoYi[h]	7	25 months	676	18k	730	Rights management system
Light-reading[i]	4	19 months	192	4k	80	Reading application
EPR[j]	4	56 months	84	1k	81	Product management platform
Zscat[k]	13	57 months	1509	49k	117	Online shopping mall
Snowy[l]	10	18 months	2755	136k	194	Rights management system
Job-offers[m]	8	2 months	660	13k	123	Internet recruitment platform
Rpush[n]	6	8 months	380	10k	110	Message push system
Grocery[o]	8	3 months	526	11k	50	Microservice mall system

[a] https://github.com/stalary/microservice-recruit.
[b] https://github.com/paulohvescovi/springcloud-course.
[c] https://github.com/lgasyou/cangjingge.
[d] https://github.com/macrozheng/mall-swarm.
[e] https://github.com/AlaaMezian/spring-boot-microservices.
[f] https://github.com/GuoGuang/madao_service.
[g] https://github.com/mtcarpenter/mall-cloud-alibaba.
[h] https://github.com/yangzongzhuan/RuoYi-Cloud.
[i] https://github.com/Zealon159/light-reading-cloud.
[j] https://github.com/bharathmit/SpringCloud-MSA.
[k] https://gitee.com/infowangxin/sc.
[l] https://gitee.com/xiaonuobase/snowy-cloud.
[m] https://gitee.com/su-aiya/susu713.
[n] https://gitee.com/shuangmulin/rpush.
[o] https://gitee.com/zx_l/grocery-micro-service.

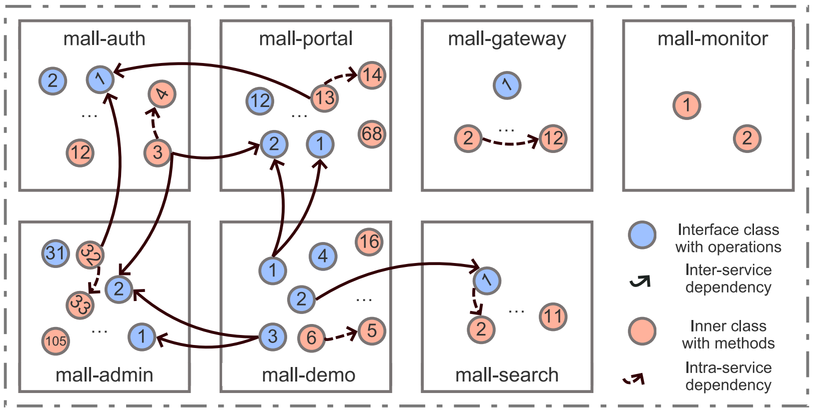


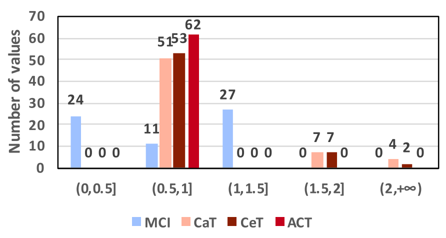
Fig. 3. Architecture of one of the analyzed systems.

and Fowler, 2014). As a result, we derived overall 113 distinct services from these projects (as Table 1).

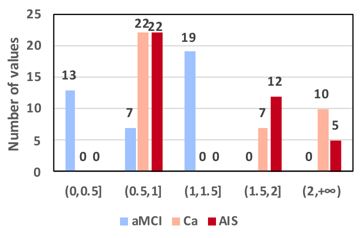
On the other hand, the inter-services dependencies in our study were collected via static code analysis from the source code. This can be done by detecting and leveraging the specific code fragments applied for inter-service communications, e.g., the "@RequestMapping" annotation for HTTP request. Driven by the popular Spring Cloud OpenFeign technique in all our subjects, we created an recovery tool called "MicroParser" that can automatically extract microservices dependencies by leveraging the specification of OpenFeign. Our extraction of microservices dependencies is theoretically in line with the state-of-the-art techniques for microservices reconstruction used by Zdun et al. (2022), and our researchers checked the extraction results. The code and recovered data set are provided as an open source artifact on GitHub[1] to enable reproducibility of this study. In particular, Fig. 3 presents an illustrative example of the architecture recovered from the systems we analyzed. Please note that for the convenience of presentation, Fig. 3 only shows a part of microservice entities and dependencies. The complete architecture information can be found in our replication data set.

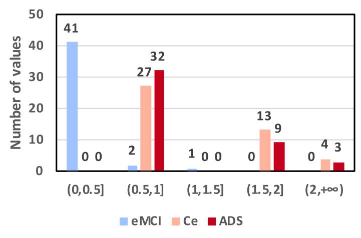
Our reverse-engineering process outputs, for each project, a set of XLS files containing all the file-level dependency information within and across microservices. Given these XLS files, we used our measurement tool (introduced in Section 3.3) to calculate the MCIs and other microservice coupling metrics for each project.

Table 2 reports the statistics of the microservice coupling values obtained from these projects. This table shows that the minimum coupling values calculated from all the open source projects are 0. In the meantime, more than 50% of the analyzed units have coupling values equal to 0 (zero). Specifically,



(a) Couplings of microservice-pairs





(b) Afferent couplings

(c) Efferent couplings

Fig. 4. Histogram distribution of non-zero microservice coupling values (Normalized).

Table 2
Metric summary for all subject projects.

Statistics		Min	Max	Avg	Median
Couplings of microservice-pairs (N = 842)	MCI	0.000	1.220	0.047	0.000
	CaT	0.000	3.000	0.091	0.000
	CeT	0.000	3.000	0.087	0.000
	ACT	0.000	1.000	0.074	0.000
Afferent coupling (N = 113)	aMCI	0.000	1.110	0.232	0.000
	Ca	0.000	7.000	0.681	0.000
	AIS	0.000	4.000	0.549	0.000
Efferent coupling (N = 113)	eMCI	0.000	1.150	0.077	0.000
	Ce	0.000	6.000	0.646	0.000
	ADS	0.000	6.000	0.549	0.000

the proportion of microservice-pair coupling, afferent coupling, and efferent coupling with non-zero value is 7.4% (62), 34.5% (39), and 38.9% (44) respectively, meaning that most of the microservices in these subjects have no inter-dependencies between them. This could be attributed to two reasons. The first reason is the principle of independent services that is advocated in the MSA community. Second, MSA is an architecture paradigm that requires many development costs, and as such most of these (non-profit) open source projects are of small and intermediate scale and with low complexity. In addition, it shows that the worst MCI is 1.22, rather than the value of 2 as we theoretically explained. This also indicates that these subjects comply to a certain extent with the service independence principle as advocated. Fig. 4 further shows the data distribution of all the non-zero microservice coupling values. It can be observed that while the values of other coupling metrics are majorly concentrated between 0.5 and 1, the MCI values are relatively evenly distributed in multiple intervals from 0 to 1.5. The other characteristics of these metrics are discussed later.

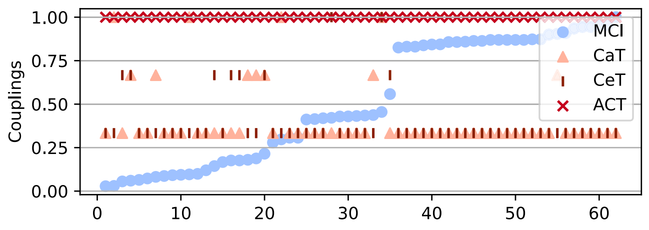
4.2. RQ1: Relation to other coupling measures

This analysis is conducted to answer the first research question, namely whether there is an association but a statistically significant difference between the MCI metrics and the existing coupling measures in literature. If so, it indicates that MCI contributes a uniquely significant/informative perspective in measuring microservice coupling that the existing metrics cannot subsume. Next, we first compare the values of MCIs and other measurements to illustrate their "associations" and "differences", and then use statistical analysis to test the significance of these relationships.

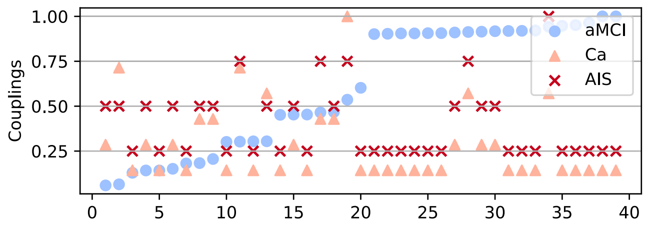
From the reporting in Section 4.1, we found that over 90% of these microservice-pairs have a coupling value of zero, meaning that they do not explicitly depend on each other. Moreover, all the employed metrics consistently assigned zero to such cases, suggesting that all these metrics agree that if there is no inter-dependency between two microservices, it is very likely that they are not coupled. On the other hand, by observing the remaining 62 pairs of microservices with MCI > 0 (approx. 8%), we see that the developed MCI metric shows a different model from that of CaT, CeT, and ACT. Fig. 5(a) depicts the 62 pairs of microservices, sorted ascending by MCI values (e.g., the far left presents the microservice-pair with the lowest MCI value, and its other coupling measurements, i.e., CaT, CeT, and ACT). From this figure, we can notice a broader range of MCI values, exemplifying the representative power of the metric for distinguishing to what extent the entities of a Caller microservice depend upon the interfaces from Callee.

Furthermore, the proportion of non-zero afferent couplings and efferent couplings is 34.5% and 38.9%, respectively. That is, in over 60% of the cases where the microservices are not afferently or efferently coupled with others in a system, both our MCIs and other metrics assign zeros to them. In Fig. 5(b) and 5(c), we focus on the microservices with afferent and efferent coupling greater than zero, respectively. From these, we see that the MCI differ from other metrics.

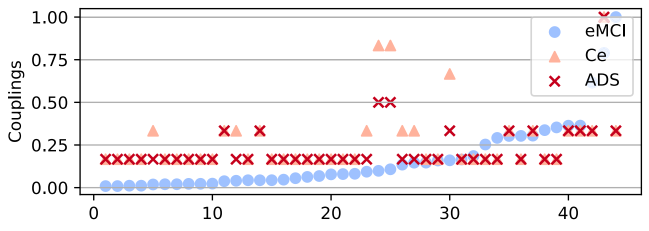
To answer this question more rigorously, we used the Spearman's rank correlation coefficient (Spearman, 1961) – a measure of the strength of correlation between two sets of variables – to test the association between these coupling metrics. While the Pearson correlation focuses on the linear relationship between two variables, the Spearman correlation is a measure of the monotone association (Hauke and Kossowski, 2011) and does not require the data to be normally distributed. In Spearman's rank correlation, the coefficient values range from -1 to 1. A correlation of -1 and 1 between two variables A and B indicates that they are perfect correlated to each other (either positive



(a) Couplings of microservice-pairs



(b) Afferent couplings



(c) Efferent couplings

Fig. 5. Coupling between microservices (Normalized).

