UNIVERSIDAD DEL VALLE
ESCUELA DE INGENIERÍA DE SISTEMA Y COMPUTACIÓN
DOCTORADO EN INGENIERÍA

TESIS DE INVESTIGACIÓN DOCTORAL

TÍTULO

MODELO INTELIGENTE PARA ESPECIFICAR DEPENDENCIA Y COMPLEJIDAD EN EL DISEÑO DE APLICACIONES BASADAS EN MICROSERVICIOS.

AUTORA
VERÓNICA DEL CONSUELO TAPIA CERDA, MSc.
UNIVERSIDAD TÉCNICA DE COTOPAXI

DIRECTOR
CARLOS MAURICIO GAONA CUEVAS, Ph.D.
UNIVERSIDAD DEL VALLE

CALI - COLOMBIA

2025

TESIS DE INVESTIGACIÓN DOCTORAL

TÍTULO:

MODELO INTELIGENTE PARA ESPECIFICAR DEPENDENCIA Y COMPLEJIDAD EN EL DISEÑO DE APLICACIONES BASADAS EN MICROSERVICIOS.

DIRECTOR:

CARLOS MAURICIO GAONA CUEVAS, PhD.

PROFESOR ESCUELA DE INGENIERIAS DE SISTEMAS Y COMPUTACIÓN

UNIVERSIDAD DEL VALLE

AUTORA:

VERÓNICA DEL CONSUELO TAPIA CERDA, MSc.

UNIVERSIDAD TÉCNICA DE COTOPAXI

GRUPOS DE INVESTIGACIÓN:

GRUPO DE ESTUDIOS DOCTORALES EN INFORMÁTICA - GEDI.

UNIVERSIDAD DEL VALLE.

GRUPO DE INVESTIGACIÓN DESARROLLO TECNOLÓGICO PARA SISTEMAS DE INFORMACIÓN AUTOMATIZADOS

UNIVERSIDAD TÉCNICA DE COTOPAXI

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

RESUMEN

La tesis aborda la creación de un modelo inteligente para evaluar dependencia y complejidad en arquitecturas basadas en microservicios, denominado MOAM (Microservices Optimization Architectures Model). Este modelo combina técnicas avanzadas de Machine Learning (ML) y Procesamiento de Lenguaje Natural (NLP) para optimizar decisiones arquitectónicas desde las etapas tempranas del desarrollo.

Las principales motivaciones fueron abordar desafíos de evaluación de arquitecturas de microservicios y el limitado número de herramientas de evaluación y optimización de estas arquitecturas.

Todos los objetivos se cumplieron porque MOAM es un modelo inteligente que especifica dependencia y complejidad en el diseño arquitectónico de aplicaciones basadas en microservicios, apoyado en técnicas de ML y NLP.

La metodología utilizada se basó en un enfoque iterativo que integró desde la revisión sistemática de literatura, la creación del modelo que analiza descripciones arquitectónicas, despliegue de resultados y la comprobación de validez mediante casos de estudio.

MOAM demostró ser una herramienta eficaz para evaluar arquitecturas de microservicios. Los principales resultados son métricas de evaluación, optimización arquitectónica porque identifica áreas problemáticas y genera recomendaciones de mejora. Contribuye a optimizar la forma de diseñar y evaluar arquitecturas por su capacidad para analizar y generar resultados automatizados.

Se identificaron ciertas limitaciones relacionadas con la escalabilidad del modelo en sistemas extremadamente grandes, se recomienda continuar la investigación para optimizar su aplicabilidad.

Además de MOAM, el trabajo aporta un conjunto de métricas, un modelo matemático para evaluar dependencia y complejidad y el uso de técnicas de NLP en el diseño de arquitecturas de microservicios.

Palabras clave: Calidad de microservicios, Arquitectura, Machine Learning, Procesamiento de Lenguaje Natural.

ABSTRACT

The thesis presents the development of an intelligent model for evaluating dependency and complexity in microservices-based architectures, known as MOAM (Microservices Optimization Architectures Model). This model integrates advanced Machine Learning (ML) and Natural Language Processing (NLP) techniques to optimize architectural decision-making from the early stages of development.

The main motivations behind this research were to address the challenges in evaluating microservices architectures and the limited availability of assessment and optimization tools for these architectures.

All research objectives were successfully met, as MOAM provides an intelligent framework that specifies dependency and complexity in the architectural design of microservices-based applications, leveraging ML and NLP techniques.

The methodology followed an iterative approach, incorporating a systematic literature review, the model development process, analysis of architectural descriptions, result deployment, and validation through case studies.

MOAM has proven to be an effective tool for assessing microservices architectures. The key outcomes include evaluation metrics, architectural optimization by identifying problematic areas, and improvement recommendations. Its ability to analyze and generate automated results contributes to optimizing the design and evaluation of microservices architectures.

Certain limitations were identified, particularly regarding scalability in extremely large systems. Further research is recommended to enhance its applicability.

Beyond MOAM, this work contributes with a set of metrics, a mathematical model for evaluating dependency and complexity, and the integration of NLP techniques into microservices architecture design.

Keywords: Quality of microservices, Architecture, Machine Learning, Natural Language Processing.

CAPÍTULO 1

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

En las últimas décadas, las arquitecturas basadas en microservicios han emergido como un paradigma esencial para el desarrollo de aplicaciones distribuidas, especialmente en contextos de alta demanda tecnológica como el comercio electrónico, los servicios financieros y la computación en la nube. Este enfoque arquitectónico, caracterizado por su modularidad y escalabilidad, ha demostrado ser una solución eficaz para superar las limitaciones de las arquitecturas monolíticas tradicionales. Sin embargo, a pesar de los avances significativos, las arquitecturas de microservicios presentan desafíos relacionados con atributos y patrones de calidad que aún no están claros [1], sobre todo, interesan a esta investigación dos aspectos que afectan profundamente la calidad, sostenibilidad y capacidad evolutiva de los sistemas, como son, dependencia y complejidad.

El fundamento del concepto de un microservicio es hacer cosas pequeñas para reducir la complejidad. En una arquitectura de microservicios, varios servicios interactúan unos con otros, cada uno con un propósito específico para el intercambio de mensajes y el aprovisionamiento de servicios web; generalmente se apalanca en un modelo de interacción asincrónica que aumenta la escalabilidad [2][3]. Así mismo, uno de los principales problemas identificados es que las dependencias rígidas entre servicios dificultan la implementación de

cambios, incrementan los riesgos de propagación de fallos y comprometen la escalabilidad del sistema. Esta complejidad arquitectónica, derivada de la naturaleza distribuida de los microservicios y puede ocasionar aumento de los costos de mantenimiento, errores difíciles de diagnosticar y reducción de la agilidad para responder a las necesidades del mercado, por tanto, se requiere tener en cuenta principios de diseño establecidos durante la creación y evolución del software, no hacerlo, puede conducir a la llamada erosión arquitectónica, que terminaría en una condición de mantenimiento inviable [4].

Lamentablemente, a pesar de la creciente adopción de microservicios en la industria, existen pocas iniciativas para el tratamiento de los desafíos antes mencionados, es decir, es necesario incorporar procesos y herramientas de seguimiento, desde la concepción y posterior evolución de los proyectos, para evaluar las características arquitectónicas de los mismas y poder contribuir a la optimización de la calidad desde las fases tempranas del desarrollo. Existen estrategias tradicionales de evaluación de la calidad de microservicios, éstas tienden a centrarse en el análisis del código o en mediciones posteriores a la implementación, lo que limita la capacidad de los arquitectos para anticipar y mitigar problemas arquitectónicos desde el inicio, trayendo como consecuencias, no solo el incremento de costos asociados al diseño y mantenimiento, sino que también pone en riesgo la sostenibilidad del sistema en contextos de alta demanda y evolución constante [5].

Abordar esta problemática es especialmente relevante en un entorno tecnológico donde los sistemas deben ser escalables, flexibles y adaptables para mantenerse competitivos. La incapacidad de gestionar eficazmente la dependencia y la complejidad desde las primeras etapas de diseño compromete la capacidad de los sistemas para evolucionar y responder a los cambios que demande el mercado. Esto plantea una oportunidad crucial para desarrollar

enfoques innovadores que permitan una evaluación sistemática y automatizada de estas métricas arquitectónicas, guiando la toma de decisiones desde una perspectiva fundamentada y cuantitativa.

Desde un enfoque académico, este estudio busca contribuir en el desafío que involucra la calidad de las aplicaciones de microservicios y, aunque se han propuesto diversas alternativas para evaluar la calidad de este tipo de arquitecturas, la mayoría carecen de modelos que integren métricas específicas para dependencia y complejidad en tiempo de diseño.

En conclusión, las consecuencias de no gestionar adecuadamente las dependencias entre microservicios involucran un acoplamiento rígido que va a dificultar la evolución del sistema, el incremento de los costos de mantenimiento y la afectación negativa de la escalabilidad. Por otro lado, las consecuencias de no gestionar adecuadamente la complejidad involucran el aumento en el riesgo de errores, la dificultad en la implementación de cambios y la reducción de la capacidad de respuesta ante nuevas demandas. Por lo tanto, la evaluación y optimización de estas características en las primeras etapas del diseño es esencial para garantizar sistemas más robustos y sostenibles.

