Measuring Coupling in Microservices Using COSMIC Measurement Method

Roberto Pedraza-Coello
Research Institute in Applied Mathematics and Systems
National Autonomous University of Mexico
CDMX, Mexico City, Mexico
Email: rpedrazacoello@gmail.com

Francisco Valdés-Souto
Science Faculty
National Autonomous University of Mexico
CDMX, Mexico City, Mexico
Email: fvaldes@ciencias.unam.mx
ORCID: 0000-0001-6736-0666

***Abstract*—The Microservices Architectural Style is one of the latest trends in software development companies. Having highly coupled microservices can lead to latency and network traffic, high interdependency between development teams, among other problems. Being able to measure the coupling between microservices in early phases of the software development life cycle could help the software architects make better decisions when designing. This paper proposes a way of measuring coupling between microservices. This metric is based on the COSMIC measurement method (ISO/IEC 19761). The paper also shows a practical implementation of this metric.**

Keywords-microservices; coupling; measurement; COSMIC; ISO/IEC - 19761.

I. INTRODUCTION

The Microservices Architectural Style (MAS) is one of the latest trends in software development companies. Its main idea is to develop an application as a set of small services. Each one of these services is called a microservice. It is an approach to software and systems architecture that builds on the concept of modularization but emphasizes technical boundaries [13].

Each microservice is implemented and operated as a small and independent system. It offers access to its internal functionality and data through a well-defined network interface. MAS increases the software development process agility because each microservices is an independent unit of development, deployment, operations, versioning, and scaling [13].

The MAS benefits caused companies, including worldwide companies, to migrate their software to this architecture style. However, MAS is not a silver bullet, and it has several challenges in the software development lifecycle phases.

The microservices of an application are interconnected between them to perform the functionality. This intercommunication could imply some coupling between the microservices. Coupling is referred to as the interdependency that exists between different objects. If the microservices depend a lot on each other, then the coupling is high. If the microservices depend little or none on each other, the coupling is low.

One of the MAS problems found in literature is the increment of consumption of network resources. This problem is partially caused because of the coupling that exists between microservices when achieving a particular functionality. Low-level coupling between microservices makes sense to reduce the consumption of network resources. Soldani [15] mentions that, since the microservices in an application intercommunicate through remote API invocations, applications generate higher network traffic with respect to monoliths (where modules interact through memory calls) or service-based applications (composed by a lower number of services, hence reducing the number of remote API Invocations).

Measuring the coupling between microservices in the early stages of the software development cycle could help to quantify the interdependency that exists between different microservices, improving the software architect’s decision making in terms of avoiding high interdependency between teams, or high network-traffic areas. Coupling metrics have been proposed over the years. For example, Chidamber and Kemerer [7] have proposed a metric called *Coupling Between Object classes (CBO).* The CBO of a class is the aggregation of the number of other classes to which it is coupled. It is mentioned that inter-class coupling occurs when methods of one class use methods or instance variables of another class. However, it is never mentioned how to count the usage of methods or instances of another class. It could be counted once by every occurrence in each method, once by every class-type object in each method, etc. The measurement procedure is not clear. Additionally, the use of scales it is not defined. Then, according to the metrology concept “Measurement Foundation” presented by Abran [2], it is not possible to evaluate the validity of this metric.

Other metric called *Weighted Methods Per Class* is proposed in [7]. The idea is to do an aggregation of the complexity of the methods of a class. in [7] the author mentions that “Complexity” is not defined more specifically to allow for the most general application of this metric. The lack of an explicit definition of complexity can result in a same class having very different result measurement values. It can be affirmed that the measurements obtained with this metric are not comparable, which is not good from a metrology point of view, as mentioned by Abran [2].

An additional metric is proposed by Chidamber [7]. It is called *Lack of Cohesion in Methods (LCOM).* It is the sum of the number of method pairs in a class whose similarity is zero (not similar) minus the count of method whose similarity is not zero (exists some similarity). This means, the lower the measure value the greater the cohesion. The lowest possible LCOM value is zero. The paper mentions that even when LCOM is equal to zero this does not imply maximal cohesiveness, since within the set of classes with LCOM = 0, some may be more cohesive than others or, in other words, some may lack of more cohesion than others. This is a problem, even though multiple classes can have LCOM = 0 some of these classes lack more cohesion than others, therefore we can affirm that this metric is not comparable.