Table 3
Correlation analysis with existing coupling metrics.

		aMCI	eMCI	Ca	Ce	AIS
eMCI	*coef.*	0.19				
	sig.	0.04				
Ca	*coef.*	0.93	0.26			
	sig.	8E−51	0.005			
Ce	*coef.*	0.19	0.96	0.27		
	sig.	0.04	2E−61	0.003		
AIS	*coef.*	0.94	0.27	1.00	0.28	
	sig.	3E−52	0.004	5E−143	0.002	
ADS	*coef.*	0.21	0.97	0.30	0.99	0.31
	sig.	0.02	1E−66	0.001	1E−96	0.001

or negative). While a correlation of 0 means that there is no tendency for B to either increase or decrease when A increases.

The correlations from Table 3 suggest monotone associations among the afferent and efferent coupling metrics. Please note that we did not conduct the correlation analysis of the microservice-pair couplings, because, due to the dominated zero-measurements (over 90% of the sample), the exhibited correlation coefficient could be highly skewed. The present correlation results in Table 3 are on the basis of afferent and efferent coupling measurements (both with less than 40% of non-zero values). Specifically, the correlations between aMCI and Ca, and that between aMCI and AIS settle at 0.93 and 0.94 respectively, while the correlation between Ca and AIS is as high as 1.00. Likewise, the correlations between eMCI and other microservices efferent coupling metrics

Table 4
Principal component analysis with existing coupling metrics.

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6
Eigenvalues	2.88	2.09	0.57	0.42	0.05	0.04
Percentage	56.57	38.13	3.03	1.18	0.83	0.25
Cum. percentage	56.57	94.70	97.73	98.91	99.75	100
aMCI	0.30	−0.41	**0.78**	−0.30	0.17	0.07
eMCI	0.38	0.37	0.34	**0.76**	0.05	−0.15
Ca	0.38	**−0.46**	**−0.45**	0.22	**0.61**	0.16
Ce	**0.45**	0.38	0.15	**−0.47**	0.23	**0.60**
AIS	0.43	**−0.45**	0.20	0.08	**−0.71**	0.25
ADS	**0.48**	0.38	0.09	−0.22	−0.20	**0.73**

(Ce and ADS) are 0.96 and 0.97 respectively, while Ce and ADS are perfectly correlated with a coefficient quite near to 1.

The above statistical results are mostly within our expectation. Specifically, Ca and AIS, as well as Ce and ADS have the highest correlation in the results, because both Ca and AIS (Ce and ADS) are calculated using only the inter-service dependencies. By contrast, the correlations between our developed metrics and the existing ones are relatively small (although also over 0.9). The relatively small coefficients are because our metrics still consider the size of microservices compared to existing coupling metrics, following the relative measurement theory. Furthermore, the high correlation between relative and absolute measurements is reasonable, because all their calculations utilize inter-service dependencies. Particularly, in more than 60% cases, the number of inter-service dependencies is 0 (zero), so their coupling measurements are all 0 (zero). In general, this implies that the microservices relied on by many other microservices (AIS) or many elements of other microservices (Ca) are likely to have high afferent coupling (aMCI) and vice versa, and the same can be said regarding the microservices efferent coupling.

Whilst the correlation results show that these measures are statistically associated, we now explain whether there is a significant difference between MCIs and other measures from a statistical point of view. To verify from a statistical viewpoint that the our measures differ significantly from the existing coupling metrics, we employed Principal Component Analysis (PCA) (Abdi and Williams, 2010)—an unsupervised machine learning algorithm for exploring the underlying relations of several input parameters. PCA is a technique that has been widely used (e.g., in Baig et al. (2019), Cazzola and Favalli (2022), Al Dallal and Briand (2012)) to identify the orthogonal dimensions that explain the relationships between software quality metrics. Here it is used to understand the relationship between the existing coupling metrics and the proposed ones. In addition, it also demonstrates that our proposed microservice coupling metrics capture new measurement dimensions. To be specific, if aMCI and eMCI are influential metrics contributing to the principal components, it means that these metrics capture new measurement dimensions that are not addressed by any of the other coupling metrics considered in the analysis. This paper used singular value decomposition (SVD) (Klema and Laub, 1980) to perform PCA.

Table 4 presents the results with all six principal components (PC), as well as the PC eigenvalues, explained variance ratio, and accumulated variance ratio. The PC eigenvalues represent the relative share of total variance that is explained by the component. A higher eigenvalue of PC means that the component is of higher magnitude and more significance. For every PC, the correlation coefficients between the measures and the identified PCs are also reported. The absolute of these coefficients represent how much the metrics contribute to the captured dimensions. Based on recommendations from the literature, we applied a 0.45 cutoff to interpret the metrics as relevant to components, and applied a 0.63 as very good (Finch et al., 2017). Each of these principal components are analyzed with the coefficients as follows:

- PC 1: The efferent coupling metrics Ce and ADS are influential metrics for this PC, with coefficient values of 0.45 and 0.48, respectively. Whereas all the other metrics have a relative weak influence on this PC (but with at least 0.30 coefficient).
- PC 2: The afferent coupling metrics Ca and AIS are influential for this PC, with coefficient values of −0.46 and −0.45, respectively. In the meantime, we see that other measures also contributed a lot to this component.
- PC 3: The proposed aMCI metric is the most influential metric for this PC, and has a fairly high coefficient value, i.e. 0.78 (higher than 0.63).
- PC 4: eMCI and Ce are the metrics that matter for this component. Moreover, the coefficient value of eMCI in this PC is good enough that arrives at 0.76.
- PC 5: Ca and AIS are influential for this PC, with high enough coefficients at 0.61 and −0.71, respectively.
- PC 6: For this PC, Ce and ADS are influential whose coefficients reach 0.60 and 0.73.

The principal component analysis shows that our proposed microservice coupling metrics cover two new dimensions as they are significant factor in PC 3 and 4 respectively, and these components both take a relatively important share of the total variance (according to the eigenvalues). This proves that aMCI and eMCI both capture a new dimension in terms of microservice couplings. These evidence supports our answer of this RQ that the proposed MCI metrics are correlated with and significantly different from existing microservice coupling measures.

4.3. RQ2: The discriminative power of MCI

One objective of measuring microservice coupling is to allow for comparing different designs, and to use the results as indicators for selecting the best design and improving the quality of weakly coupled microservices. The discriminative power focused in this RQ is an assessment about whether a metric is capable of separating high-quality and low-quality components (Arvanitou et al., 2015). For microservice coupling, if a metric's discriminative power is poor, it means that the chances that it will differentiate between differently coupled microservices are low; thus its usefulness as a coupling indicator is questionable. As a result, there exists a high percentage of cases in which the metric incorrectly considers different microservices to be the same in terms of coupling, which may to some degree discourage software developers from applying the metric to assess their product quality.

The root question is, how to measure the discriminative power of a metric? In this section, we first adapt a discriminative power measurement from Al Dallal (2010) that can be extracted from the code repository of a software system. After that, we use this measure to compare the discriminative power of the coupling metrics under consideration.

4.3.1. Discriminative power measurement

We first focus on the discriminative power of a metric in quantifying the coupling between two microservices, and then extend the measurement to other types of coupling metrics. We illustrate the rationale using five cases depicted in Fig. 6.

The coupling-related information between two microservices can be depicted using the connections between the services' members. The way in which the members are connected and coupled is referred to here as the Connectivity Pattern (CP) (Al Dallal, 2010). Suppose there are two microservices in a system, say Caller and Callee, the Caller service has implemented two entities (e.g., interface and inner classes), and the Callee service has published two interfaces. In the first case in Fig. 6, all entities in Caller rely on all the public interfaces of Callee; on the other extreme, none of the entities in Caller depend upon the interfaces of Callee; in the third case, half of the entities in Caller microservice invoke all interfaces of Callee. It is obvious that, in these cases, the two microservices are differently coupled: in the first case, they have the worst coupling; in the second case, they are least coupled. A microservice coupling metric with high discriminative power, is expected to clearly indicate such differences in the degrees of coupling. From the three cases, we see that the differences in their connectivity patterns are manifested in the number of Coupling Interactions (CI) between microservices. That is, a discriminative coupling metric has a high probability of obtaining different values for the different CIs among them.

Furthermore, the sizes of microservices can make a difference on the possible connectivity patterns between two microservices. Generally, as interface number of the Callee and entity number of the Caller increase, the possible number of coupling interactions in the microservices becomes larger (see the 4th and 5th cases in Fig. 6). Accordingly, the connectivity pattern from microservice Caller M_i to microservice Callee M_j can be defined by entity number of the Caller $|Ent_i|$, interface number of the Callee $|Int_j|$, and number of coupling interactions among them $|CI_{i,j}|$. Formally, it can be considered as a triplet: $CP(M_i, M_j) = (|Ent_i|, |Int_j|, |CI_{i,j}|)$. Take Fig. 6 as an example, overall 5 distinct connectivity patterns can be derived due to their distinct values of entity, interface and coupling interaction numbers. We thus propose the following discriminative power measure for metrics that quantify the coupling between two microservices.

Discriminative Power Measure (DPM): the probability that a metric will obtain different coupling values for two microservices with different connectivity patterns in terms of the sizes of Caller entities and Callee interfaces, as well as coupling interactions among them. Formally, the discrimination measure of metric m in a microservice system M is defined as:

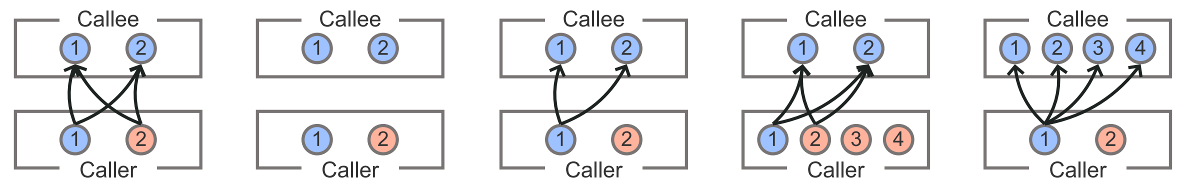
$$DPM(M, m) = \frac{\#DistCV(M, m)}{\#DistCP(M)}$$

Where #DistCP(M) is considered the number of possible connectivity patterns between every two microservices in system M, #DistCV(M,m) is the total number of distinct coupling values when metric m is applied to all the distinct connectivity patterns of M.