El modelo propuesto en esta investigación, MOAM (Microservices Optimization Architectures Model), representa una herramienta propicia al integrar técnicas avanzadas de Machine Learning (ML) y Procesamiento de Lenguaje Natural (NLP) para analizar descripciones arquitectónicas y generar métricas cuantitativas que evalúen la dependencia y la complejidad. Este enfoque no solo permite identificar problemas potenciales, sino que también ofrece recomendaciones prácticas para mejorar el diseño arquitectónico, ofrece un enfoque sistemático y automatizado para apoyar a los desarrolladores en la toma de decisiones arquitectónicas desde las etapas iniciales del desarrollo.

El impacto potencial de este modelo trasciende el ámbito académico, ofreciendo una herramienta que puede ser adoptada por arquitectos de software y equipos de desarrollo en diversos sectores. Al automatizar la evaluación arquitectónica MOAM facilita la toma de decisiones fundamentadas que pueden favorecer a una reducción de los costos asociados al rediseño y mejora la calidad general del sistema desde las etapas iniciales. Este estudio tiene el potencial de aportar a las prácticas actuales, estableciendo un método de evaluación y optimización de arquitecturas de microservicios en tiempo de diseño.

1.2 PLANTEAMIENTO DEL PROBLEMA

El problema que aborda este trabajo de investigación doctoral radica en la dificultad de evaluar y optimizar arquitecturas de microservicios durante las etapas iniciales del desarrollo de software. En particular, se enfoca en los desafíos asociados a la dependencia y la complejidad, dos características fundamentales que influyen directamente en atributos clave de calidad como la escalabilidad, flexibilidad y mantenibilidad del sistema. Estas características, aunque cruciales, suelen ser difíciles de medir y gestionar debido a la naturaleza distribuida de los microservicios y la falta de herramientas efectivas que puedan automatizar su evaluación.

Los sistemas basados en microservicios, a pesar de su creciente adopción por su modularidad y adaptabilidad, presentan retos significativos en cuanto a su diseño y mantenimiento. La dependencia excesiva entre servicios puede dificultar su escalabilidad e introducir riesgos sistémicos, mientras que la complejidad estructural puede aumentar los costos de mantenimiento y reducir la capacidad de adaptación a los cambios del entorno. Además, la falta de métricas específicas y procesos estandarizados para evaluar estos factores

complica la toma de decisiones arquitectónicas en etapas tempranas, lo que a menudo conduce a sistemas rígidos y difíciles de evolucionar [4].

Para las aplicaciones de microservicios, las decisiones de arquitectura en el desarrollo del software, es uno de los primeros desafios que deben enfrentar los equipos. El problema se agudiza en escenarios donde los equipos de desarrollo necesitan diseñar sistemas rápidamente para responder a demandas cambiantes del mercado, enfrentando limitaciones de tiempo y recursos. En este contexto, los enfoques tradicionales para medir atributos arquitectónicos no ofrecen la granularidad ni la capacidad de análisis necesarias para abordar las complejidades inherentes a los microservicios [6][7].

Martin Fowler [8] describe la arquitectura de software como el concepto más elevado que los desarrolladores tienen acerca de un sistema en su entorno. Este entendimiento compartido abarca cómo se estructura el sistema y cómo interactúan sus componentes a través de interfaces. En este sentido, las decisiones arquitectónicas deben tomarse al inicio del proyecto para permitir que todos los miembros del equipo se adapten a estas decisiones, una buena arquitectura implica establecer procesos adecuados desde el principio, ya que las modificaciones posteriores pueden resultar complicadas y costosas. Fowler también menciona la irreversibilidad (acoplamiento excesivo, baja cohesión, deuda técnica, falta de refactorización, entre otras) como un factor clave que contribuye a la complejidad en el desarrollo de software. Por lo tanto, uno de los principales objetivos del arquitecto es diseñar una arquitectura que minimice este aspecto, facilitando así cambios y adaptaciones futuras en el diseño del software.

Desde la misma perspectiva anterior, la dependencia y la complejidad como dos características que tienen la capacidad de definir una arquitectura de microservicios, inciden directamante en la dificultad para generar cambios, el

costo de implementación y el tiempo de generación de dichos cambios. Es decir, mientras un componente arquitectónico es más difícil de cambiar, también es más complejo, entonces, el cambio es más costoso y afecta a la calidad interna del producto. La calidad interna permite entregas más rápidas con la menor cantidad de fallos y errores en el proceso, está directamente relacionada con las propiedades de flexibilidad y escalabilidad de las aplicaciones.

Otros desafíos relacionados con arquitecturas de microservicios son: 1) manejo de conceptos de modelado apropiados para casos de uso cambiantes, 2) necesidad de implementar herramientas de validación automática, en especial, desde el enfoque de la cohesión y responsabilidad única, 3) patrones de diseño para resolver problemas recurrentes, 4) relación entre los atributos de calidad, 5) métricas y patrones que no están claramente definidos. Todas estas son preocupaciones para la ingeniería de software porque su tratamiento es fundamental para diseñar correctamente sistemas que utilicen arquitecturas de microservicios [7] [9] [10] [11].

En general, se denota que, para el desarrollo de las aplicaciones basadas en microservicios, los temas relacionados con su arquitectura son relevantes y actualmente desafiantes, por tanto, establecer estrategias para un diseño eficiente representa una de las misiones elementales pero a la vez más difíciles de los arquitectos de software, pues requiere de tiempo para identificar las dependencias entre los servicios y la atención permanente para gestionar las mismas durante todo el ciclo de vida del software [12]. Así también, la complejidad es otro desafío importante, ya que son aplicaciones compuestas de muchos servicios que deben funcionar de forma independiente pero que requieren comunicación y evolución constante [13].

Este trabajo propone como solución el desarrollo de MOAM (Microservices Optimization Architectures Model), un modelo inteligente que utiliza técnicas avanzadas de Machine Learning y Procesamiento de Lenguaje Natural (NLP) para generar de forma automática la arquitectura de una aplicación de microservicios, posteriormente, analiza las descripciones arquitectónicas y proporcionar una serie de métricas para evaluar la dependencia y la complejidad del diseño arquitectónico. La propuesta busca no solo identificar problemas en la arquitectura, sino también guiar a los desarrolladores en la mejora de sus diseños, optimizando la calidad de los sistemas desde las primeras etapas del desarrollo.

Bajo el contexto anteriormente descrito, se plantea el problema de la siguiente manera:

¿Cómo evaluar, de forma automática, la dependencia y la complejidad en arquitecturas de microservicios durante las etapas iniciales del desarrollo de software, utilizando técnicas avanzadas de Machine Learning y Procesamiento de Lenguaje Natural para contribuir en la toma de decisiones arquitectónicas de los equipos de desarrollo?

1.3 PREGUNTAS DE INVESTIGACIÓN

Para resolver el problema de investigación se definieron las siguientes preguntas específicas (Preguntas de Investigación – PI).

1.3.1 PREGUNTA DE INVESTIGACIÓN 1 – PI.1

¿Qué desafíos y problemas se presentan en la evaluación de la calidad de los microservicios, asociados a los atributos, métricas y herramientas aplicables a la medición de la calidad de este tipo de software?

Con esta pregunta se pretende identificar los desafíos y problemas que se evidencian en la evaluación de la calidad de los microservicios relacionados de forma específica con los atributos y métricas que se aplican en la medición de la calidad de los sistemas basados en la arquitectura de microservicios. Para ello, se realizó una revisión sistemática de la literatura en bases de datos de información bibliográfica relacionadas al campo de la ingeniería informática como son: IEEE, Scopus y ACM.

1.3.2 PREGUNTA DE INVESTIGACIÓN 2 – PI.2

¿Cuál es la relación que se presenta entre la dependencia y la complejidad de las arquitecturas de los microservicios con los problemas de flexibilidad y escalabilidad de los mismos?

Para resolver esta pregunta se plantean dos actividades: 1) realizar una revisión sistemática de la literatura en bases de datos del estilo descrito en la revisión de la PI.1 y, 2) aplicar una encuesta a desarrolladores para que, desde su visión práctica, enuncien los problemas de flexibilidad y escalabilidad que se pueden presentar en el desarrollo de microservicios relacionados con arquitecturas dependientes y complejas.

1.3.3 PREGUNTA DE INVESTIGACIÓN 3 – PI.3

¿Cómo diseñar un modelo matemático de evaluación de la calidad para determinar dependencia y complejidad en la arquitectura de aplicaciones basadas en microservicios?

Luego de haber identificado las métricas que se aplican en la evaluación de los atributos de calidad de microservicios (PI.1), y de conocer los problemas de flexibilidad y escalabilidad que se pueden presentar en los microservicios a causa

de la dependencias y complejidad en sus arquitecturas (PI.2); se propuso un modelo matemático para medir estas características, y posteriormente, a partir de los resultados, plantear mejoras a los diseños arquitectónicos de los microservicios.

1.3.4 PREGUNTA DE INVESTIGACIÓN 4 – PI.4

¿Qué técnicas de Machine Learning para Procesamiento de Lenguaje Natural se pueden aplicar para definir un modelo inteligente que especifique dependencia y complejidad en el diseño de aplicaciones basadas en microservicios?

Para responder esta pregunta se desarrolló un proceso en MOAM (Microservices Optimization Aquitectures Model) que, mediante técnicas avanzadas de Machine Learning para Procesamiento de Lenguaje Natural, genera el diagrama arquitectónico de las aplicaciones con sus correspondientes descripciones de agrupamiento y dependencias internas y externas, luego las descripciones son analizadas y posteriormente evaluadas mediante el modelo matemático de especificación de dependencia y complejidad generado en la PI.3.