Currently, the only type of software measurement with international standards adopted by the ISO is the measurement of functional size. It is also the only type of software measurement that has a method that complies with the metrology requirements [2]. Up to now, there are five standards of software Functional Size Measurement Methods (FSMM). Of those five standards, the ISO/IEC 19761 COSMIC method is the only standard belonging to the second generation, including several use domains, like Management Information Systems (MIS), real-time infrastructure, Etc. It also solves most of the problems with the FSMM of the 1st generation [16].

This paper presents an approach to measuring coupling between microservices. The coupling metric is based completely on the standard ISO/IEC 197611, the COSMIC method. This approach is proposed to define an objective metric that can improve the knowledge about the coupling between microservices in order to provide to software architects quantitative elements, based in an international standard, to take decisions.

The paper is organized as follows: Section 2 explains the background about microservices and the context about COSMIC. Section 3 presents related work on coupling measurement methods. Section 4 describe the proposed coupling measurement method based on the COSMIC standard, including an example of its application. Section 5 presents the conclusions of the paper and future work.

II. BACKGROUND

This section presents the background of microservices and the COSMIC measurement method.

A. *Microservices*

The Netflix company, like other companies, had a problem a few years ago. They had a monolithic web system that was modified by multiple people every day. The software, with its updates, was deployed once or twice a week. If one of the changes caused a problem, it was hard and time-consuming to diagnose a cause. When a company is trying to compete in the agile environment where the updates must be delivered to the consumer as quick as possible this situation can cause many internal and commercial troubles. Because of these troubles and a few more, Netflix decides to migrate its business software to MAS.

The most repeated MAS definition in literature is from Fowler's and Lewi's blog, where they define that “The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies [10].”

To better understand this architectural style, it is useful to compare it with the monolithic style. For example, a software that follows a client-server architecture usually consists of three parts: A client-side application, a database, and a server-side application. This server-side application is a monolith, which means a single executable unit. Every change to the server-side app implies building and deploying a new version of the app. However, with MAS, each microservice is implemented and operated as a small and independent system. The microservice offers access to its internal functionality and data via a network interface. This improves the agility of the development process, because every microservices becomes an independent unit of development, deployment, operation, versioning and scaling [10].

The MAS general idea is to develop an application as a set of interconnected services. This interconnection generates a certain coupling between the microservices. It is recognized that if the coupling is high, then technological and management problems can arise.

In most of programming paradigms the software quality characteristics defined as low coupling and high cohesion are ideal. For example, in the Object-Oriented Paradigm a software with low coupling is achieved when each object depends on little or nothing of other objects. The same idea of low coupling applies to MAS. High coupling between microservices could cause latency and network traffic, high interdependency between development teams, among other problems [9][17].

Currently the evaluation of coupling is made with subjective methods, and then there is a need to measure coupling between microservices in a formal and standardized way. With good measures, several problems can be identified and characterized, and decisions can be taken.

B. *COSMIC*

Several ISO/IEC standards have been developed oriented to measure the software functionality in the software engineering field. The ISO/IEC 14143 standard [12] includes a set of rules regarding size measurement in functionality units. For this type of measurement, the standard proposes the following definitions:

“Functional size is defined as the size of the software derived by quantifying the Functional User Requirements” [12]

“Functional User Requirements (FUR) stands for a sub-set of the User Requirements describing what the software does, in terms of tasks and services” [12]

“Functional Size Measurement (FSM) is the process of measuring functional size” [12]

The COSMIC measurement method has been acknowledged by the ISO/IEC as conforming to the rules laid down in the ISO/IEC 14143 and has taken the form of the standard ISO/IEC 19761. There are others FSM that have taken the form of ISO/IEC standards. However, COSMIC is the only one that belongs to the “Second Generation” of FSM methods. The other FSMs belong to the First Generation [3].