The same DPM measure can be applied to the afferent coupling metrics of a microservice, but the connectivity pattern is now defined by the entity number of all the Caller services, the interface number of the Callee, and the coupling interaction number from all the Caller services to the Callee. Similarly, the connectivity pattern for the efferent coupling metrics is defined by the entity number of the Caller service, the interface number of all the Callee services, and the coupling interaction number from the Caller to all the Callee services. Next we estimate the discriminative power of the MCI metrics using the DPM measure.

4.3.2. Results and analysis

Given the discriminative power measure, we executed a program (also accessible on GitHub[1]) on each of the open source projects to calculate the DPM measure of the microservice coupling metrics under consideration and listed the results as Table 5. The first and second columns of Table 5 show the name of the project and its number of services. The third, 8th, and 12th columns report the number of all distinct connectivity patterns in each project, in terms of microservice-pair coupling, afferent



(1) Case 1: #ent = 2, #int = 2, #CI = 4
(2) Case 2: #ent = 2, #int = 2, #CI = 0
(3) Case 3: #ent = 2, #int = 2, #CI = 2
(4) Case 4: #ent = 4, #int = 2, #CI = 4
(5) Case 5: #ent = 2, #int = 4, #CI = 4

Fig. 6. Illustrative examples for DPM.

Table 5
Discriminative power of microservice coupling metrics.

Project	#Services	Microservice-pair coupling					Afferent coupling				Efferent coupling			
		DistCP	MCI	CaT	CeT	ACT	DistCP	aMCI	Ca	AIS	DistCP	eMCI	Ce	ADS
Light-reading	4	10	**0.40**	0.30	0.20	0.20	4	**0.75**	**0.75**	**0.75**	4	**0.75**	**0.75**	**0.75**
EPR	4	5	**0.40**	**0.40**	**0.40**	**0.40**	3	**0.67**	**0.67**	**0.67**	3	**0.67**	**0.67**	**0.67**
Xavius	5	11	**0.27**	0.18	0.18	0.18	5	**0.60**	0.40	0.40	5	**0.60**	**0.60**	**0.60**
Madao_service	6	24	**0.25**	0.17	0.13	0.08	6	**0.67**	0.50	0.50	6	**0.83**	0.50	0.50
Rpush	6	17	**0.24**	0.18	0.12	0.12	5	**0.60**	**0.60**	**0.60**	5	**0.80**	0.40	0.40
Enarx	7	12	**0.25**	0.17	0.17	0.17	6	**0.50**	0.33	0.33	6	**0.50**	0.33	0.33
Cangjingge	7	19	**0.11**	**0.11**	**0.11**	**0.11**	6	**0.33**	**0.33**	**0.33**	6	**0.33**	**0.33**	**0.33**
Mall-swarm	7	38	**0.21**	0.05	0.08	0.05	7	**0.71**	0.43	0.43	7	**0.71**	0.57	0.57
RuoYi	7	26	**0.12**	0.08	0.12	0.08	7	**0.43**	0.29	0.29	7	**0.43**	**0.43**	0.29
Recruit	8	28	**0.36**	0.07	0.07	0.07	8	**0.88**	0.63	0.63	8	**0.88**	0.50	0.50
Job-offters	8	37	**0.11**	0.05	0.08	0.05	8	**0.38**	**0.38**	**0.38**	8	**0.50**	0.38	0.25
Grocery	8	30	**0.10**	**0.10**	**0.10**	0.07	8	**0.25**	**0.25**	**0.25**	8	**0.38**	**0.38**	0.25
Snowy	10	60	**0.03**	**0.03**	**0.03**	**0.03**	10	**0.20**	**0.20**	**0.20**	10	**0.20**	**0.20**	**0.20**
Mall-cloud	13	102	**0.09**	0.04	0.04	0.02	13	**0.54**	0.23	0.31	13	**0.54**	0.31	0.31
Zscat	13	113	**0.05**	0.04	0.03	0.02	13	**0.23**	0.15	**0.23**	13	**0.46**	0.23	0.15
Average	7.53	35.47	**0.13**	0.07	0.07	0.06	7.27	**0.48**	0.36	0.38	7.27	**0.54**	0.39	0.36

coupling, and efferent coupling, respectively. Generally, when the number of services increases, the number of distinct connectivity patterns per project increases. The calculated DPM values for the considered microservice coupling metrics are listed in the remaining columns, in which the best results for each project is highlighted in boldface.

In most of the cases, the DPM values of a coupling metric decrease as the service number in a project increases. This is due to the fact that mostly the number of distinct coupling values increases much slower than the number of unique connectivity patterns, as the number of services increases. This means that mostly the likelihood of lacking discrimination anomaly (LDA) (Al Dallal, 2010) of coupling measurement increases as the size of a software system increases. However, this observation does not always hold. For example, in the Recruit project with 8 services, the DPM value of MCI is 0.36, much higher than that in the projects with 7 services, where the DPM values vary from 0.11 to 0.25. On the other hand, the DPM values of CaT, CeT, and ACT in this project are 0.07, generally smaller than that in those 7-services projects. It means that in some projects, a discriminative metric can significantly mitigate the LDA issue in microservice coupling measurements.

It can be noticed that MCI has the largest DPM values among all the metrics considered for microservice-pair couplings, afferent couplings, and efferent coupling in Table 5. For example, in the Mall-swarm project, the discriminative power of MCI is estimated as 0.21, while that of CaT, CeT, and ACT vary from 0.05 to 0.08, merely 23.8% to 38.1% of MCI. In addition, in the afferent coupling metrics, aMCI outperforms others with an average DPM of 0.48; and in the efferent ones the average of eMCI is high at 0.54. This indicates that the MCIs metrics are most capable in separating high and low coupled microservices, and it is more reliable to use MCIs as an indicator for the quality management of microservices.

The other metrics also reach the highest DPM values in some projects. This is in part due to that in such projects (e.g., EPR), the inter-service dependencies are quite few, which leaves most microservices with a coupling degree of zero and insignificant differences of MCI and other metrics in discriminating coupling degrees. In addition, the smallest DPM values in terms of the afferent (efferent) couplings are usually related to both Ca and AIS (or both Ce and ADS). It suggests that all these existing coupling metrics are poorly discriminative for measuring microservice coupling.

4.4. RQ3: Association with change impacts

This RQ aims to investigate if microservices with higher MCI values are less independent, and as a result, they are more difficult to fix and change during their evolution. Change impact, implying that a software artifact is less independent because it has to change due to changes in other artifacts of the system, has been considered as one of the most significant maintenance issues in software development (Kretsou et al., 2021). On the other hand, recent industrial survey shows that microservice couplings could lead to ripple effects on changes which make adding or changing functionality slower and error-prone (Bogner et al., 2019). Thus, in this paper, we focus on empirically evaluating the association between MCI metrics and the independence of microservices using change impact analysis.

Such an empirical evaluation can be achieved using either targeted controlled experiments under research settings, or case studies of industrial (or open source) software products (Briand et al., 1999). Both strategies can have a contrasting effect on the

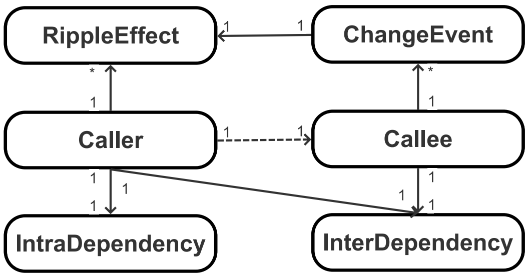
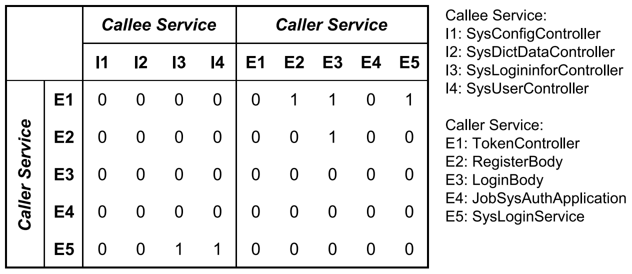


Fig. 7. Core elements in change impact analysis.



		Callee Service				Caller Service				
		I1	I2	I3	I4	E1	E2	E3	E4	E5
Caller Service	E1	0	0	0	0	0	1	1	0	1
	E2	0	0	0	0	0	0	1	0	0
	E3	0	0	0	0	0	0	0	0	0
	E4	0	0	0	0	0	0	0	0	0
	E5	0	0	1	1	0	0	0	0	0

Callee Service:
I1: SysConfigController
I2: SysDictDataController
I3: SysLogininforController
I4: SysUserController

Caller Service:
E1: TokenController
E2: RegisterBody
E3: LoginBody
E4: JobSysAuthApplication
E5: SysLoginService

Fig. 8. Dependency matrices constructed for Fig. 2.

internal and *external* validity (Siegmund et al., 2015) of the evaluation results. More specifically, controlled experiments provide greater support for minimizing instrumentation effects which can influence the internal validity of the evaluation, but can negatively affect external validity because such experiments are not necessarily representative of real-world cases (Perepletchikov and Ryan, 2010). In contrast, case studies maximize external validity, but could be affected by instrumentation effects. Thus, in this section, we present a study that follows the first approach since it has been suggested that the internal validity of a study should be investigated prior to establishing external validity (Perepletchikov and Ryan, 2010).

MSA is an architecture style where polyglot technology stacks are often employed in practice (Lewis and Fowler, 2014), including polyglot interprocess communication and programming languages, and as such it is time-consuming and laborious to conduct a large-scale industrial investigation at this stage of the study. Moreover, although we can find some open source microservice projects developed using a unified technology stack, we found that most of these projects have only a short evolutionary history. The reason is probably because MSA architectures require high development costs, and as a result, most of these (non-profit) open source projects are of small or intermediate scale, and with low complexity. However, from these projects, we can only capture a few instances of inter-service change impacts, which are not enough for statistical analysis with the proposed coupling metrics to draw valid conclusions.