1.3.5 PREGUNTA DE INVESTIGACIÓN 5 – PI.5

¿Cómo demostrar que MOAM representa una alternativa eficaz para ayudar a los desarrolladores en la toma de decisiones sobre la definición de las arquitecturas de microservicios?

El modelo inteligente de especificación de dependencia y complejidad se aplicó en 4 casos de estudio, a partir de estos casos se validó la propuesta verificando que MOAM representa una alternativa eficaz para ayudar a los desarrolladores en la toma de decisiones sobre la definición de las arquitecturas

de microservicios. Con los resultados generados en la fase anterior, se realizó un análisis comparativo de la propuesta con otros enfoques similares, señalando semejanzas, diferencias y resaltando, sobre todo, el aporte del modelo propuesto [14].

1.4 OBJETIVOS Y ALCANCE

1.4.1 OBJETIVO GENERAL

Crear un modelo inteligente para especificar dependencia y complejidad en el diseño de aplicaciones basadas en microservicios, usando técnicas de Machine Learning para Procesamiento de Lenguaje Natural.

1.4.2 OBJETIVOS ESPECÍFICOS

Objetivo 1. Identificar los desafíos y problemas que se presentan en la evaluación de la calidad de los microservicios, asociados a los atributos, métricas y herramientas aplicables a la medición de la calidad de este tipo de software.

Objetivo 2. Caracterizar la relación que existe entre la dependencia y la complejidad en la arquitectura de microservicios con los problemas de flexibilidad y escalabilidad de los mismos.

Objetivo 3. Proponer un modelo matemático de evaluación de la calidad para determinar dependencia y complejidad en la arquitectura de aplicaciones basadas en microservicios.

Objetivo 4. Definir el modelo inteligente para especificar y evaluar la dependencia y complejidad en el diseño de aplicaciones basadas en microservicios, usando técnicas de Machine Learning para Procesamiento de Lenguaje Natural.

Objetivo 5. Demostrar que MOAM representa una alternativa eficaz para ayudar a los desarrolladores en la toma de decisiones sobre la definición de las arquitecturas de microservicios.

1.5 ALCANCE

El alcance de esta investigación se centra en abordar los problemas críticos asociados con la evaluación y optimización de las arquitecturas basadas en microservicios durante las etapas iniciales del diseño. En particular, el estudio se enfoca en dos métricas fundamentales, la dependencia y la complejidad. Estas características han demostrado ser determinantes en la calidad general de los sistemas basados en microservicios, afectando atributos clave como escalabilidad, flexibilidad y mantenibilidad. Sin embargo, debido a la naturaleza distribuida de los microservicios, estas métricas son complejas de medir y gestionar de manera eficiente en fases tempranas del ciclo de desarrollo.

El trabajo se limita a la evaluación arquitectónica en tiempo de diseño, excluyendo análisis posteriores al despliegue o durante la ejecución del sistema. La investigación propone un enfoque sistemático basado en técnicas avanzadas de Machine Learning y Procesamiento de Lenguaje Natural, herramientas que permiten generar el diagrama de arquitectura de una aplicación de microservicios y luego analizar las descripciones arquitectónicas para evaluar sus características de dependencia y complejidad y reportar los resultados de cada métrica. Este enfoque no incluye el análisis directo de código fuente ni aspectos relacionados con la implementación de los sistemas, en su lugar, orienta hacia una evaluación conceptual y estructural de las aplicaciones.

El modelo desarrollado, denominado MOAM (Microservices Optimization Architectures Model), fue validado mediante estudios de caso en arquitecturas reales y experimentos controlados. Esta validación permitió demostrar su efectividad al comparar los resultados con métodos existentes en la literatura, asegurando su funcionamiento eficaz tanto en contextos académicos como en la industria.

Finalmente, el impacto esperado de esta investigación se refleja en la generación de una herramienta práctica y automatizada que facilita la toma de decisiones arquitectónicas durante las primeras etapas de diseño. Además, este trabajo pretende contribuir al avance del conocimiento científico en la evaluación de la calidad de microservicios, proporcionando un marco metodológico que puede ser adoptado en diferentes contextos de desarrollo de software. Con esta propuesta se espera contribuir a mejorar la calidad de los sistemas desde su concepción optimizando factores críticos y reduciendo costos asociados al mantenimiento y la evolución futura.

1.6 PUBLICACIONES Y RECURSOS DERIVADOS DE ESTA TESIS

Como parte de la fase de comunicación de este trabajo de investigación doctoral se escribieron tres artículos científicos publicados en revistas internacionales, un artículo en redacción y una ponencia en congreso internacional:

1.6.1 PUBLICACIONES EN REVISTAS INTERNACIONALES

- Verónica C. Tapia and Carlos M. Gaona, Research Opportunities in Microservices Quality Assessment: A Systematic Literature Review, Journal of Advances in Information Technology, Vol. 14, No. 5, 2023, doi: 10.12720/jait.14.5.991-1002.
- Verónica C. Tapia, Carlos M. Gaona, Alison Marcalla, Edison Aimacaña Evaluando Dependencia y Complejidad en el Diseño de Microservicios: Un Enfoque Integral Revista Ibérica de Sistemas e Tecnologias de Informação, RISTI, N.º E73, 09/2024, pág. 590 ISSN: 1646-9895, https://www.risti.xyz/issues/ristie73.pdf.

- Verónica C. Tapia and Carlos M. Gaona, Automation of Software Development Stages with the OpenAI API, CSSE-Computer Systems Science and Engineering, 12- 2024, DOI: 10.32604/csse.2024.056979, https://www.techscience.com/csse/online/detail/22113.

1.6.2 ACTAS DE LA CONFERENCIA

- Verónica C. Tapia, Carlos M. Gaona, Alison Marcalla, Edison Aimacaña Evaluation dependency and complexity in microservice design: An Integrated Approach, I Congreso Internacional de Difusión de Resultados de I+D+I, UTC, 2024, ISBN 978-9942-7195-2-2.

1.6.3 ARTÍCULO EN REDACCIÓN

- Mathematical model of dependency and complexity specification of microservice-based applications.

1.6.4 RECURSOS

- Prototipo de una herramienta inteligente para medir dependencia y complejidad en arquitecturas de microservicios: MOAM (Microservices Optimization Architectures Model)

1.7 ORGANIZACIÓN DE LA TESIS

El documento está estructurado en siete capítulos, de la siguiente manera:

Capítulo 1: Introducción. Este capítulo aborda la motivación detrás del estudio, estableciendo los desafíos clave relacionados con la dependencia y complejidad en las arquitecturas de microservicios. Plantea preguntas de

investigación y objetivos generales, destacando la relevancia del modelo MOAM como solución automatizada para estos problemas.

Capítulo 2: Estado del Arte. Se realiza una revisión de la literatura, destacando conceptos fundamentales de microservicios, tales como: calidad del software; atributos, métricas y desafíos de la calidad de microservicios; técnicas avanzadas de Machine Learning y Procesamiento de Lenguaje Natural (NLP). Estos conceptos proporcionan el marco teórico para el modelo propuesto.

Capítulo 3: Metodología. Explica el enfoque basado en la ciencia del diseño para desarrollar y validar el modelo MOAM. Se describen las fases metodológicas como la fundamentación, desarrollo, evaluación y divulgación, integrando los distintos métodos aplicados como las revisiones sistemáticas, encuestas, prototipado, experimentos controlados y estudio de caso. En este capítulo se describen el modelo y su proceso de implementación, de forma secuencial se van respondiendo a las preguntas de investigación PI.1, P.I.2, P.I.3 y P.I.4.

Capítulo 4: Evaluación del modelo. Describe el proceso de validación del modelo a través de estudios de caso y análisis comparativos. Destaca resultados que demuestran la eficacia de MOAM frente a otros enfoques, enfatizando su impacto en la mejora de las arquitecturas. En este capítulo se sustenta la respuesta a la quinta pregunta de investigación P.I.5.

Capítulo 5: Conclusiones: Resume las contribuciones principales, limitaciones y futuras líneas de investigación. Se reafirma la utilidad del modelo en contextos reales y se destacan sus beneficios para arquitectos y desarrolladores de software.

CAPÍTULO 2

2 ESTADO DEL ARTE

2.1 APLICACIONES BASADAS EN MICROSERVICIOS

Las aplicaciones basadas en microservicios (AMS) son un enfoque arquitectónico y organizativo de desarrollo de software en el que las aplicaciones están compuestas por pequeños servicios independientes que se comunican a través de APIs basadas en RESTful o RPC y protocolos ligeros [15] [16].