COSMIC introduces its own homologated and standardized measure unit called Cosmic Function Point (CFP). 1 CFP represents the size of one data movement (Entry, eXit, Read or Write). Therefore, functional size can be measured by counting the number of data movements. More information about COSMIC and different guidelines to apply COSMIC can be found on the COSMIC website [1].

III. RELATED WORK

Allen and Khoshgoftaar [5] present a way of measuring Coupling and Cohesion in a module-based software. The coupling measurement method starts with a measurement protocol that results in a graph-abstraction representing some aspect of software design. For example, class inheritance, class type, method invocation, class-attribute references. Different abstractions, for a same software, can result in different measures. The metric is based in the software abstracted as a graph and separating the graph into modules. An issue observed for this method is that it is based on a software abstraction generated by humans. Humans have different points of view when abstracting software, and there are no right or wrong abstractions. So, one same software can have multiple abstractions, and each abstraction can have a different coupling measurement, what is not considered correct.

Arisholm et al. [6] propose three different approaches to measure the strength of a coupling relation: number of messages, number of distinct method invocations, and distinct classes. The number of messages refers to the number of different messages that are exchanged between two entities. The other two represent the number of methods called, and classes used by a method in an object. An issue observed in this paper is that there are no standard metric units for messages, method invocations and classes. Also, as the measurement is done at runtime, the measurement can vary a lot depending on when the measurement is being done. So, comparing the coupling of two software becomes problematic. They should be compared at equal conditions for the comparison to be valid. It is not defined how to achieve equal conditions. From a metrology point of view, the measurements should be comparable.

These same situations are observed in Lavazza et al [14]. They propose a theoretical framework based on Axiomatic Approaches for the definition of dynamic software measures. This paper also presents measures based on this framework. These are defined for dynamically quantifying coupling. The coupling measurements are based on counting, at runtime: the number of distinct methods invoked by each method in each object, the count of the total number of distinct messages sent from one object class to other objects, and the count of the distinct number of classes that a method uses. Once again, the measurement obtained for one software is not comparable with the measurement obtained for another software. For instance, the messages send from one object o other object could consider distinct entities, or domain object information, in the same message.

Hassoun et al. [11] propose a relation called DCM (Dynamic Coupling Metric) to formalize the idea of dynamic coupling. That metric works at the object level. It is mentioned that measuring object coupling gives an insight into the system structure and allows the comparison of architectural aspects of a different system relative to reuse and maintenance. This paper uses a complexity measure in its' formulas. However, it is not mentioned how to calculate the complexity nor what complexity means for the context of the paper.

IV. COUPLING METRIC BASED IN COSMIC

One of the main differences between some of the metrics that are usually used to measure software and the COSMIC method is that the COSMIC method complies with the 3 metrology concepts, mentioned by Abran [2] for a “good” design of a software measurement method: Measurement foundation, Quantities and units, and Measurement Standards-Etalons. In one or more of these concepts is where popular software metrics like Function Points, Use Case points, Cyclomatic Complexity, Quality Models, among others fail.

The proposal is to use the concepts of the COSMIC measurement method to measure the coupling between microservices to ensure that, when the coupling between two microservices is measured, the measurement is consistent, repeatable, and comparable. A good measurement method is independent of the person measuring and the measurement environment.

For this paper, microservices coupling refers to the dependency that exists from one microservice MS1 to another microservice MS2. Whenever MS1 makes an HTTP request (or through another protocol) to MS2, it is because MS1 needs to send messages (eXit) or receive messages (Entry) from MS2. In this sense, it is understood that there is a unilateral or a bilateral coupling. By using the COSMIC concepts [8] it can be said that a relationship between two microservices is defined by a correspondence rule that can be hierarchical (exclusively one service uses the services of another), or bidirectional (both services use services of the other service).

For the proposed metric, when said that MS1 is coupled to MS2, it is meant that MS1 depends on MS2 to complete a certain portion of its own functionality. However, it does not necessarily mean the same in inverse mode.

It can be said that MS1 is coupled to MS2 when MS1 starts a request/response communication with MS2. A good analogy is to imagine a client-server architecture where MS1 is the client and MS2 is the server, the client depends on the server, not the other way around.