In the following, we first analyze and model the change impacts among microservices and then propose a suite of measures for quantifying them for evaluation purpose in this study. After that, we introduce their measurement process using a series of simulation-based experiments, and their correlations to MCIs and other baseline coupling metrics. Please note that we first focus on the change impacts among two microservices, and then extend the analysis of change impacts in other more complex situations.

4.4.1. Change impact analysis

Regarding to the change impacts between two microservices, the core elements are depicted in Fig. 7 in form of a class diagram. Specifically, a service being called (Callee) corresponds to a service that calls it (Caller), which forms an unique state of inter-service dependencies between them. At the same time, the invocation dependencies within the Caller service are also unique. Based on this architectural state, the Callee service may undergo several change events, each having a different impact on the Caller service. Next we detail the introduction of each of the elements.

1. Callee service: The microservice Callee is the one being depended upon by another microservice, e.g., the jobsys-system service in Fig. 2. Although each microservice is constituted by a set of interface and inner classes, the changes made to those inner classes of a Callee often do not influence its dependents (because the contracts are not changed). Therefore, in the analysis of microservice change impacts, only the interface classes of a Callee are of our interests. Generally, the **changes** made to a Callee have the potential to affect the Caller that depends on it, if and only if, all of the following conditions are satisfied:

- The changes are made to its interfaces, because only the interfaces of a Callee service have the potential to be directly invoked by other microservices in MSA.
- The changes made are about a deletion or modification of an existing operation, while the addition of a new operation may not affect the Caller service. Note that an operation's modification can be considered as first deletion and then addition, thus we collectively refer to the changes that could impact other microservices here as operation deletion.
- The deleted operation is directly called by the Caller service. In such cases, the microservice Caller has to make corresponding changes to avoid conflicts, or else it may invoke a non-existent operation.

2. Caller service: Each microservice that depends on others plays the role of Caller service, e.g., the jobsys-auth service in Fig. 2. Each Caller service consists of multiple entities (including interface and inner classes), each of which could possibly invoke the published interfaces of Callee. The **ripple effects** on a Caller that are imposed by a change made to Callee can be divided into two aspects: direct and indirect ripple effects.

- Direct ripple effect: the entities in a Caller service that directly invoke a Callee service, could be directly affected by the changes made to the Callee. For example, the 5th entity of Caller in Fig. 2 could be directly affected by changes made to the third and 4th interfaces of Callee.
- Indirect ripple effect: the directly affected entities in Caller, could further emit changes to its dependents, thereby causing those dependents to be indirectly affected by the Callee. Take Fig. 2 as an example, the affected 5th entity of Caller could lead to changes to the first entity, if the 5th entity modifies the declaration of its publish methods.

3. Inter-service Dependency Matrix (InterDM): As aforementioned, a prerequisite for whether the change of a Callee will influence its Caller is that the Caller directly depends on the interfaces that are modified. To represent the dependency relationship between two microservices, a binary matrix whose row number k is the number of Caller entities and whose column number l is the number of interfaces in Callee is leveraged (see the left sub-matrix in Fig. 8). The matrix has rows indexed by the entities of Caller and columns indexed by the interfaces of Callee, and so for

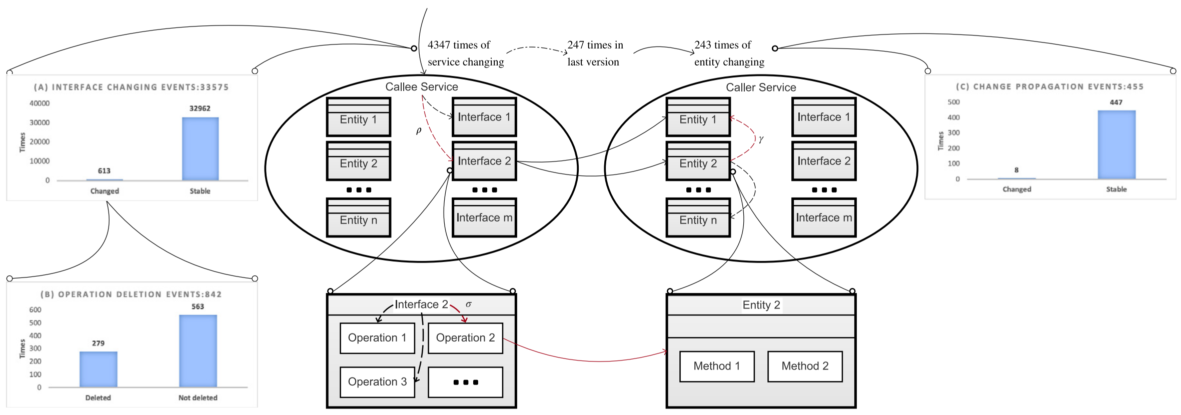


Fig. 9. Change impact Bayesian network of a microservice-pair.

$1 \le i \le k$, $1 \le j \le l$, we write:

$$InterDM_{ij} = \begin{cases} 1 & \text{if the } i\text{th entity of Caller remotely} \\ & \text{invokes the } j\text{th interface of Callee} \\ 0 & \text{otherwise} \end{cases}$$

For example, the $InterDM_{53}$ is set to 1 due to the dependency from the 5th entity of Caller to the third interface of Callee. Nevertheless, this 2D dependency matrix is not enough to accurately describe the change impacts. Because, the changes of an interface in Callee can only propagate to Caller's entities if these entities directly invoke the changed operations, as discussed earlier. Therefore, in order to accurately analyze such impacts, it is necessary to model the operations. In more details, every interface of Callee, int, can be represented as a non-empty operation vector, Op_{int}. For every entity of Caller microservice, ent, that depends upon int, the operation subset of int that are invoked directly by ent, denoted by $Op_{ent,int}$, is utilized, where the $Op_{ent,int} \subset Op_{int}$ relation is ensured.

4. Intra-service Dependency Matrix (IntraDM): Based on the *interDM*, the entities of Caller that will be directly impacted by changes made to Callee can be determined, which corresponds to the direct ripple effects. In addition, the affected entities in Caller could further emit changes to other entities that depend on them, which results in the indirect ripple effects of Callee. To describe such indirect effects, another binary matrix can be constructed to represent the dependencies within the Caller microservice (see the right sub-matrix in Fig. 8), where the rows and columns of this matrix are both the entity number of Caller. More specifically, this matrix has rows and columns both indexed by the entities of Caller, and so for $1 \le i \le k$, $1 \le j \le k$, we have:

$$IntraDM_{ij} = \begin{cases} 1 & \text{if the } i\text{th entity invokes the } j\text{th, } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

Here we set the diagonal values in intraDM as zeros, because such cells represent dependencies within an entity itself (e.g., an reference from a method of an inner class to its attribute), which would not affect other entities and cause additional change impacts.

4.4.2. Change impact modeling

From the above analysis, we see that there exist both deterministic and non-determinate factors in the change impacts among microservices. For example, it is interminate if a change made to a Callee microservice is about the interfaces. Generally, the uncertainty, affecting architecture's modification, can be attributed to many facets, such as changes in feature requirements and developers' decision makings. For modeling the non-determinate factors, we now try to build a probabilistic model based on Bayesian approach which is appropriate for modeling quality issues with uncertainties, such as change impacts (Tang et al., 2007; Salama and Bahsoon, 2017). We adhere to the Bayesian approach – an ideally suited knowledge representation for reasoning and decision making under uncertainty – in the probabilistic modeling, because it is guaranteed to define a unique probability distribution over the network variables. By modeling the unknown parameters of the sampling distribution through a probability structure, the Bayesian approach supports a quantitative discourse on these parameters (Robert, 2007).

The probabilistic modeling based on Bayesian consists of two components: the structure often referred as the qualitative model, and the parameters (i.e. conditional probabilities) referred as the quantitative one (Kjaerulff and Madsen, 2008). Thus, we first deduce a probabilistic relational model for microservice change impacts, as depicted in the center of Fig. 9. In the relational model, the elements that are depicted are from only two microservices, namely Callee and Caller, for showing how the changes made to the Callee can be propagated to the Caller. An edge within the Callee service represents a "whole-part" relationship, meaning that a Callee service is represented by a set of interfaces and an interface constituted by a set of operations. In the meantime, if a "whole" changes, then, each of its elements (i.e. "parts") will have a certain possibility (the ρ and σ probability in Fig. 9) of changing. Otherwise, all of the parts will remain in a stable state. That is, the changes made to the whole and part elements are causally connected. Second, the edge of a solid line, which corresponds to a remote invocation between microservices, expresses a deterministic change impact between them. For example, if an entity of Caller subscribes an operation that has to be deleted in Callee in a commit, then, in the following commits this entity will have to make corresponding changes to avoid potential errors. According to these edges, we can definitely determine whether the changes made to the Callee service will affect the Caller, and which entities will be affected. Finally, each dashed edge within the Caller service corresponds to a direct dependency and an invocation between two entities, and thus the changes made to a dependee would possibly affect the dependent (corresponding to γ in Fig. 9), if the changes are about the contracts between them.

The relational model could be read as follows: if the Callee service changes, each of its interfaces will have a certain probability (ρ) to be modified. The modification made to an interface, generally imposed on its constituent operations, could affect the Caller microservice that depends on it only when the changes are about the deletion of existing operations, as aforementioned in Section 4.4.1. If an interface changes, then, each of its existing operations will have a probability (σ) to be deleted. Once there is an operation deleted in Callee service, the entities in Caller that directly invoke the operation will definitely have to make

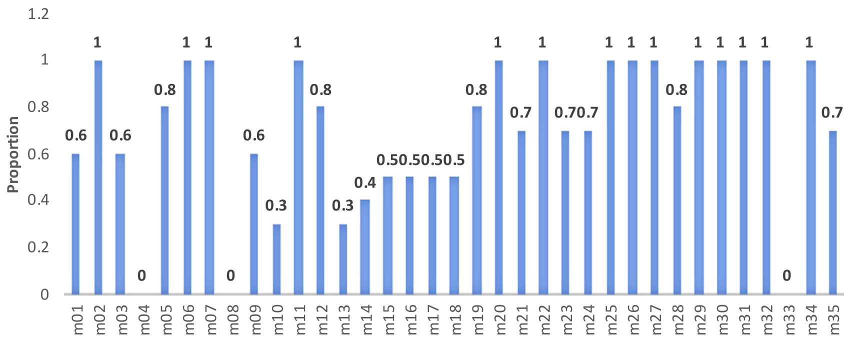


Fig. 10. Proportion of commits with one changing interface.

corresponding changes, which lead to the direct ripple effects on Caller. Moreover, as discussed before, the directly affected entities may further propagate their changes to other entities (with probability γ), which are referred as the indirect ripple effects.