Para Chris Richardson [17], los microservicios deben tener las siguientes características clave:

- Mantenible
- Comprobable
- Acoplado
- Implementable de forma independiente
- Capaz de ser desarrollado por un equipo pequeño

Por lo general, los microservicios se organizan como un conjunto de pequeños servicios granulares que se pueden desarrollar en diferentes plataformas a través de múltiples herramientas tecnológicas. (Figura 1)

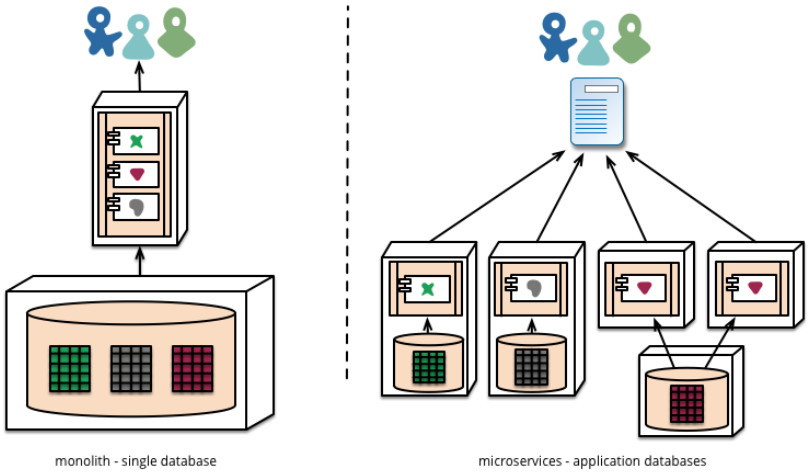


Figura 1. Arquitectura monolítica y arquitectura de microservicios . Fuente: Fowler [18].

2.1.1 ARQUITECTURA DE MICROSERVICIOS

La arquitectura de un microservicio propone descomponer aplicaciones en pequeños componentes, donde cada componente realiza una funcionalidad puntual de negocio. Esto difiere del desarrollo de aplicaciones basadas en arquitecturas monolíticas o arquitectura basadas en n-capas [19].

Una arquitectura de microservicios es un enfoque para la generación de una aplicación de servidor como un conjunto de servicios pequeños. Esto significa que una arquitectura de microservicios está orientada principalmente hacia el back-end, aunque el enfoque también se utiliza para el front-end. Cada servicio se ejecuta en su propio proceso y se comunica con otros procesos mediante protocolos como HTTP/HTTPS, WebSockets o AMQP (Advanced Message Queuing Protocol). Cada microservicio implementa un dominio de un extremo a otro específico o una capacidad empresarial dentro de un determinado límite de contexto, y cada uno se debe desarrollar de forma autónoma e implementar de forma independiente. Por último, cada microservicio puede poseer su modelo de datos de dominio relacionado y su lógica del dominio (soberanía y

administración de datos descentralizada), y podría basarse en otras tecnologías de almacenamiento de datos (SQL, NoSQL) y lenguajes de programación [20].

Una buena forma de dividir los microservicios puede ser aplicando el Principio de Responsabilidad Simple, donde cada microservicio realiza una única funcionalidad de negocio y propone un modelo centrado en procesos para analizar la estructura y las dependencias entre la capa de negocio y la capa de acceso a los datos de las aplicaciones [21].

En el enfoque de microservicios, la funcionalidad se aísla en servicios más pequeños, por lo que cada servicio puede escalarse de forma independiente. Este enfoque permite modificaciones ágiles e iteraciones rápidas de cada microservicio, ya que puede cambiar áreas específicas y pequeñas de aplicaciones complejas, grandes y escalables. Diseñar la arquitectura de aplicaciones específicas basadas en microservicios habilita una integración continua y prácticas de entrega continua. También acelera la entrega de nuevas funcionalidades en la aplicación. La composición específica de las aplicaciones también le permite ejecutar y probar los microservicios de manera aislada y hacerlos evolucionar de forma autónoma, a la vez que, mantiene contratos claros entre ellos, siempre y cuando no cambie las interfaces o los contratos, puede cambiar la implementación interna de cualquier microservicio o agregar nuevas funcionalidades sin que ello interrumpa otros microservicios. [20]

2.1.2 DECISIONES DE ARQUITECTURA

Al desarrollar un microservicio, el tamaño no debe ser lo más importante. En su lugar, el punto importante debe ser crear libremente servicios acoplados para que tenga autonomía de desarrollo, implementación y escala, para cada servicio. Por supuesto, al identificar y diseñar microservicios, debe intentar que sean lo

más pequeños posible, siempre y cuando no tenga demasiadas dependencias directas con otros microservicios. Más importante que el tamaño del microservicio es la cohesión interna que debe tener y su independencia respecto a otros servicios.[20]

Según [22], los criterios de división de microservicios son: capacidades comerciales, acceso a los datos, estructura del equipo, dependencias, consumo de recursos, ciclos de entrega. En cuanto a las prácticas aplicadas en el desarrollo están las siguientes:

La decisión sobre la granularidad de los microservicios, además de dividir por capacidades comerciales, los desarrolladores también deben considerar las restricciones específicas del producto y la implementación, por ejemplo, los relacionados con el consumo de recursos anticipados, la estructura del equipo, los ciclos de entrega.

Asignar un propietario explícito a cada microservicio (frente al modelo de propiedad compartida) ayuda a mejorar la integridad arquitectónica, la productividad y la calidad del producto resultante.

Configurar un soporte extenso y completo de registro y monitoreo temprano en el ciclo de vida de desarrollo, así como invertir en el seguimiento distribuido desde el principio, ayudan a aumentar calidad del producto. Soporte de herramientas automatizado, el uso de herramientas estandarizadas y el uso de interfaces abiertas, ayudan a reducir los costos operativos.

2.1.3 SERVICIO

Para Chris Richardson [17], uno de los desafíos del enfoque de AMS, es decidir cuándo tiene sentido usarlo. Al desarrollar la primera versión de una

aplicación, a menudo no se tienen los problemas que resuelve este enfoque. Además, el uso de una arquitectura distribuida y elaborada, disminuirá la velocidad del desarrollo, este puede ser un problema importante para las nuevas empresas cuyo mayor desafío suele ser cómo evolucionar rápidamente el modelo de negocio y la aplicación que lo acompaña. Sin embargo, más adelante, cuando el desafío sea cómo escalar y se necesite utilizar la descomposición funcional, las dependencias enredadas pueden dificultar la descomposición de su aplicación monolítica en un conjunto de servicios.

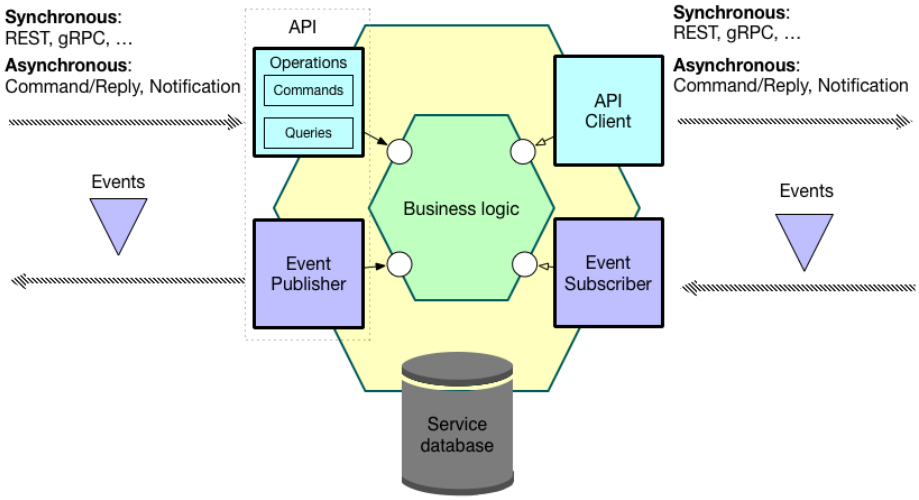


Figura 2: Estructura de un servicio. Fuente: [17]

La figura 2, muestra la estructura de un servicio desde la perspectiva de Chris Richardson [17]. Una arquitectura hexagonal, el núcleo del servicio es su lógica empresarial que está rodeada de adaptadores que se comunican con otros servicios y aplicaciones.

La lógica del negocio implementa las operaciones de la API y publica eventos. La lógica empresarial invoca las operaciones de otros servicios y se suscribe a sus eventos, persiste los datos en la base de datos del servicio.

Un servicio puede colaborar con otros servicios, un servicio puede invocar las operaciones de otro servicio, un servicio también puede suscribirse a los eventos publicados por otro servicio.

La base de datos de un servicio es privada, un servicio generalmente tiene una base de datos que almacena sus datos y, a veces, datos replicados de otros servicios.

En cuanto a las ventajas y desventajas de los servicios, a continuación se presenta un listado de ellas, de acuerdo al enfoque de Martin Fowler [18].

Ventajas:

Límites de módulos sólidos: los microservicios refuerzan la estructura modular que es particularmente importante para equipos grandes.

Implementación independiente: los servicios simples son más fáciles de implementar y, dado que son autónomos, es menos probable que causen fallas en el sistema cuando estos fallan.

Diversidad tecnológica: con los microservicios puede combinar varios lenguajes, marcos de desarrollo y tecnologías de almacenamiento de datos.

Desventajas:

Distribución : Los sistemas distribuidos son más difíciles de programar, ya que, las llamadas remotas pueden ser lentas y correr el riesgo de fallar.

Coherencia eventual: Mantener una coherencia sólida es extremadamente difícil para un sistema distribuido, lo que significa que todos tienen que gestionar la coherencia final.

Complejidad operativa: La complejidad operativa en los microservicios radica en gestionar múltiples servicios independientes en un entorno distribuido, abordando retos como el despliegue, escalabilidad, monitoreo, seguridad y manejo de fallos. Requiere herramientas avanzadas como Kubernetes, sistemas de observabilidad y estrategias CI/CD para garantizar un funcionamiento eficiente y cohesivo del sistema.