Keeping the last example, to measure how coupled is MS1 to MS2 we need to count, for each MS1 functional process, the number of data movements that are exchanged between MS1 and MS2 for all the cases where MS1 starts the communication with MS2.

The proposed metric is based on determining the degree of coupling of a particular microservice based on COSMIC method concepts, considering the defined scope of the measurement.

The coupling concept is approached this way because, usually, the microservices offer their services via an HTTP API [10]. These APIs allow the microservices to offer their services to multiple clients. Following the previous example, MS1 is a client of MS2. However, MS2 could have 1000 more clients. It is considered that it does not make sense to think that MS2 is coupled to 1000 clients just because the 1000 clients use MS2's services. It makes even less sense if, from an MS2 perspective, it does not matter what client is using the services.

The COSMIC measurement manual [8] explains how to measure software by counting the data movements in each of the functional processes. There are certain rules of when a certain data movement is considered for the measurement and when not. The coupling metric proposed in this article is based on using the same rules that COSMIC uses and applying them to the metric's context. The coupling measurement between a microservice MS1 and a microservice MS2 can be calculated by counting the Entry and eXit data movements done in each of the functional processes of MS1 when those data movements move data from/to MS2. Also, MS1 must start the communication with MS2 during the functional process that it is being measured.

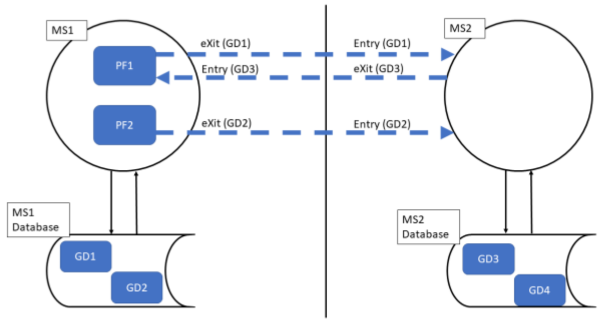


Figure 1. Simple example of MS1 coupled to MS2

For example, Figure 1 shows that functional process 1 (PF1) needs one eXit data movement and one Entry data movement from MS2 to complete its functionality. This means a total of 2 data movements in PF1 from MS1 to MS2. It also can be observed that functional process 2 (PF2) needs to send (eXit) one data group to MS2 to complete its functionality. By adding up the data movements from their two functional processes, the measurement's result is that the coupling level from MS1 to MS2 is 3.

Measuring the coupling between microservices allows one to obtain an objective value of the dependency from one microservice to another microservice. If the dependency is low, then the coupling between microservices is also low, independently how many instances of MS1 or MS2 are generated, the coupling level is measure of dependency between services, not about instances at execution.

V. APPLYING THE METRIC

This section shows an example on how to apply the proposed metric. The example is based on a case study called C-Reg [4]. The case study can be found in the COSMIC web application [1]. The case study shows the whole process of measuring functional size.

To the best of our knowledge, this case study was not thought as MAS software. However, there is communication between the measured software and other pieces of software. This paper assumes that three software pieces mentioned in the case study were built as microservices. This premise does not affect the COSMIC measurement nor the Coupling measurement.

As shown in Figure 2, the C-Reg application has multiple functional users. Some of these functional users are humans and other functional users are software. For this paper, we can ignore human functional users and focus on software functional users.

The C-Reg app [4] counts with 19 functional processes, from which 11 do at least one data movement between C-Reg and one or more software functional users. The rest of the functional processes only communicate with human functional users, so they fall out of the context of this paper. Table I shows the names of the 19 functional processes,) the ones with at least one data movement between C-Reg and external software are marked in green, the external software is considered a functional user (Billing System, Course Catalog System). See Figure 2.