Next we discuss the quantitative non-deterministic factors (i.e. conditional probability parameters) in the relational model:

- Interfaces changing rate (ρ): *how likely an interface will be changed if its microservice is modified.* The higher the rate, the more change-prone an interface is.
- Operations deletion rate (σ): *how likely an operation will be deleted if its interface changes.* A higher score, means that the changes of a microservice have more opportunities to impact others.
- Changes propagation rate in a service (γ): *how likely the changes on an entity will be propagated to the other entities that depend on it.* The higher the value, the larger the indirect ripple effects of inter-services.

The above three rates all describe a probability of a Boolean-valued question, e.g., changed or not (stable). For estimating the value of these rates, each of them can be considered as a parameter that obeys the Binomial distribution population (Shao, 2003), if any "interface changing"/"operation deletion"/"change propagation" events are pair-wise independent. Take the interfaces changing rate as an example, suppose X is used to represent the number of changed interfaces, $X=1$ denotes that the interface has been changed, $X=0$ denotes a stable interface. Given that we used ρ to express the changing interface rate, then, X follows a Binomial distribution: $P(X=1)=\rho$, and $P(X=0)=1-\rho$.

To estimate the values of these parameters (ρ, σ, and γ) and build the change impact quantitative model (i.e. Bayesian network), we quantitatively analyzed and measured these parameters in the open source projects and then used the measurements as prior knowledge base. For this purpose, we chose 8 out of the 16 open source projects as the analysis subjects in the parameters estimation. We chose these projects, rather than using all 16 projects mainly because they have at least six months of revision history from which we can extract sufficient maintenance data for probability calculation. In the following, we first discuss whether the "interface change"/"operation deletion"/"change propagation" events are independent of each other so that ρ, σ, and γ can be considered as parameters that obey the Binomial distribution population, then estimate the values of these parameters using the point estimation method in statistics (Shao, 2003).

Before doing so, please note that the goal of this section is to build a Bayesian network for modeling how a change event made to a Callee service can impact its Caller. For simplicity, we limit the scope of a change event to a code commit, which is usually considered the smallest unit of software changes (Liu et al., 2020). Furthermore, the commit concept has been broadly used as a unit for describing and quantifying maintenance activities (e.g., Mo et al., 2016; Xiao et al., 2022). To provide a fair basis for our analysis of the change impact, we consistently use this change granularity (i.e. a code commit) in all our modeling, measurements, and experiments.

Intuitively, whether an interface is modified or not is not related to another interface. From the open source projects, we observed that in most of the commits no more than one interface is modified. More specifically, for 25 out of the 35 services with interfaces changed, at least 60% of the commits contained only one interface modification (see Fig. 10). In addition, the interface-pairs that are modified simultaneously in more than one commit account for only 0.07% of all possible interface-pairs. These indicate that changes made to interfaces are likely to be independent of each other. At the same time, the deletion of each operation in an interface is also independent of each other to a certain extent, because each operation is the smallest functional unit released by the interface, and thus the coupling between each other is relatively small. In addition, only one deletion event may occur to an operation, and it is difficult to observe that two operations are deleted simultaneously in more than one commit. Finally, whether modifications of different entities will affect the entities that depend on them are independent of each other, because the modifications made by these entities are often unrelated to each other. It is conceivable that even though these entities are modified for the same reasons, the modifications performed in the different entities could be very different due to the business requirements and developers' understandings.

Fig. 9 shows that among the 8 analyzed open source projects, a total of 4347 service changing events have occurred to the revision history. When a service changes, each of its interfaces will have a certain chance to be modified. Thus, in all these service changing events, overall 33575 interface changes may have occurred, but in reality, we only observed 613 interface changing events, see Fig. 9. (A). Based on recommendation from the literature (Shao, 2003), we apply the mean (0.02) as the point estimation of ρ value. Similarly, as shown in Fig. 9. (B), in the 613 interface changing events that occurred, each operation of the interfaces (832 in total) would have a certain possibility to be deleted. But in fact, we observed 279 operations deletion events, and the remaining 563 operations were categorized as not deleted. These un-deleted operations may just have their implementation modified, or just remain stable (i.e. changing other operations). Therefore, we used the average of 0.33 as the point estimation of operations deletion rate σ. When estimating the third parameter, we only used the latest versions of these open source projects. We chose the last versions, rather than using all versions mainly because, according to our observations, these latest versions are often more stable than their historical versions and thus less subject to large chunk of requirements update that could possibly bring many misjudged "co-changes". Among the 247 service changing events of the latest version, a total of 243 entity changing events occurred wherein a depended entity was changed. Each of such changes may further affect other entities that directly depend on the changed entity, so these 243 times of entity changing could have further caused 455 entities to change. However, we actually observed that only 8 entities were affected by the changed entities, as Fig. 9. (C), meaning that eight commits of depended entities are followed by changes occurred to the dependents in a successive commit, indicating that these dependents have changed possibly due to the ripple effects from the corresponding depended entities. Hence, the γ value is estimated using the mean value of 0.02 to represent the rate of change propagation within a service.

4.4.3. Change impact measurement

Based on the above analysis and modeling of change impacts, in this section we propose a suite of measures for quantifying such impacts across microservices. Suppose there are two microservices from a project, Caller and Callee. If every time changes made to the service Callee result in several files of Caller to change, it means that the change impact of Callee on Caller is so severe that the service Callee is not independently changeable and that the service Caller is unstable. On the other extreme, if the service Caller rarely reacts to changes made to service Callee, or the reaction is quite small, it means that these microservice can be changed independently to some extent. It is obvious that, in the first case, the microservices have the most change impacts, and in the second case the least impacts. We then propose the following measures to quantify the change impacts of service Callee on service Caller:

- *Directly Affected Scope (DAS):* number of changed code files ("scope") in the Caller service due to the direct ripple effects of the Callee. The higher the score, the more affected the microservice is.
- *Directly Affected Frequency (DAF):* number of times the files in the Caller service have to change directly due to changes made to the Callee.
- *Overall Affected Scope (OAS):* number of changed code files in the Caller service due to the direct and indirect (overall) ripple effects from the Callee service.
- *Overall Affected Frequency (OAF):* number of times the files of in the Caller service were changed due to the direct and indirect ripple effects from the Callee service.

These measures manifest how much a microservice was affected and how costly (in terms of frequency) were the affected changes due to its dependencies on another microservice. The following section introduces how we collect these measures based on their corresponding definitions.

4.4.4. Measurement strategy and process

The above measures manifest how a microservice was affected by another during its maintenance. To fairly collect such measurements, ideally, a set of microservices projects of various sizes and domains should be chose firstly. To investigate the change impacts among microservices during their maintenance, for each project, the microservices that have reached their stable design should be selected, on which there is no significant requirements changes themselves. Given these microservices, we should measure their MCI values, then monitor and collect the change impacts their depended services imposing on them for a long enough period of time to get statistically valid results. Ideally, the change impacts from each of their depended services should be analyzed separately, so that we can compare if the services that have higher MCIs with them indeed impose larger ripple effects on them, and thus they are less independent of these services.

However, finding such controlled microservices projects is not trivial in reality. Together with the consideration of the polyglot nature of MSA (discussed at the beginning of Section 4.4), we decided to investigate the change impacts between microservices and analyze their association with MCI metrics through a series of experiments. In the following, we introduce an algorithm we designed for simulating such impacts for this experimental evaluation. Please note that the algorithms used in this section are also provided online as an open access artifact[1] to enable reproducibility.

The algorithm in Fig. 11 shows how to estimate the change impacts of service Callee on service Caller. The algorithm uses a binary $k \times l$ matrix to represent the inter-service dependencies, and a binary $k \times k$ matrix for the intra-dependencies within the Caller service, where k is the number of entities of Caller and l is the number of interfaces of Callee. In addition, a set of vectors with ranging sizes are employed to represent the operation set of these interfaces and how the entities depend on the interfaces. In the meantime, the algorithm uses four parameters, where the three probabilities ρ, σ, and γ are set to simulate an random change event to service Callee and its ripples effects on service Caller, as aforementioned, and the time of iterations t is designed to represent the times of change events for reducing the effect of chance factors on experimental results.

The algorithm first generates the core constituents of the Caller and Callee services. These are two randomly created binary matrices to represent the inter- and intra-dependencies, as well as the operation sets of all the interfaces and the operation subsets for showing how the entities depend on the interfaces. After that, the coupling values between Caller and Callee are measured using the four microservice-pair coupling metrics ACT, CaT, CeT, and MCI.

In the next step, we simulate t change events made to the Callee service, and measure how much the Caller service will be impacted and how costly will be the affected changes using the four change impact measures. For every interface of the Callee service, if it is to be changed according to the probability ρ, then, for each of its operations, we further check whether it is to be deleted using the predefined probability σ. If none of its operations is to be deleted, the interface is ignored because it represents a change that will not impact the Caller service. Otherwise, the deleted operations are added to the list of doL.

Using the deleted operation set, we now quantify the impacts that are directly imposed on the Caller service (Step 5.3 to 5.6). From each operation that has been "deleted", we now traverse the inter-dependency matrix and the corresponding dependent operation sets to find the entities that directly invoke this operation. Detecting an entity relying on a "deleted" operation suggests that it has to be changed accordingly to avoid potential conflicts. Hence, once such an entity is detected, the affected frequency increases by one, and the entity is added to the list of directly affected entities. After traversing from all the deleted operations, we calculate the DAS and DAF value of this iteration. These are expressed as the number of entities in the directly affected entities list and the affected change frequency, respectively.