2.2 CALIDAD DEL SOFTWARE

La familia de normas ISO/IEC 25000, conocida como SQuaRE (System and Software Quality Requirements and Evaluation), es un conjunto de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software, es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126 e ISO/IEC 14598. [23]

2.2.1 ATRIBUTOS DE CALIDAD DE SOFTWARE

Los atributos de calidad se pueden clasificar en dos tipos, según el tipo de mapeo resultante. En el primer tipo, el atributo de calidad se asigna con varios requisitos específicos. En el otro tipo, el atributo de calidad no se asigna con requisitos funcionales específicos, pero tiene un efecto sobre todos los requisitos funcionales. [24].

Los atributos de calidad son factores generales que afectan el comportamiento de un sistema en tiempo de ejecución, el diseño del sistema y la experiencia del usuario. Describen el propósito del sistema en el ambiente para el cual fue construido. Desde una perspectiva técnica, los atributos de calidad direccionan decisiones arquitectónicas y de diseño significativas.

La ISO 25000, define los siguientes criterios de calidad:

- Funcionalidad
- Eficiencia – Desempeño
- Compatibilidad
- Operabilidad – Usabilidad
- Fiabilidad
- Seguridad
- Mantenibilidad
- Portabilidad

Cabe indicar que, cada criterio tiene sus respectivos subcriterios.

Atributos de calidad aplicables a microservicios

La calidad en microservicios se refiere a un conjunto de características que permiten evaluar su efectividad, rendimiento y capacidad de satisfacer las necesidades de los usuarios. En el contexto de arquitecturas basadas en microservicios (MSA), la calidad incluye aspectos como la granularidad adecuada, el acoplamiento bajo y la cohesión alta, además de métricas como tiempo de respuesta, escalabilidad y seguridad. Este marco es esencial para garantizar que los microservicios sean independientes, mantenibles y capaces de adaptarse a cambios en el sistema o en los requisitos del negocio [9].

Los atributos de calidad de microservicios se basan en el estándar ISO/IEC 25010, incluyen características como rendimiento, compatibilidad, mantenibilidad y seguridad. Para medir la calidad se utilizan métricas como tamaño, acoplamiento y cohesión. La arquitectura es importante porque define componentes, dependencias y sirve como base para análisis, evolución y gestión del sistema [25]. A continuación, se detallan los atributos mencionados:

- La escalabilidad en microservicios se refiere a la capacidad de un sistema para manejar un aumento o disminución en la carga de trabajo sin comprometer el rendimiento. Existen tres enfoques principales: 1) escalabilidad horizontal, consiste en agregar más recursos a nivel lógico, como añadir servidores a un clúster existente, 2) escalabilidad vertical, se refiere a aumentar los recursos físicos de una unidad, como añadir más CPU o memoria a un servidor, y 3) escalabilidad Z-axis, implica dividir la carga entre diferentes servidores según un criterio específico, como la clave primaria de una entidad de base de datos.
- El rendimiento mide la capacidad de un sistema para cumplir con ciertos requisitos de tiempo al responder a eventos. En microservicios el rendimiento está influenciado por: 1) asignación de recursos, puede afectar el rendimiento al cambiar el tiempo de respuesta o la capacidad de procesamiento (throughput), 2) interacciones entre servicios, una mayor cantidad de microservicios puede incrementar la sobrecarga de comunicación, disminuyendo el rendimiento, 3) trade-offs, un diseño de microservicios más granular puede mejorar la escalabilidad, pero incrementar las interacciones podría afectar negativamente el rendimiento.
- La disponibilidad mide la capacidad de un sistema para estar operativo dentro de ciertos límites de tiempo. Se calcula como la probabilidad de que un sistema proporcione los servicios requeridos durante un intervalo de tiempo determinado: 1) tolerancia a fallos, los microservicios deben ser diseñados para manejar fallos de hardware, red o nivel de aplicación, 2) reparación rápida, es esencial reducir el tiempo de inactividad para cumplir con los acuerdos de nivel de servicio (SLA), 3) soluciones comunes, uso de réplicas, balanceadores de carga y mecanismos como failover para mantener la disponibilidad.

- La monitoriabilidad evalúa la capacidad de supervisar el sistema mientras está en funcionamiento, un aspecto crucial en microservicios debido a su naturaleza dinámica y compleja. Los niveles de monitoreo son: 1) nivel de infraestructura, supervisión de máquinas virtuales, contenedores, etc., 2) nivel de aplicación, evaluación de métricas como tiempos de respuesta, 3) nivel de entorno, supervisión de la red y otros componentes externos. Una buena monitorabilidad mejora otros atributos como escalabilidad, rendimiento y disponibilidad.
- La seguridad se refiere a la capacidad del sistema para proteger datos e información frente a accesos no autorizados, mientras proporciona acceso a usuarios y sistemas autorizados. 1) complejidad de la red, debido a que la lógica se distribuye en múltiples servicios, las interacciones aumentan, lo que crea más puntos de ataque, 2) medidas comunes, encriptación de datos en tránsito y en reposo, 3) uso de tokens para autenticación y autorización, 4) firewall y segmentación de redes para limitar el acceso.
- La testabilidad mide la facilidad con la que se pueden identificar errores en el sistema mediante pruebas. En microservicios esto es especialmente importante debido a las relaciones complejas entre servicios. Tipos de pruebas: 1) pruebas unitarias: verifican funciones y métodos específicos dentro de un microservicio, 2) pruebas de integración, evalúan cómo interactúan los adaptadores de entrada/salida con otros servicios, 3) pruebas de componentes, prueban cada servicio por separado sin depender de otros servicios, 4) pruebas de extremo a extremo, involucran todo el sistema, incluyendo la interfaz de usuario. La testabilidad afecta otros atributos como rendimiento, disponibilidad y seguridad, al garantizar que el sistema pueda demostrar sus fallos en condiciones controladas.

Los atributos no son independientes, son fundamentales para garantizar que las arquitecturas basadas en microservicios sean robustas, escalables, seguras y fáciles de mantener. Su correcta implementación y monitoreo permite que los sistemas cumplan con los requisitos de los usuarios y sean sostenibles a largo plazo.

La tabla 1 presenta un conjunto amplio de criterios de evaluación de calidad para arquitecturas de microservicios clasificados según análisis estático y dinámico. En el análisis estático se evalúan atributos como granularidad, líneas de código (LOC), interfaces abiertas, alta cohesión y bajo acoplamiento. En el análisis dinámico se incluyen atributos como costo de ejecución, tiempo de respuesta, disponibilidad, tasa de ejecución exitosa, frecuencia de uso, escalabilidad, gestión de salud, balanceo de carga, alineación organizacional y seguridad. Esta clasificación, basada en [9], organiza atributos clave para analizar y mejorar la calidad de los sistemas basados en microservicios, cubriendo tanto características estructurales como comportamentales.

Tabla 1. Atributos de calidad aplicables a microservicios

Atributos de calidad para microservicios	
Análisis estático	**Análisis dinámico**
Granularidad	Execution cost
LOC	Response Time
Open interfaces	Availability
High cohesion	Succ. Exec. Rate
Loose coupling	Usage Frequency
	Scalability
	Independence
	Maintainability
	Deployment
	Health Management
	Modularity

Manageability
Performance
Reusability
Tech. Heterogeneity
Agility
Security
Load Balancing
Org. Alignment

Fuente: [9]

Desde la perspectiva de este trabajo, los atributos forman un marco esencial para evaluar y garantizar la calidad de los microservicios, promoviendo su independencia, flexibilidad y capacidad de adaptación a distintos entornos y requerimientos. En los siguientes párrafos se citan las definiciones de algunos de los atributos:

- Granularidad. Define el tamaño y alcance de un microservicio, evaluando si está diseñado para abordar una funcionalidad de negocio específica de manera efectiva. Se mide comúnmente a través de métricas como líneas de código (LOC) o el número de operaciones disponibles en sus interfaces. Un nivel adecuado de granularidad equilibra la simplicidad del microservicio con su capacidad para cumplir objetivos de negocio, evitando servicios demasiado grandes (difíciles de mantener) o demasiado pequeños (innecesariamente complejos).
- Cohesión. Mide la relación y alineación de las operaciones dentro de un microservicio para garantizar que se enfoquen en una única responsabilidad o funcionalidad de negocio. Un microservicio con alta cohesión sigue el principio de "responsabilidad única", lo que facilita

su comprensión, mantenimiento y evolución al evitar la dispersión de objetivos dentro del servicio.

- Acoplamiento. Evalúa las dependencias entre microservicios. El acoplamiento bajo implica que un microservicio interactúa con otros mediante interfaces bien definidas y evita referencias circulares. Esto mejora la flexibilidad y reduce los riesgos de fallo en cascada.
- Tiempo de Respuesta. Mide el retraso entre una solicitud enviada a un microservicio y la recepción de la respuesta. Es crucial para la percepción de rendimiento y se calcula mediante el tiempo de ejecución excluyendo retrasos de red.
- Seguridad. Evalúa la capacidad de proteger los datos y prevenir accesos no autorizados. Se analiza considerando las interfaces expuestas y la implementación de tecnologías específicas para asegurar el sistema.
- Resiliencia. Describe la capacidad del microservicio para recuperarse de fallos, manteniendo la consistencia de datos y restaurando el estado previo al error.
- Reutilización. Mide la capacidad de un microservicio para ser utilizado en otros sistemas o contextos con modificaciones mínimas. Los servicios con bajo acoplamiento y alta cohesión son más reutilizables, lo que reduce costos de desarrollo en el largo plazo.