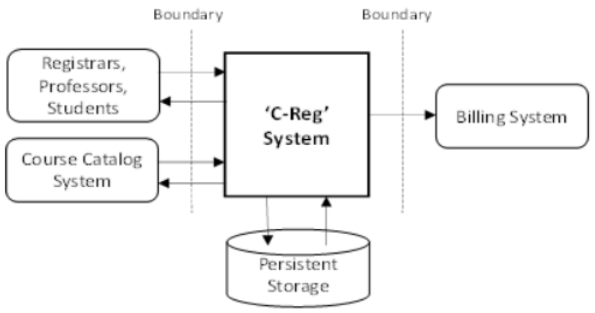


Figure 2. C-Reg Application Context Diagram. Obtained from [4]

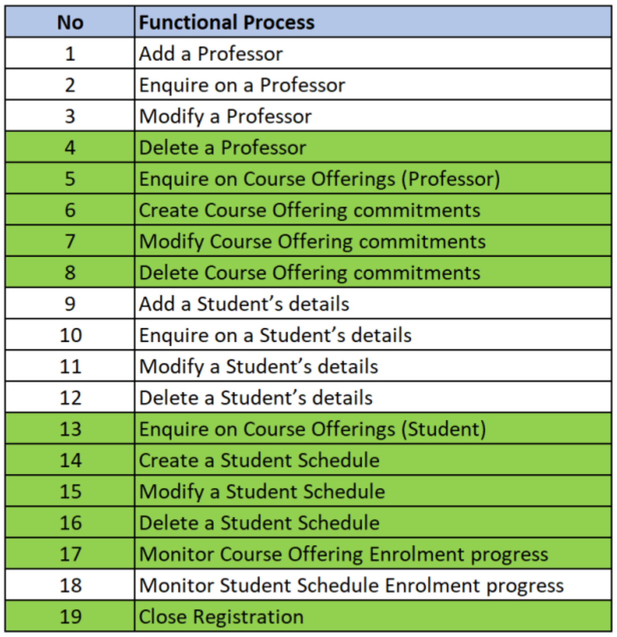
Table II and Table III show the detail of 2 of the functional processes to explain how the coupling metric can be applied practically. First, the functional process called "Delete a professor" is presented in Table II. It can be observed that there are 2 data movements between C-Reg and Course Catalog. These 2 data movements are considered with the rest of the data movements between C-Reg and Course Catalog to measure how coupled is C-Reg to the course catalog.

The following functional process that is presented is called "Close Registration". The details of this functional process can be observed in Table III. The table shows that there are 3 data movements between C-Reg and Course Catalog. It also shows that there is one data movement between C-Reg and Billing System. These two results will be considered when calculating how coupled C-Reg is to Course Catalog, and how coupled C-Reg is to Billing System.

By analyzing the tables of each of the functional processes in [4] and applying the proposed coupling metric, we obtain the results presented in Table IV. It can be observed that C-Reg has a level of coupling of 21 CFP with Course Catalog, including 21 data movements between C-Reg and Course Catalog. It also can be observed that C-Reg has a coupling level of 1 CFP with Billing System.

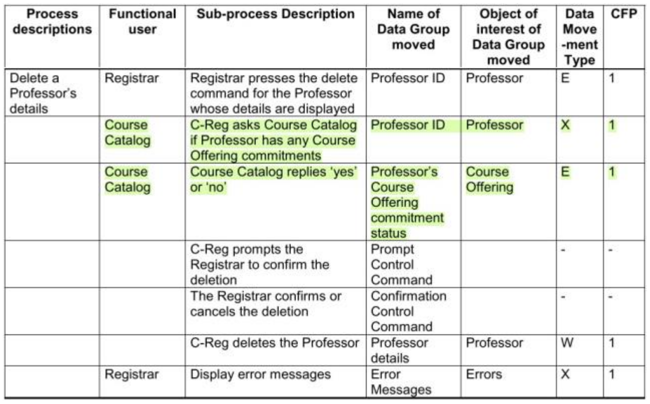
It is easy to observe that the coupling from C-Reg with Billing System is 1 CFP, and with Course catalog System the coupling is 21 CFP, so there is 21 times more coupling with Course catalog Systema than Billing System.