Based on the directly affected entities, we need to further traverse the intra-dependency matrix so as to estimate the indirect ripple effects on the service Caller (Step 5.7 to 5.9). More specifically, from each of the directly affected entities, we traverse forward the entities that depend on it along with the dependency path and ripple the changes by predicting whether the entities in the path will be affected. For determining whether an entity will be affected, we consider the fact that if all the direct

```
Algorithm: Calculating the change impacts of service Callee on service Caller
Inputs:
coupling metric set: MCI, CaT, CeT, ACT
sizes of microservices: entity number of Caller $k$, interface number of Callee $l$, and operation number range per interface $[a, b]$
parameters setting: iteration times $t$, interfaces changing rate $\rho$, operations deletion rate $\sigma$, change propagation rate $\gamma$
Output: coupling values, DAS, DAF, OAS, OAF
1.  Create a binary matrix InterDM of size $k \times l$
2.  Create a binary matrix IntraDM of size $k \times k$
3.  Create $l$ operation sets of sizes ranging from $a$ to $b$, each corresponding to a Callee interface. And for each operation set,
    create $k$ subsets for the entities representing how the entities of Caller depend on Callee
4.  Calculate the coupling values using the metric sets
5.  For each iteration within time $t$, simulate a change event made to Callee:
    5.1 Create an empty deleted operations list $doL$
    5.2 For each interface of Callee, do
        5.2.1  Randomly determine whether it is to be changed with probability $\rho$, and the operations to be deleted using $\sigma$
        5.2.2  If it is not changed, or if none of its operations are deleted, then go to Step 5.2 (i.e., consider the next interface)
        5.2.3  Add the deleted operations to list $doL$
    5.3 Create an empty unique directly affected entities list $DAEL$
    5.4 Initialize the number of affected change frequency $acf$ as zero
    5.5 For each deleted operation in $doL$, determine the directly affected changes in Caller
        5.5.1  For each Caller entity $ent$ directly depending on the operation, do
                   Add $ent$ to $DAEL$ list, and add one to $acf$
    5.6 DAS = number of entities in list $DAEL$, DAF = $acf$
    5.7 Create a unique overall affected entities list $OAEL$ that involves all elements of $DAEL$
    5.8 For each entity $ent$ in $DAEL$, determine the indirectly affected changes in Caller
        5.8.1  For each entity $ent'$ that directly and indirectly depends on $ent$, do
                   If its directly depended entity is changed, then randomly determine whether it is to be changed using $\gamma$ as
                   the probability, if so, then add $ent'$ to $OAEL$ list, and add one to $acf$
    5.9 OAS = number of entities in list $OAEL$, OAF = $acf$
6.  Calculate the average of DAS, DAF, OAS, OAF of $t$ times.
```

Fig. 11. Algorithm for simulating the change impacts of service Callee on service Caller.

dependents of an entity have not be changed, then the current entity will not be affected. Otherwise, we use the probability γ to randomly decide whether to ripple the entity. After traversing all directly affected entities, we calculate the OAS and OAF values using the number of entities in the OAEL list that includes all the directly and indirectly affected entities, and the number of affected change frequency, respectively.

This algorithm is implemented as a Java-based tool. After simulating all t times of changes to the Callee, the tool generates the average change impact values for each of the four measurements under consideration.

4.4.5. Experiment and analysis

Given the measures of change impacts—Directly Affected Scope (DAS), Directly Affected Frequency (DAF), Overall Affected Scope (OAS), and Overall Affected Frequency (OAF), and the designed simulation algorithm for collecting these measures, we now conducted a series of experiments to explore the correlation between MCIs and the change impacts among microservices. For the correlation analysis, we decided to use Spearman's rank correlation coefficient due to the non-parametric nature of the studied microservice metrics. In this section, the Spearman's correlation coefficient provides a measure of the association between microservice coupling values and the four change impact measures within the simulation samples. Note that we first focus on the correlation between couplings among two microservices and their change impacts, and then extend the simulation algorithm for exploring the correlation between other types of coupling measures and the corresponding change impacts.

1. Coupling between a microservice-pair

In this subsection, we aim to investigate if two microservices, i.e. Caller and Callee, with higher MCIs are involved in larger change impacts. For benchmarking purposes, we let each Caller service implement 4 to 28 entities, each Callee service publish 1 to 4 interfaces, and each of the interfaces provide 3 to 7 public operations. These parameters were set based on the lower and upper quartiles of the statistical observations from the open source projects. For every pair of the randomly generated Caller and Callee services, we first calculated their coupling values using the MCI, CaT, CeT, and ACT metrics. In the meantime, we measured the average change impacts by simulating 1000 times of change events made to the Callee service. To account for the effect of microservice sizes on change impacts, we further normalized them by computing the change impacts of per Callee interface on per Caller entity—dividing the ripple effects by the product of number of Callee interfaces and number of Caller entities.

Table 6
Correlation of microservice-pair couplings and change impacts.

		DAS	DAF	OAS	OAF
MCI	*coef.*	0.87	0.87	0.87	0.87
	sig.	1E−31	2E−31	8E−32	1E−31
CaT	*coef.*	0.71	0.71	0.76	0.78
	sig.	1E−16	10E−17	4E−20	3E−21
CeT	*coef.*	0.32	0.35	0.34	0.36
	sig.	1E−3	3E−4	6E−4	3E−4
ACT	*coef.*	0.52	0.52	0.52	0.52
	sig.	3E−8	3E−8	3E−8	3E-8

Table 7
Correlation of afferent couplings and change impacts.

		DAS	DAF	OAS	OAF
aMCI	*coef.*	0.60	0.57	0.60	0.55
	sig.	5E−11	9E−10	4E−11	2E−9
Ca	*coef.*	0.35	0.32	0.40	0.37
	sig.	3E−4	1E−3	4E−5	2E−4
AIS	*coef.*	0.14	0.15	0.11	0.12
	sig.	0.175	0.135	0.280	0.229

Table 8
Correlation of efferent couplings and change impacts.

		DAS	DAF	OAS	OAF
eMCI	*coef.*	0.48	0.47	0.44	0.39
	sig.	4E−7	7E−7	4E−6	5E−5
Ce	*coef.*	0.13	0.17	0.08	0.09
	sig.	0.183	0.082	0.440	0.393
ADS	*coef.*	0.28	0.26	0.26	0.25
	sig.	0.005	0.008	0.010	0.012

Finally, we analyzed the relations between the coupling degree and the normalized change impacts within 100 samples. In this way, we were able to investigate whether there is a correlation between the MCI coupling values and the change impacts among a microservice-pair.

Table 6 presents the coefficient and significance values of the correlation analysis. All of these significance values are less than 0.05, indicating these correlated relationships are statistically significant. This table shows that all the four change impact measures have the highest positive correlation with MCI, meaning the higher the MCI, the more severe the change impacts of the corresponding service Callee on service Caller, and the less independent these services are. The CeT and ACT metrics similarly show positive correlations but the correlations are much weaker than with MCI. Although CaT also displays positive correlations with these change impact measures, its correlations with DAS and DAF are much lower, meaning that this coupling measure is less correlated with the direct ripple effects among microservices.

2. Afferent microservice coupling

In a second experiment, we investigated the relation between microservice afferent couplings and the corresponding change impacts by configuring ten coexisting Caller services and one Callee service, and analyzing the ripple effects of changes made to the Callee service on these Caller services. This experiment aims to explore the change impacts of one service on other services in a system. The simulation experiment configuration makes sure that the ripple effects on these Caller services are imposed by the same changes made to the Callee service. Given our aim to conduct a controlled experiment, an effort was made in the configuration to manipulate only the afferent coupling while keeping the size of service Callee (controlled variables) as constant in order to prevent its effect on the experimental results. Note that we still used random sizes of the Caller services (i.e. ranging from 4 to 28 entities) for simulating a real software project.

The microservice afferent couplings are calculated using aMCI and other baseline metrics: Ca and AIS. For measuring the change impacts of the Callee on all the Caller services, we accumulated the ripple effects that are imposed on each of the Caller services. To account for the influences of microservice sizes on change impacts, we further normalize them by computing the ripple effects per entity—dividing the four ripple effects measures by the number of overall entities in these Caller services. The correlation results are shown in Table 7. Table 7 shows that aMCI has the highest positive correlation with the four change impact measures, meaning that the greater the afferent coupling of a service, the larger the impacts of its changes on other services in the system, and it is less likely that this service is independently changeable.

3. Efferent microservice coupling

In a third experiment, we analyzed the change impacts of ten coexisting Callee services on one Caller service and their correlation with the microservice efferent couplings. In this experiment configuration, the changes made to each of the Callee services are independently generated (using the basic simulation algorithm), meaning that we do not concern about their ripple effects on each other for simplifying experimental design. Given our aim of conducting a controlled experiment, in the configuration an effort was made to only manipulate the efferent coupling while keeping the size of service Caller (controlled variables) as constant in order to prevent the effect on experimental results. In addition, in order to simulate a real software system, we designed these Callee services with various sizes, i.e. ranging from 1 to 4 interfaces.

Table 8 shows the correlation between microservice efferent couplings that are computed using eMCI, Ce, and ADS, and the corresponding change impacts of ten Callee services on one Caller service. The accumulated ripple effects are further normalized by dividing them by the total number of interfaces in the Callee services, on account of the size effects of changing interfaces. From Table 8, eMCI is the one mostly correlated with the four change impact measures. This indicates that the greater the efferent coupling value of a service, the more likely it will be affected by the changes made to other services in the system, and thus this service is more unstable.

5. Discussion

This section discusses the comparison of MCIs with other related metrics, their practical implications, and the threats to validity of this research.

5.1. Comparison with other related metrics

In this section, we compare the proposed microservice coupling metrics MCIs with the metrics adapted from the MSA literature. In addition to Ca, Ce, AIS, and ADS, several metrics were also applied to measure microservice-level coupling based on the interdependencies among services, such as CBM (Coupling Between Microservice) (Taibi and Systä, 2019), ExterCoup (External Coupling) (Selmadji et al., 2020), and FEC (Frequency of External Calls) (Taibi and Systä, 2019). All these metrics have been used to assess microservice decomposition alternatives in the migration from a monolithic architecture to an MSA (Li et al., 2019; Daoud et al., 2021).

The problem is, even though these metrics that reflect the absolute magnitude of 'coupling evidence' (e.g., call paths) between microservices have been demonstrated to be useful for MSA design in a system-level, they rarely focus on comparing the independence of individual microservices and identifying the most detrimentally coupled microservices in a system. Bogner et al. (2019) also pointed out the absence of coupling measures to assess individual microservices in architecture refactoring and MSA evolution.

Different from these existing metrics, our MCI is the only metric suite that measures how *dependent* the microservices are relative to the possible dependencies between them, based on the relative measurement theory (Allen and Yen, 2001). Our objective is microservices' comparison and refactoring support, i.e. identifying the most over-coupled microservices to indicate refactoring opportunities and comparing design alternatives for refactoring solutions selection. To the best of our knowledge, we are not aware of an existing metric that has successfully demonstrated its capability in comparing the coupling (and less independence) level of individual microservices in a system. Our empirical investigation indicates that MCIs outperform the existing metrics in separating groups of high and low coupled microservices (see Section 4.3) and in the positive correlation with microservices' independence (see Section 4.4). This suggests that MCI has the potential to be valuable for measuring, comparing, and refactoring microservices architecture.