Así también, el trabajo de Felipe Osses y otros [10] describe un estudio exploratorio, basado en una revisión sistemática de la literatura que arrojó 1067 estudios sobre 44 patrones y 2 tácticas arquitectónicas para microservicios, la mayoría de patrones y tácticas arquitectónicas están asociadas a cinco atributos de calidad: escalabilidad, flexibilidad, capacidad de prueba, desempeño y

elasticidad. Es decir, con base en los resultados de este estudio, se puede deducir que los mencionados atributos son aplicables a microservicios.

2.2.1.1 *Métricas de Calidad*

Las métricas de calidad son herramientas críticas en la ingeniería de software ya que proporcionan un medio para medir de manera objetiva los atributos no funcionales de un sistema, fundamentales para su éxito a largo plazo. Estas métricas permiten evaluar cómo un software cumple con estándares de calidad establecidos y aseguran que los sistemas sean efectivos, confiables y sostenibles [9], [11], [26].

Clasificación y Función:

Métricas internas (estáticas): Evalúan la calidad del diseño y la arquitectura antes de la ejecución. Los Indicadores de estass métricas son el acoplamiento, la cohesión y la complejidad ciclomática ayudan a entender la mantenibilidad, modularidad y claridad del código, facilitan decisiones como la refactorización y la gestión de deuda técnica.

Métricas externas (dinámicas): Analizan el comportamiento del sistema durante la ejecución, los parámetros de este tipo de métricas son el tiempo de respuesta, la disponibilidad, la latencia y el uso de recursos evalúan el rendimiento y la experiencia del usuario. Son críticas para garantizar la escalabilidad y la capacidad de respuesta bajo diferentes cargas.

Relación con Atributos de Calidad:

- Rendimiento: Métricas como el tiempo de ejecución y la tasa de transacciones exitosas son esenciales para sistemas en tiempo real o con alta carga transaccional.

- Mantenibilidad: Indicadores como las líneas de código (LOC) y el número de dependencias reflejan la facilidad de actualización y la estabilidad ante cambios.
- Escalabilidad: El uso eficiente de CPU, memoria y red permite identificar cuellos de botella y planificar crecimiento.
- Seguridad: Métricas relacionadas con el tiempo de detección y solución de vulnerabilidades son clave para sistemas críticos.

Las métricas de calidad permiten monitorear el progreso del desarrollo y detectar problemas tempranamente, reduciendo costos y riesgos; además, proveen una base para comparar alternativas de diseño, tomar decisiones informadas y mejorar procesos y prácticas, aseguran que el software entregue valor al usuario final al cumplir con requisitos de eficiencia y confiabilidad.

Aunque la importancia de estas métricas es un hecho evidente, existen desafíos en su aplicación como es el limitado número de herramientas automáticas que integren la dificultad de correlacionar métricas de calidad para medir diferentes factores. Sin embargo, estas limitaciones presentan oportunidades para la investigación y el desarrollo de nuevas metodologías que permitan evaluar arquitecturas modernas como los microservicios, considerando su alta modularidad y dependencia de sistemas distribuidos.

2.2.1.2 *Desafíos en la evaluación de la calidad de microservicios*

Los desafíos más citados en la bibliografía son: seguridad, granularidad, rendimiento, monitoreo, estrategia organizacional, orquestación, coreografía, escalabilidad, descomposición y refactorización de monolitos, la mayoría son atributos de análisis estático [27].

A continuación, algunos otros estudios destacados sobre estos desafíos:

Estudio [28], menciona como desafíos la dependencia del software como servicio, la cultura y estructura organizacional, la gobernanza, las dificultades asociadas con la descomposición/refactorización de monolitos, la gestión de datos maestros, la orquestación y coreografía, las pruebas y el rendimiento.

Estudio [29], señala como desafíos generales las pruebas, el monitoreo y el modelado de microservicios basado en el rendimiento; como desafíos particulares, las estrategias para pruebas de regresión de desempeño, monitoreo de desempeño bajo cambios continuos de software y conceptos de modelado de desempeño apropiados para casos de uso cambiantes.

Estudio [30], los desafíos en el desarrollo de microservicios son: medir, controlar y mantener un nivel satisfactorio de calidad de la arquitectura del sistema.

Estudio [31], señala como desafíos las dificultades asociadas con la descomposición y refactorización de monolitos.

Estudio [32], indica que no hay suficiente evidencia sobre cómo evaluar si las decisiones arquitectónicas tomadas en microservicios producen deuda técnica.

2.3 PROCESAMIENTO DEL LENGUAJE NATURAL (NLP)

El procesamiento del lenguaje natural (NLP), es una disciplina de la inteligencia artificial (IA) enfocada en la interacción entre las computadoras y el lenguaje humano, buscando que las máquinas comprendan, procesen y generen lenguaje de forma similar a las personas, se sitúa en el ámbito de la inteligencia artificial y la lingüística computacional, con el objetivo de investigar el lenguaje natural como medio de comunicación entre humanos y máquinas [33].

Originalmente, el término se refería únicamente a la capacidad de un sistema para interpretar textos. Sin embargo, ha evolucionado para abarcar toda la lingüística computacional. Dentro de sus principales áreas se encuentran la

generación de lenguaje natural (NLG, por sus siglas en inglés), que permite a las computadoras crear mensajes propios, y la comprensión del lenguaje natural (NLU, por sus siglas en inglés), enfocada en interpretar jerga, errores ortográficos, errores de pronunciación y otras variantes del lenguaje. NLP utiliza aprendizaje automático para analizar y comprender datos lingüísticos. Los modelos entrenados identifican patrones gramaticales y lingüísticos reales para aplicaciones como traducción automática, asistentes virtuales y chatbots. Estas tecnologías procesan datos en segundo plano y responden a comandos o necesidades específicas, como correctores ortográficos, análisis de sentimientos y clasificación de correos [34][35]

El Procesamiento del Lenguaje Natural (NLP) es un campo de estudio interdisciplinario que se enfoca en dotar a los sistemas computacionales de la capacidad para comprender, interpretar y procesar el lenguaje humano de manera efectiva. Este dominio combina principios y técnicas de la ciencia de datos, la inteligencia artificial (especialmente del aprendizaje automático) y la lingüística, con el propósito de extraer y generar significado a partir del lenguaje humano, permitiendo su aplicación en una amplia variedad de contextos [36].

Las diferentes formas de expresión lingüística humana, que incluyen tanto el lenguaje escrito como el oral y otras modalidades, presentan retos únicos en su tratamiento. No obstante, el procesamiento de textos escritos es el área en la que el NLP ha alcanzado un mayor desarrollo, gracias a la abundancia de datos disponibles en formato digital y a la relativa simplicidad para acceder y analizar dichos recursos. NLP abarca a la ciencia de datos, la inteligencia artificial (Machine Learning), y la lingüística [37].

El procesamiento del lenguaje natural (NLP) utiliza aprendizaje automático para analizar y comprender datos lingüísticos. Los modelos entrenados

identifican patrones gramaticales y lingüísticos reales para aplicaciones como traducción automática, asistentes virtuales y chatbots. Estas tecnologías procesan datos en segundo plano y responden a comandos o necesidades específicas, como correctores ortográficos, análisis de sentimientos y clasificación de correos.

2.3.1 FUNCIONAMIENTO DEL PROCESAMIENTO DE LENGUAJE NATURAL (NLP)

Los pasos fundamentales para el funcionamiento del procesamiento del lenguaje natural son [37]:

Preprocesamiento de texto

Este primer paso transforma el texto sin procesar en un formato más comprensible para las máquinas, incluye los siguientes subpasos:

- Tokenización: División del texto en unidades más pequeñas como palabras o frases.
- Normalización: Conversión de todo el texto a minúsculas para evitar diferencias entre palabras similares.
- Eliminación de palabras vacías: Filtrado de palabras comunes que no aportan significado significativo.
- Stemming y lematización: Reducción de palabras a su forma raíz para facilitar el análisis.
- Limpieza del texto: Eliminación de caracteres especiales y números que pueden complicar el análisis.

Este proceso asegura que el texto esté limpio y estandarizado para un análisis efectivo por modelos de machine learning.

Extracción de características

Consiste en convertir el texto preprocesado en representaciones numéricas. Se utilizan técnicas como:

- Bolsas de palabras y TF-IDF: Métodos para cuantificar la presencia e importancia de las palabras en un documento.
- Incrustaciones de palabras (Word2Vec, GloVe): Representaciones vectoriales que capturan relaciones semánticas entre palabras.
- Incrustaciones Contextuales: Mejoran la representación al considerar el contexto en el que aparecen las palabras.

Análisis de texto

Este paso implica interpretar el texto mediante diversas técnicas, tales como:

- Etiquetado de Partes del Discurso (POS): Identificación de funciones gramaticales.
- Reconocimiento de Entidades Nombradas (NER): Detección de nombres, lugares y fechas.
- Análisis Sintáctico: Estudio de las relaciones gramaticales entre palabras.
- Análisis de Sentimientos: Evaluación del tono emocional del texto.
- Comprensión del Lenguaje Natural (CLN): Análisis del significado detrás de las oraciones

A través de estas técnicas, el análisis de texto de NLP transforma el texto no estructurado en información.