TABLE I. C-REG'S FUNCTIONAL PROCESSES



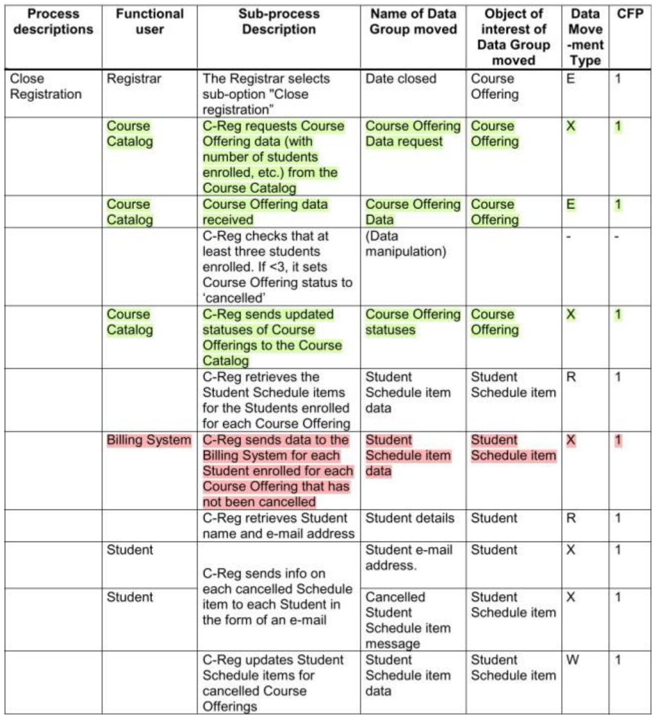
No	Functional Process
1	Add a Professor
2	Enquire on a Professor
3	Modify a Professor
4	Delete a Professor
5	Enquire on Course Offerings (Professor)
6	Create Course Offering commitments
7	Modify Course Offering commitments
8	Delete Course Offering commitments
9	Add a Student's details
10	Enquire on a Student's details
11	Modify a Student's details
12	Delete a Student's details
13	Enquire on Course Offerings (Student)
14	Create a Student Schedule
15	Modify a Student Schedule
16	Delete a Student Schedule
17	Monitor Course Offering Enrolment progress
18	Monitor Student Schedule Enrolment progress
19	Close Registration

TABLE II. FUNCTIONAL PROCESS "DELETE PROFESSOR" DETAILS. MARKED IN GREEN THE SUBPROCESSES OF COMMUNICATION BETWEEN C-REG AND COURSE CATALOG. ADAPTED FROM [4]



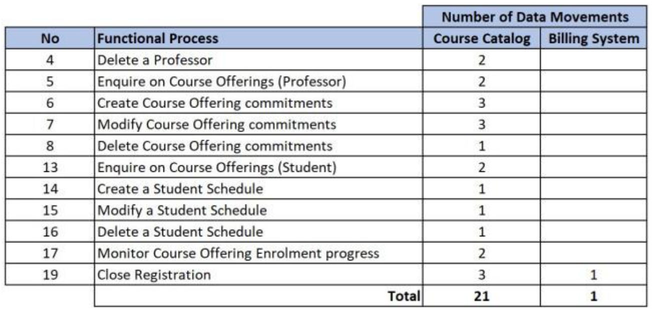
Process descriptions	Functional user	Sub-process Description	Name of Data Group moved	Object of interest of Data Group moved	Data Move -ment Type	CFP
Delete a Professor's details	Registrar	Registrar presses the delete command for the Professor whose details are displayed	Professor ID	Professor	E	1
	Course Catalog	C-Reg asks Course Catalog if Professor has any Course Offering commitments	Professor ID	Professor	X	1
	Course Catalog	Course Catalog replies 'yes' or 'no'	Professor's Course Offering commitment status	Course Offering	E	1
		C-Reg prompts the Registrar to confirm the deletion	Prompt Control Command		-	-
		The Registrar confirms or cancels the deletion	Confirmation Control Command		-	-
		C-Reg deletes the Professor	Professor details	Professor	W	1
	Registrar	Display error messages	Error Messages	Errors	X	1