5.2. Practical implications of MCI

MCIs' supports for refactoring. The experimental results suggest that a microservice having high eMCI is potentially change-prone, due to the ripple effect on changes of other services. In the meantime, a microservice with high aMCI value can potentially impose a large change impact on other services in the system. These implications largely confirm the downsides of high efferent and afferent coupling (Almugrin et al., 2016). As such, software managers and architects could use eMCI and aMCI to identify the services that are change-prone in the future and the services that are likely to be the root of unstable architecture. These represent clear refactoring opportunities for improving MSA design. For example, *one can identify the most vulnerable microservices in the system (e.g., most unstable), then compare refactoring solutions and select the one with the least interdependencies*. Nonetheless, refactoring with aMCI and eMCI could be time-consuming and labor-intensive for many start-up teams if there are many involved microservices. In such cases, software managers may find beneficial to consider micro-refactoring (Zhong et al., 2022) with MCI, i.e. *refactoring two microservices that are the most detrimentally coupled in a system.*

Possible means to improve MCIs. The results on MCI measurements revealed that a higher MCI value is generally negative for quality, as it is a significant indicative of larger change impacts in software maintenance. To address this issue, microservices developers are recommended to attain and maintain a low degree of dependencies (and a high degree of independence) between microservices. One promising strategy of reducing inter-service dependencies is to implement functionalities by promoting code reuse among microservices. Existing research (Fu and Cai, 2019) has shown that functionality overlap among software components can also benefit quality through enhancing understandability and lowering security vulnerabilities. However, this may cause semantic coupling (Fregnan et al., 2019) among microservices. If the relevant functionalities are changed frequently, this could introduce the "modularity violation" smell (Mo et al., 2019), which may increase the change impacts between microservices. This implies that practitioners have to take care of structure, semantic, and evolutionary information together when adopting this strategy. In addition, microservices developers may choose to reduce inter-service dependencies following the "microservices restructuring/refactoring" strategy (Zhong et al., 2022), e.g., extracting the code files involved in the intricate dependencies into a new microservice.

5.3. Threats to validity

This section explains the threats to the validity of our study (Wohlin et al., 2012; Siegmund et al., 2015), primarily in terms of how they were mitigated, but also about limitations in a few cases where they were not adequately addressed.

Internal Validity: stems from the conclusions that can be drawn about the causal effect of independent variables on dependent variables.

Selection. How the subjects are selected from a larger group can lead to the selection effects with respect to causality. To reduce this effect, we used randomly selected microservice open source projects as the subjects for case studies and used randomly generated Caller and Callee microservices in the experiments of coupling measurements and change impact analysis.

Reverse engineering. The accuracy of architecture recovery has the potential to affect causality. For recovering the interservice dependencies, our reverse engineering tool MicroParser automatically identified the HTTP requests and responses that were explicitly declared by developers using the OpenFeign annotations. Nonetheless, it is possible for developers to declare some incorrect requests or responses. To address this risk, our researchers performed a manual validation of the derived requests and responses.

Multiple groups. The control group and the selected experiment group are affected in the same way except the employed microservice coupling metrics for correlation analysis for reducing the potential effects of multiple groups experiments.

Experimental setting. The simulated experiments for change impact analysis in this paper depends on three parameters, ρ, σ, and γ, which represents the rate of interface changing in a service, operations deletion, and changes propagation between entities, respectively. For minimizing these threats, we performed empirical investigation on eight open source projects to collect sufficient data for these parameters estimation. We also collected additional evidence in these projects to show that the change of each interface and the deletion of operations are independent to each other.

External Validity: refers to the degree to which the obtained results can be generalized to other MSA-based systems.

Languages. Driven by a series of popular technology frameworks, e.g., SpringBoot and SpringCloud, most of the microservices-based systems we found in GitHub and GitLab were implemented using Java language. Thus, we focused on Java projects in this paper, and cannot claim that MCI is applicable to microservices projects written in other languages, such as Go or Javascript.

Technology frameworks. As aforementioned, the popularity of SpringCloud has been widely embraced by microservices developers. In addition, the Spring Cloud OpenFeign technique – a declarative REST client – makes the published service interfaces and operations explicit in codes, from which we are able to statically scan and detect inter-service dependencies. Hence, in this paper we measured microservices projects that use SpringCloud only and cannot claim MCI's generalization to projects using other techniques like RPC.

Environments. We only studied 15 open-source projects from GitHub and GitLab, which poses another threat to external validity. The reason is the scarcity of microservice projects (with plenty of deployable services and complex business logic) in the current open source community. To reduce this threat, we tried to choose projects of different sizes and domains. We also analyzed several versions of the projects, as well as all commits available in these versions. Nevertheless, it is undeniable that the empirical data used in the current study are not sufficient. We have to further extend the empirical evaluation on industrial microservices projects, to test if the results of our study can be applied to this types of projects.

Construct Validity: refers to the degree to which experimental variables accurately quantify the concepts they purport to measure.

In the dependent variables used to measure change impacts, the Directly Affected Scope (DAS) and Overall Affected Scope (OAS) of the number of modified files quantify the amount of change (Stevanetic and Zdun, 2018), and the Directly Affected Frequency (DAF) and Overall Affected Frequency (OAF) quantify the change-proneness of an artifact (Misirli et al., 2016) by measuring the frequency of revisions in which a file has changed. The indicators have been already studied and validated in the change impact analysis literature (Kretsou et al., 2021), and thus can be considered constructively valid.

6. Conclusion

This paper proposes a novel suite of metrics, *Microservice Coupling Index*, to measure the independence of individual microservices. Based on the relative measurement theory and Martin's principles, these metrics aim to provide a comparative gauge on to what extent the microservices in a system are dependent and coupled and a timely indicator of architecture refactoring for decoupling them.

We performed case studies using 15 open source projects, which has revealed that the relative coupling measures outperform the classical ones in each of the comparisons in discriminating high and low coupled microservices, and thus are more helpful in comparing design alternatives. We also collected four measures using a series of simulation-based experiments to indicate microservices' independence in terms of change impacts, and confirmed that in all comparisons the developed MCI metrics are more correlated with these measures than other baselines. The larger the MCIs, the less likely the changes can be localized and the individual microservices can be independently evolved. Our investigation suggests that MCIs have the potential to be valuable metrics for measuring, comparing, and improving microservices independence.

In the future, the experimental evaluation should be extended on industrial software systems in order to better assess the efficiency of the proposed MCI measures in the change impacts and independence analysis of microservices architecture. A second possible direction for improvement would be to consider a method level granularity for MCI, instead of a class level granularity, as in the current proposal. Furthermore, it would be interesting to investigate the possibility of applying the relative measurement theory to other aspects of microservices measurement, such as other types of coupling (e.g., dynamic, semantic, and logical coupling), style of communication (i.e. asynchronous) and the cohesion of microservices.

CRediT authorship contribution statement

Chenxing Zhong: Conceptualization, Methodology, Investigation, Software, Writing – original draft, Project administration. **He Zhang:** Conceptualization, Supervision, Writing – review & editing. **Chao Li:** Investigation, Data curation, Software, Writing – review & editing. **Huang Huang:** Methodology, Writing – review & editing, Visualization. **Daniel Feitosa:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to our data/code in the paper.

Acknowledgement

This work is jointly supported by the National Key Research and Development Program of China (No. 2019YFE0105500) and the Research Council of Norway (No. 309494), the Key Research and Development Program of Jiangsu Province (No. BE2021002-2), as well as the National Natural Science Foundation of China (No. 62072227, No. 62202219).

References

Abdi, H., Williams, L.J., 2010. Principal component analysis. Wiley Interdiscip. Rev. Comput. Stat. 2 (4), 433–459.
Al Dallal, J., 2010. Measuring the discriminative power of object-oriented class cohesion metrics. IEEE Trans. Softw. Eng. 37 (6), 788–804.
Al Dallal, J., Briand, L.C., 2012. A precise method-method interaction-based cohesion metric for object-oriented classes. ACM Trans. Softw. Eng. Methodol. 21 (2), 1–34.
Al-Debagy, O., Martinek, P., 2020. A metrics framework for evaluating microservices architecture designs. J. Web Eng. 19 (3–4), 341–370.
Alahmari, S., Zaluska, E., De Roure, D.C., 2011. A metrics framework for evaluating SOA service granularity. In: Proceedings of the 2011 IEEE International Conference on Services Computing. IEEE, pp. 512–519.
Alali, A., Bartman, B., Newman, C.D., Maletic, J.I., 2013. A preliminary investigation of using age and distance measures in the detection of evolutionary couplings. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE, pp. 169–172.
Allen, M.J., Yen, W.M., 2001. Introduction to Measurement Theory. Waveland Press.
Almugrin, S., Albattah, W., Melton, A., 2016. Using indirect coupling metrics to predict package maintainability and testability. J. Syst. Softw. 121, 298–310.
Arvanitou, E.-M., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P., 2015. Introducing a ripple effect measure: A theoretical and empirical validation. In: Proceedings of the 9th International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 1–10.
Assunção, W.K., Colanzi, T.E., Carvalho, L., Pereira, J.A., Garcia, A., de Lima, M.J., Lucena, C., 2021. A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study. In: Proceedings of the 28th International Conference on Software Analysis, Evolution and Reengineering. IEEE, pp. 377–387.
Athanasopoulos, D., Zarras, A.V., Miskos, G., Issarny, V., Vassiliadis, P., 2014. Cohesion-driven decomposition of service interfaces without access to source code. IEEE Trans. Serv. Comput. 8 (4), 550–562.
Auer, F., Lenarduzzi, V., Felderer, M., Taibi, D., 2021. From monolithic systems to microservices: An assessment framework. Inf. Softw. Technol. 137, 106600.
Baig, J.J.A., Mahmood, S., Alshayeb, M., Niazi, M., 2019. Package-level stability evaluation of object-oriented systems. Inf. Softw. Technol. 116, 106172.
Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A., 2013. An empirical study on the developers' perception of software coupling. In: Proceedings of the 35th International Conference on Software Engineering. IEEE, pp. 692–701.
Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A., 2019. Assuring the evolvability of microservices: Insights into industry practices and challenges. In: Proceedings of the 35th International Conference on Software Maintenance and Evolution. IEEE, pp. 546–556.
Briand, L.C., Morasca, S., Basili, V.R., 1999. Defining and validating measures for object-based high-level design. IEEE Trans. Softw. Eng. 25 (5), 722–743.
Carvalho, L., Garcia, A., Colanzi, T.E., Assunção, W.K.G., Pereira, J.A., Fonseca, B., Ribeiro, M., Lima, M.J.D., Lucena, C., 2020. On the performance and adoption of search-based microservice identification with toMicroservices. In: Proceedings of the 36th International Conference on Software Maintenance and Evolution. pp. 569–580.
Cazzola, W., Favalli, L., 2022. Towards a recipe for language decomposition: Quality assessment of language product lines. Empir. Softw. Eng. 27 (4), 82.
Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.