Entrenamiento de modelos

Consiste en suministrar al modelo una cantidad extensa de datos textuales o lingüísticos previamente etiquetados, de forma que éste pueda identificar

patrones, estructuras gramaticales, semánticas y contextuales en el lenguaje. A medida que el modelo analiza y aprende de estos ejemplos, va ajustando sus parámetros internos para mejorar su capacidad de predecir o clasificar textos futuros con precisión, detectar sentimientos, reconocer entidades, resumir documentos o traducir idiomas, entre otras tareas. Una vez completado el entrenamiento, el modelo debería ser capaz de aplicar los patrones y reglas extraídas a nuevos fragmentos de texto, respondiendo adecuadamente a las distintas problemáticas o funcionalidades requeridas por el caso de uso.

2.3.2 APLICACIONES Y HERRAMIENTAS DEL PROCESAMIENTO DEL LENGUAJE NATURAL

Entre las aplicaciones y herramientas más comunes del NLP están [34] [38]:

- Extracción de información: Analiza textos estructurados y no estructurados para extraer datos relevantes.
- Traducción automática: Predice estructuras gramaticales en distintos idiomas.
- Motores de búsqueda: Mejoran resultados basándose en las interacciones del usuario.
- Asistentes de voz y chatbots: Responden a comandos predefinidos y mejoran la productividad.
- Clasificación y análisis de datos: Desde filtrado de spam hasta análisis de opiniones, optimizan procesos como reclutamiento, atención al cliente y gestión empresarial.
- Análisis de sentimientos: Evalúan emociones en redes sociales para comprender mejor el contexto de publicaciones y retroalimentación.

2.3.3 HERRAMIENTAS DESTACADAS DE NLP

Algunas herramientas para aplicar técnicas de NLP son [37][35]:

- SpaCy, Gensim: Librerías de código abierto para procesamiento de texto y minería de datos.
- Amazon Comprehend, IBM Watson Tone Analyzer, Google Cloud Translation: Servicios comerciales diseñados para analizar texto, identificar emociones y realizar traducciones dinámicas.
- Natural Language Toolkit (NLTK): Biblioteca en Python para tareas como clasificación y tokenización.
- TensorFlow: Plataforma para entrenar modelos de machine learning aplicables a NLP.

2.4 MACHINE LEARNING

Machine learning (ML) es un campo de la inteligencia artificial que se centra en desarrollar algoritmos y modelos capaces de aprender patrones a partir de datos y hacer predicciones o tomar decisiones sin necesidad de ser programados explícitamente para cada tarea. Desde hace varios años, este campo ha recibido mucha atención precisamente por su capacidad de predecir con precisión una amplia variedad de fenómenos complejos. Además de las predicciones, los modelos ML son capaces de producir conocimiento sobre las relaciones de dominio contenidas en los datos denominadas interpretaciones aplicadas [39] [40].

El aprendizaje automático es una rama en evolución de los algoritmos computacionales que están diseñados para emular la inteligencia humana aprendiendo del entorno circundante, es considerado crucial en la nueva era del

Big Data. Las técnicas basadas en el aprendizaje automático se han aplicado con éxito en diversos campos que van desde el reconocimiento de patrones, visión por computadora, ingeniería de naves espaciales, finanzas, entretenimiento, biología computacional, entre otras áreas. Los grados de complejidad de los algoritmos aplicados en estos procesos pueden variar y pueden involucrar varias etapas de interacciones sofisticadas hombre-máquina y toma de decisiones [41]

2.4.1 ALGORITMOS DE MACHINE LEARNING PARA NLP

De acuerdo con [42][43] los tipos de Machine Learning son:

- Algoritmos más usados de Supervised Learning.

Se basa en la información de entrenamiento, se proporciona una cantidad de datos definidos con etiquetas, esto hace que el sistema se vaya entrenado. Al introducir una cantidad suficiente de estos datos, sin necesidad de utilizar etiquetas se pueden introducir nuevos datos, basándose en varios patrones que durante su entrenamiento ha registrado.

- Algoritmos más usados de Unsupervised Learning.

Las etiquetas o valores verdaderos no son utilizados en este tipo de aprendizaje. Ya que en este caso, la finalidad de este sistema es la de la abstracción y comprensión de patrones de información de modo directo. Se trata de un método de entrenamiento similar a la forma en que procesa la información un ser humano. También es conocido como un modelo de problema (clustering).

- Aprendizaje semisupervisado

En este tipo de aprendizaje, se toman en cuenta los datos supervisados y no supervisados, combinando los dos anteriores para que pueda clasificar de manera adecuada.

- Aprendizaje por refuerzo

A partir de la experiencia, los sistemas aprenden con este modelo de aprendizaje. Se trata de una técnica que se basa en el error y la prueba, optimizando el comportamiento del sistema, utilizando funciones de premio. Para la Inteligencia Artificial es una forma muy interesante de aprendizaje, ya que no requiere introducción de información en gran cantidad.

- Transducción

Este sistema es muy similar al aprendizaje supervisado, aunque no construye una función de manera clara, solo trata de predecir variedades de ejemplos futuros.

- Aprendizaje multi-tarea

Son los métodos de aprendizaje que emplean previamente conocimiento aprendido por el sistema, donde están propensos a enfrentarse a diversos problemas.

- Técnicas de Machine Learning para NLP

Según Juan Bagnato [44] las técnicas comunes de Machine Learning para realizar NLP son:

- Tokenizar
- Tagging Part of Speech (PoS)
- Shallow parsing / Chunks

- Significado de las palabras
- Pragmatic Analysis
- Bag of words

2.5 TRABAJOS RELACIONADOS

En esta sección se citan algunos trabajos relacionados con el fin de presentar otras contribuciones al avance del conocimiento en la disciplina de la evaluación de la calidad de las aplicaciones basadas en microservicios:

El estudio [9] cuyo objetivo es identificar y definir un conjunto mínimo de atributos de calidad aplicables a microservicios resultantes de la descomposición de sistemas monolíticos. Se busca establecer métricas claras y generalizables para evaluar la idoneidad de los microservicios generados por herramientas automatizadas, con el fin de reducir la dependencia del juicio subjetivo de arquitectos de software. El trabajo aborda la falta de consenso en la literatura sobre lo que constituye un "buen" microservicio y propone un enfoque sistemático para la validación de la calidad de estos componentes en la transición desde arquitecturas monolíticas, El estudio identificó un conjunto mínimo de atributos de calidad esenciales para evaluar microservicios descompuestos a partir de aplicaciones monolíticas, incluyendo granularidad, cohesión, acoplamiento, tiempo de respuesta, escalabilidad, costo de ejecución, manejo de fallos, reusabilidad y seguridad, los cuales pueden evaluarse mediante análisis estático o dinámico. La validación en la industria confirmó su relevancia, destacando la necesidad de herramientas más precisas para medir independencia y acoplamiento, así como mejoras en la automatización del proceso de evaluación arquitectónica.

Estudio [45], propone Microservices Backlog, un modelo basado en algoritmos genéticos para especificar la granularidad e identificar microservicios a partir de un product backlog en metodologías ágiles. El enfoque se valida mediante experimentos comparativos con técnicas existentes como Domain-Driven Design (DDD), Service Cutter, el método de Baresi y otros. Los resultados muestran que el modelo optimiza la cohesión y el acoplamiento, permitiendo una descomposición más eficiente de los microservicios en tiempo de diseño. El trabajo aborda el problema de encontrar la granularidad adecuada para microservicios, una tarea generalmente subjetiva y basada en la experiencia de los arquitectos de software. Los resultados experimentales validan su aplicabilidad, mostrando mejoras significativas en términos de acoplamiento, cohesión y mantenibilidad respecto a métodos tradicionales. Se propone como trabajo futuro la integración de técnicas de clustering y análisis semántico para mejorar la precisión de la descomposición.

El trabajo [46] , el objetivo del estudio fue proponer un marco automatizado para agrupar historias de usuario en el desarrollo ágil de software mediante técnicas de aprendizaje automático. El marco pretende reducir el esfuerzo manual que supone la gestión de historias de usuario, que a menudo requiere mucho tiempo y es propensa a errores, utilizando métodos de extracción de características y agrupamiento para categorizar automáticamente las historias de usuario en función de sus similitudes. Los resultados del estudio indicaron que TextCNN era el algoritmo de extracción de características más eficaz para agrupar historias de usuario. Entre las técnicas de agrupación, K-means y la agrupación jerárquica superaron a las demás. La combinación de TextCNN con K-means o Mean-Shift dio los mejores resultados en términos de eficacia de agrupamiento de historias de usuarios.

El trabajo [47] presenta un enfoque basado en métricas y herramientas para evaluar arquitecturas de microservicios. Se propone un método de evaluación que permite analizar la adherencia a principios arquitectónicos clave, utilizando modelos generados a partir de datos de comunicación del sistema. La herramienta desarrollada, denominada Microservice Architecture Analysis Tool (MAAT), permite visualizar y evaluar las dependencias y calidad de la arquitectura. El método y la herramienta MAAT fueron aplicados a un caso de estudio real en una aplicación con aproximadamente 50 microservicios. Algunos hallazgos clave incluyen: identificación de problemas de acoplamiento, se encontraron ciclos en la comunicación entre servicios, lo que puede afectar la escalabilidad y rendimiento, se verificó si los microservicios eran demasiado grandes o pequeños, según el número de interfaces. El análisis de métricas facilitó la discusión con arquitectos del sistema, confirmando la utilidad de la herramienta para evaluar y mejorar el diseño de microservicios.