TABLE III. FUNCTIONAL PROCESS "CLOSE REGISTRATION" DETAILS. MARKED IN GREEN THE SUBPROCESSES OF COMMUNICATION BETWEEN C-REG AND COURSE CATALOG. MARKED IN RED THE SUBPROCESSES OF COMMUNICATION BETWEN C-REG AND BILLING SYSTEM. ADAPTED FROM [4]



Process descriptions	Functional user	Sub-process Description	Name of Data Group moved	Object of interest of Data Group moved	Data Move -ment Type	CFP
Close Registration	Registrar	The Registrar selects sub-option "Close registration"	Date closed	Course Offering	E	1
	Course Catalog	C-Reg requests Course Offering data (with number of students enrolled, etc.) from the Course Catalog	Course Offering Data request	Course Offering	X	1
	Course Catalog	Course Offering data received	Course Offering Data	Course Offering	E	1
		C-Reg checks that at least three students enrolled. If <3, it sets Course Offering status to 'cancelled'	(Data manipulation)		-	-
	Course Catalog	C-Reg sends updated statuses of Course Offerings to the Course Catalog	Course Offering statuses	Course Offering	X	1
		C-Reg retrieves the Student Schedule items for the Students enrolled for each Course Offering	Student Schedule item data	Student Schedule item	R	1
	Billing System	C-Reg sends data to the Billing System for each Student enrolled for each Course Offering that has not been cancelled	Student Schedule item data	Student Schedule item	X	1
		C-Reg retrieves Student name and e-mail address	Student details	Student	R	1
	Student	C-Reg sends info on each cancelled Schedule item to each Student in the form of an e-mail	Student e-mail address.	Student	X	1
	Student		Cancelled Student Schedule item message	Student Schedule item	X	1
		C-Reg updates Student Schedule items for cancelled Course Offerings	Student Schedule item data	Student Schedule item	W	1

TABLE IV. COUPLING MEASUREMENT VALUES FOR C-REG CASE STUDY



		Number of Data Movements	
No	Functional Process	Course Catalog	Billing System
4	Delete a Professor	2	
5	Enquire on Course Offerings (Professor)	2	
6	Create Course Offering commitments	3	
7	Modify Course Offering commitments	3	
8	Delete Course Offering commitments	1	
13	Enquire on Course Offerings (Student)	2	
14	Create a Student Schedule	1	
15	Modify a Student Schedule	1	
16	Delete a Student Schedule	1	
17	Monitor Course Offering Enrolment progress	2	
19	Close Registration	3	1
	Total	**21**	**1**

VI. COUPLING METRICS ANALYSIS

It can be observed in Table V the main differences between the related work and the prosed metric based on COSMIC. Five columns are presented:

- International Standard: Is it based on an International Standard?
- Metrology Requirements: Does it comply with the metrology concepts mentioned by Abran [2]?
- Comparable: Is it valid to compare the measurement results of different software?
- Proved on MAS: Is there a case study where the metric (or an adaptation of it) was used to measure coupling between microservices?

Following with Table V, possible answers to these questions are:

- Yes
- No
- SP: Yes, if and only if the same procedure was used to measure the software
- EC: Yes, if and only if, somehow, equal conditions between the software is achieved.
- NF: No references found

TABLE V. COUPLING METRICS ANALYSIS

Coupling Measurement	International Standard	Metrology Requirements	Comparable	Proved on MAS
Coupling with COSMIC	Yes	Yes	Yes	Yes
Allen and Khoshgoftaar [5]	No	No	No	NF
Arisholm et al. [6]	No	No	SP and EC	NF
Coupling Between Object Classes [7]	No	No	SP	No
Lavazza et al [14]	No	No	SP and EC	NF
Hassoun et al [11]	No	No	No	NF

VII. CONCLUSION

Many companies are developing software based on MAS because of the multiple benefits that come with it. However, MAS is not a silver bullet. Developers face multiple challenges when developing software based on MAS. Some problems could be generated because of the coupling that exists between microservices when achieving a particular functionality, then low-level coupling between microservices could avoid high interdependency between teams, or high network-traffic areas reducing the consumption of network resources, for instance.

When two microservices communicate a lot with each other, it can be said that these two are highly coupled. Finding highly coupled microservices in the design phase of the software development life cycle could lead a software architect to make better decisions about the software design.