Cojocaru, M.-D., Uta, A., Oprescu, A.-M., 2019. Attributes assessing the quality of microservices automatically decomposed from monolithic applications. In: Proceedings of the 18th International Symposium on Parallel and Distributed Computing. IEEE, pp. 84–93.

Czibula, I.G., Czibula, G., Miholca, D.-L., Onet-Marian, Z., 2019. An aggregated coupling measure for the analysis of object-oriented software systems. J. Syst. Softw. 148, 1–20.

Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., El Fazziki, A., 2021. A multi-model based microservices identification approach. J. Syst. Archit. 118, 102200.

Di Francesco, P., Lago, P., Malavolta, I., 2019. Architecting with microservices: A systematic mapping study. J. Syst. Softw. 150, 77–97.

Engel, T., Langermeier, M., Bauer, B., Hofmann, A., 2018. Evaluation of microservice architectures: A metric and tool-based approach. In: Proceedings of the 30th International Conference on Advanced Information Systems Engineering. Springer, pp. 74–89.

Finch, A.P., Brazier, J.E., Mukuria, C., Bjorner, J.B., 2017. An exploratory study on using principal-component analysis and confirmatory factor analysis to identify bolt-on dimensions: The EQ-5D case study. Value Health 20 (10), 1362–1375.

Fregnan, E., Baum, T., Palomba, F., Bacchelli, A., 2019. A survey on software coupling relations and tools. Inf. Softw. Technol. 107, 159–178.

Fu, X., Cai, H., 2019. Measuring interprocess communications in distributed systems. In: Proceedings of the 27th International Conference on Program Comprehension. IEEE, pp. 323–334.

Galin, D., 2004. Software Quality Assurance: From Theory to Implementation. Pearson Education.

Halstead, M.H., 1977. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc..

Harrison, R., Counsell, S.J., Nithi, R.V., 1998. An evaluation of the MOOD set of object-oriented software metrics. IEEE Trans. Softw. Eng. 24 (6), 491–496.

Hauke, J., Kossowski, T., 2011. Comparison of values of Pearson's and Spearman's correlation coefficient on the same sets of data. Quaestiones Geographicae 30 (2), 87–93.

Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S., 2018. Microservices: The journey so far and challenges ahead. IEEE Softw. 35 (3), 24–35.

Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., Zheng, Q., 2019. Service candidate identification from monolithic systems based on execution traces. IEEE Trans. Softw. Eng. 47 (5), 987–1007.

Jin, W., Liu, T., Zheng, Q., Cui, D., Cai, Y., 2018. Functionality-oriented microservice extraction based on execution trace clustering. In: Proceedings of the 25th International Conference on Web Services. IEEE, pp. 211–218.

Kalia, A.K., Xiao, J., Krishna, R., Sinha, S., Vukovic, M., Banerjee, D., 2021. Mono2Micro: A practical and effective tool for decomposing monolithic Java applications to microservices. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1214–1224.

Kazemi, A., Azizkandi, A.N., Rostampour, A., Haghighi, H., Jamshidi, P., Shams, F., 2011. Measuring the conceptual coupling of services using latent semantic indexing. In: Proceedings of the 2011 IEEE International Conference on Services Computing. IEEE, pp. 504–511.

Kirbas, S., Sen, A., Caglayan, B., Bener, A., Mahmutogullari, R., 2014. The effect of evolutionary coupling on software defects: An industrial case study on a legacy system. In: Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement. pp. 1–7.

Kjaerulff, U.B., Madsen, A.L., 2008. Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis. Springer.

Klema, V., Laub, A., 1980. The singular value decomposition: Its computation and some applications. IEEE Trans. Automat. Control 25 (2), 164–176.

Kramer, S., Kaindl, H., 2004. Coupling and cohesion metrics for knowledge-based systems using frames and rules. ACM Trans. Softw. Eng. Methodol. 13 (3), 332–358.

Kretsou, M., Arvanitou, E.-M., Ampatzoglou, A., Deligiannis, I., Gerogiannis, V.C., 2021. Change impact analysis: A systematic mapping study. J. Syst. Softw. 174, 110892.

Lewis, J., Fowler, M., 2014. Microservices: A definition of this new architectural term. URL http://martinfowler.com/articles/microservices.html.

Li, B., Sun, X., Keung, J., 2013. FCA–CIA: An approach of using FCA to support cross-level change impact analysis for object oriented Java programs. Inf. Softw. Technol. 55 (8), 1437–1449.

Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., Gao, Q., Ge, J., Shan, Z., 2019. A dataflow-driven approach to identifying microservices from monolithic applications. J. Syst. Softw. 157, 110380.

Liu, S., Gao, C., Chen, S., Yiu, N.L., Liu, Y., 2020. ATOM: Commit message generation based on abstract syntax tree and hybrid ranking. IEEE Trans. Softw. Eng. 48.

Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall.

Martin, R.C., 2003. Agile Software Development: Principles, Patterns, and Practices. Prentice-Hall.

Mazlami, G., Cito, J., Leitner, P., 2017. Extraction of microservices from monolithic software architectures. In: Proceedings of the 24th International Conference on Web Services. IEEE, pp. 524–531.

McCabe, T.J., 1976. A complexity measure. IEEE Trans. Softw. Eng. SE-2 (4), 308–320.

Misirli, A.T., Shihab, E., Kamei, Y., 2016. Studying high impact fix-inducing changes. Empir. Softw. Eng. 21 (2), 605–641.

Mo, R., Cai, Y., Kazman, R., Xiao, L., Feng, Q., 2016. Decoupling level: A new metric for architectural maintenance complexity. In: Proceedings of the 38th International Conference on Software Engineering. IEEE, pp. 499–510.

Mo, R., Cai, Y., Kazman, R., Xiao, L., Feng, Q., 2019. Architecture anti-patterns: Automatically detectable violations of design principles. IEEE Trans. Softw. Eng. 47 (5), 1008–1028.

Newman, S., 2021. Building Microservices. O'Reilly Media.

Oman, P., Hagemeister, J., 1994. Construction and testing of polynomials predicting software maintainability. J. Syst. Softw. 24 (3), 251–266.

Papamichail, M.D., Symeonidis, A.L., 2020. A generic methodology for early identification of non-maintainable source code components through analysis of software releases. Inf. Softw. Technol. 118, 106218.

Perepletchikov, M., Ryan, C., 2010. A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software. IEEE Trans. Softw. Eng. 37 (4), 449–465.

Perepletchikov, M., Ryan, C., Frampton, K., Tari, Z., 2007. Coupling metrics for predicting maintainability in service-oriented designs. In: Proceedings of the 18th Australian Software Engineering Conference. IEEE, pp. 329–340.

Robert, C.P., 2007. The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation. Springer.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.E., et al., 1991. Object-Oriented Modeling and Design. Prentice-Hall.

Salama, M., Bahsoon, R., 2017. Analysing and modelling runtime architectural stability for self-adaptive software. J. Syst. Softw. 133, 95–112.

Santos, N., Silva, A.R., 2020. A complexity metric for microservices architecture migration. In: Proceedings of the 17th International Conference on Software Architecture. IEEE, pp. 169–178.

Sellami, K., Ouni, A., Saied, M.A., Bouktif, S., Mkaouer, M.W., 2022. Improving microservices extraction using evolutionary search. Inf. Softw. Technol. 151, 106996.

Selmadji, A., Seriai, A.-D., Bouziane, H.L., Mahamane, R.O., Zaragoza, P., Dony, C., 2020. From monolithic architecture style to microservice one based on a semi-automatic approach. In: Proceedings of the 17th International Conference on Software Architecture. IEEE, pp. 157–168.

Shao, J., 2003. Mathematical Statistics. Springer.

Siegmund, J., Siegmund, N., Apel, S., 2015. Views on internal and external validity in empirical software engineering. In: Proceedings of the 37th International Conference on Software Engineering. IEEE, pp. 9–19.

Spearman, C., 1961. The Proof and Measurement of Association Between Two Things. Appleton-Century-Crofts.

Stevanetic, S., Zdun, U., 2018. Supporting the analyzability of architectural component models - empirical findings and tool support. Empir. Softw. Eng. 23 (6), 3578–3625.

Taibi, D., Systä, K., 2019. A decomposition and metric-based evaluation framework for microservices. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science. Springer, pp. 133–149.

Tang, A., Nicholson, A., Jin, Y., Han, J., 2007. Using Bayesian belief networks for change impact analysis in architecture design. J. Syst. Softw. 80 (1), 127–148.

Valderas, P., Torres, V., Pelechano, V., 2020. A microservice composition approach based on the choreography of BPMN fragments. Inf. Softw. Technol. 127, 106370.

Vera-Rivera, F.H., Puerto, E., Astudillo, H., Gaona, C.M., 2021. Microservices backlog–A genetic programming technique for identification and evaluation of microservices from user stories. IEEE Access 9, 117178–117203.

Waseem, M., Liang, P., Shahin, M., Di Salle, A., Márquez, G., 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. J. Syst. Softw. 182, 111061.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer.

Xiao, L., Cai, Y., Kazman, R., Mo, R., Feng, Q., 2022. Detecting the locations and predicting the costs of compound architectural debts. IEEE Trans. Softw. Eng. 48 (9), 3686–3715.

Zdun, U., Queval, P.-J., Simhandl, G., Scandariato, R., Chakravarty, S., Jelic, M., Jovanovic, A., 2022. Microservice security metrics for secure communication, identity management, and observability. ACM Trans. Softw. Eng. Methodol. Just Accepted.

Zhong, C., Huang, H., Zhang, H., Li, S., 2022. Impacts, causes, and solutions of architectural smells in microservices: An industrial investigation. Softw. - Pract. Exp. 52 (12), 2574–2597.