In this paper, we propose a way of measuring coupling between microservices. This metric is based on the COSMIC measurement standard to ensure that the measurement obtained is consistent, repeatable, and comparable when the coupling between microservices is measured. The paper also shows a practical example of how to measure coupling between microservices with the proposed metric.

It is observed from the results (Table IV) that the results make sense with the reality that represent the C-Reg system. The C-Reg software is coupled to two external functional users software: Billing System and Course Catalog System. The coupling measurement of C-Reg to: the Billing System is 1 CFP, and for the Course Catalog System the coupling is 21 CFP. There is 21 times more coupling with Course catalog System than with Billing System.

In comparison with the other coupling metrics presented in this paper, the proposed metric complies better with the metrology concepts of a good measurement method. The main advantage of this metric is that it is based on an International Standard.

A. Future Work

There can be situations where low-coupled microservices are generating a lot of network traffic, and high-coupled microservices are generating little network traffic. For example, MS1 and MS2, two low-coupled microservices, could generate a lot of network traffic if they include high-usage functionality. Other example is MS3 and MS4, two highly-coupled microservices, that could generate little network traffic if they include low-usage functionality. It could be interesting to look at the correlation that exists between the coupling measurement and network traffic in different kinds of MAS software systems.

Low coupling is a software quality characteristic in all software, not only on microservices, and it could be interesting to find a way to adapt this proposed metric to measure coupling in all kinds of software, not only the ones based on MAS.

It could be interesting to do a comparison of how reliable other coupling metrics against the metric are proposed in this paper. Considering that a good measurement method is independent of the person measuring and the measurement environment. The measurement results must be consistent, repeatable, and comparable

REFERENCES

[1] COSMIC Sizing - The open standard for software size measurement.

[2] A. Abran.Software Metrics and Software Metrology. 2010.

[3] A. Abran and C. Woodward. Guideline on how to convert ‘FirstGeneration’ Function Point sizes to COSMIC sizes. (November):0–53,2016.

[4] A. Lesterius, A. Abran, C. Symmons. Course Registration (‘C - REG ’) System Case Study. (December):1–43, 2015.

[5] E. B. Allen, T. M. Khoshgoftaar, and Y. Chen. Measuring couplingand cohesion of software modules: An information-theory approach.International Software Metrics Symposium, Proceedings, pages 124–134, 2001.

[6] E. Arisholm, L, C Briand, and A. Føyen. Dynamic coupling measurement for object-oriented software. IEEE Transactions on Soft-ware Engineering, 30(8):491–506, 2004.

[7] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering,20(6):476–493, 1994.

[8] Common Software Measurement International Consortium (COSMIC).Measurementt Manual v4.0.2. (December):1–115, 2017.

[9] S. S. de Toledo, A. Martini, and Dag I.K. Sjøberg. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. Journal of Systems and Software, 177:110968,2021.

[10] M. Fowler and J. Lewis. Microservices - A definition of this new architectural term, 2014.

[11] Y. Hassoun, R. Johnson, and S. Counsell. A dynamic runtime coupling metric for meta-level architectures. Proceedings of the European Conference on Software Maintenance and Reengineering,CSMR, 8:339–346, 2004.

[12] ISO; IEC. Information technology — Software measurement — Functional size measurement — Part 6: Guide for use of ISO/IEC 14143series and related International Standards (ISO/IEC 14143-6:2006(E)),2006.

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

[14] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. On the definition of dynamic software measures. International Symposium on Empirical Software Engineering and Measurement, pages 39–48, 2012.

[15] J. Soldani, D. A. Tamburri, and W. J. Van DenHeuvel. The pains and gains of microservices: A Systematic grey literature review.Journal of Systems and Software, 146:215–232, 2018.

[16] F. Valdés-Souto, R. Pedraza-Coello, and F. C. Olguín-Barrón. COSMIC sizing of RPA software: A case study froma proof of concept implementation in a banking organization. CEURWorkshop Proceedings, 2725:1–15, 2020.

[17] R. Subramanyam and M. S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on Software Engineering, 29(4):297–310,2003.