El trabajo [48] propone MicroQuality, un enfoque basado en datos para evaluar la calidad de arquitecturas de microservicios (MSA). Su objetivo es proporcionar una solución extensible y automatizada que permita medir atributos de calidad clave en sistemas basados en microservicios, utilizando técnicas de Model-Driven Engineering (MDE). El estudio evalúa las aplicaciones en tiempo de diseño, utilizando modelos arquitectónicos extraídos del código fuente. Se definen métricas de calidad relevantes, como acoplamiento, cohesión y complejidad. Se utiliza un metamodelo de calidad para describir cómo deben calcularse estas métricas. Las métricas pueden calcularse individualmente o agregarse en atributos más complejos como la mantenibilidad.

CAPÍTULO 3

3 METODOLOGÍA

Para este trabajo se plantea una estrategia metodológica adaptada al planteamiento de Alan Hevner y otros [49], acerca de la investigación de la ciencia del diseño en Sistemas de Información como artefacto de tecnologías de la información, cuyo propósito de creación es abordar un problema importante, el cual debe describirse de manera eficaz, permitiendo su implementación y aplicación en un dominio apropiado (figura 3).

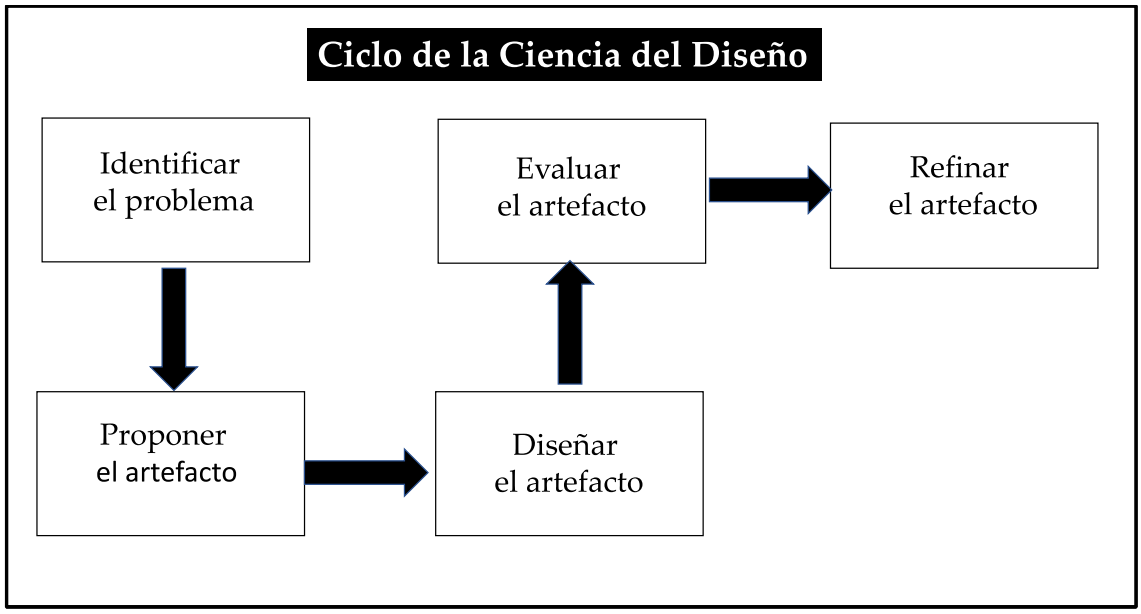


Figura 3: Ciclo de la Ciencia de Datos. Fuente: [49]

En esta investigación, el término artefacto en adelante se reemplaza por modelo.

El trabajo propone la creación de un modelo inteligente para especificar dependencia y complejidad en el diseño de aplicaciones basadas en microservicios denominado “MOAM”.

Tabla 2. Fases, actividades, métodos y resultados esperados

Fases	Actividad	Métodos	Resultados esperados
Fase 1. Fundamentación	Elaboración de la fundamentación teórica sobre arquitecturas basadas en microservicios.	Método de análisis y síntesis de la información bibliográfica.	Fundamentación teórica
	Establecer el estado del arte sobre atributos, métricas y desafíos en la calidad de microservicios.	Método de revisión bibliográfica, [32].	Estado del arte
			Artículo científico de revisión enviado a revisión en una revista internacional.
			Caracterización de los problemas presentados desde la práctica.
			Artículo científico sobre la caracterización de los problemas de flexibilidad y escalabilidad a causa de la dependencia y complejidad en el diseño de microservicios.
			Artículo científico sobre las métricas para el cálculo de dependencia y complejidad.
Fase 2. Desarrollo	Establecer el estado de la práctica en relación a los problemas de flexibilidad y escalabilidad que se pueden presentar, a causa de la dependencia y complejidad en las arquitecturas de microservicios.	Técnica de la encuesta aplicada a desarrolladores de aplicaciones basadas en microservicios.	Informe técnico sobre el diseño del modelo matemático para el cálculo de la dependencia y complejidad en arquitecturas de microservicios.

	Identificar las métricas de dependencia y complejidad que afecten los atributos de flexibilidad y escalabilidad en arquitecturas de microservicios.	Análisis y síntesis de información bibliográfica.	Informe técnico sobre las técnicas de ML aplicadas a la NLP.
	Elaborar el modelo matemático para el cálculo de dependencia y complejidad en arquitecturas de microservicios.	Análisis y síntesis de información bibliográfica	Artículo científico para la divulgación del modelo inteligente creado, enviado a revisión en una revista internacional.
	Identificar las técnicas de ML para NLP que se pueden aplicar para definir el modelo inteligente que especifique dependencia y complejidad en aplicaciones basadas en microservicios. Elaborar el modelo inteligente para definir el diseño arquitectónico de microservicios, optimizando las características de dependencia y complejidad.		
Fase 3. Evaluación	Determinar el caso de estudio para validar MOAM.	Método propuesto por Alan Hevner y otros [33] para el diseño de modelos de TI.	Informe técnico donde se detalla el caso de estudio propuesto.
	Desarrollar la aplicación de microservicios para el caso de estudio, aplicando MOAM para definir el modelo arquitectónico.	Caso de estudio	Artículo científico sobre las pruebas, validación y refinamiento del modelo creado.
	Probar, validar y refinar la propuesta implementada.	MOAM para el diseño de la arquitectura de la aplicación de microservicios.	
	Análisis comparativo de la propuesta con otros enfoque similares, identificando semejanzas, diferencias y aportes de la propuesta planteada.	Experimentos controlados.	
Fase 4. Divulgación	Elaborar el informe final de la investigación.	Análisis y síntesis de información.	Artículos de investigación donde se describa el desarrollo del trabajo doctoral y los aportes obtenidos, enviados a revistas internacionales.
	Elaborar artículos de investigación para la presentación en eventos científicos internacionales.		Ponencias en eventos científicos internacionales.

Una visión global del proceso metodológico que se propuso en este trabajo se describe en las cuatro fases detalladas en la Tabla 2.

- Fase 1: Fundamentación, se resuelven las preguntas PI.1 y PI.2.
- Fase 2: Desarrollo, se resuelven las preguntas PI.3 y PI.4.
- Fase 3: Evaluación, se resuelve la pregunta PI.5.
- Fase 4: Divulgación, consiste en la difusión de los resultados de cada una de las fases a través de la publicación de artículos científicos y ponencias en eventos académicos y científicos.

Con el cumplimiento de lo planificado en cada una de las fases se logran los objetivos determinados en esta propuesta doctoral.

3.1 MÉTODOS DE INVESTIGACIÓN

Diseñar un modelo en el contexto de la ciencia del diseño requiere un enfoque riguroso, que combine métodos creativos, técnicos y empíricos para garantizar que el resultado sea funcional, relevante y científicamente fundamentado. A continuación, se detallan los métodos de investigación clave que se emplearon durante el proceso de diseño del modelo:

3.1.1 INVESTIGACIÓN EXPLORATORIA

En la fase de identificación del problema se realizaron revisiones de literatura y una encuesta a expertos para definir y comprender el problema que el modelo debe resolver.

3.1.2 REVISIÓN SISTEMÁTICA DE LA LITERATURA

Para la revisión de la literatura se buscaron investigaciones referidas al tema en bases de datos científicas como Scopus, ACM y la IEEE Xplore que permitieron identificar modelos existentes y establecer el estado del arte.

3.1.3 PROTOTIPADO

En esta fase se proponen y se construyen las versiones iniciales del modelo:

Originalmente se diseñó un modelo concebido con la idea de automatizar las distintas fases del desarrollo de una aplicación. Los resultados del proceso se publicaron en la revista internacional "CSSE-Computer Systems Science and Engineering", en el artículo "Automation of Software Development Stages with the OpenAI API", [50].

Posteriormente se propone una nueva alternativa para el modelo, el proceso de creación se describe en la sección 3.2.

3.1.4 EXPERIMENTOS CONTROLADOS

Para la validación y evaluación se diseñaron experimentos que permitieron medir la eficacia y eficiencia del modelo, esto se hizo a través de pruebas con casos de estudio. Las pruebas permitieron contrastar el modelo con otras propuestas existentes para evaluar métricas de dependencia en aplicaciones basadas en microservicios.

3.1.5 ESTUDIO DE CASO

El estudio de caso fue la estrategia de validación y evaluación del modelo propuesto, para ello, se utilizaron cuatro casos de estudio: Cargo Tracking, JpetStore, Bank of Anthos y Hipster Shop.

3.2 PROCESO DE CREACIÓN DEL MODELO INTELIGENTE PARA ESPECIFICAR DEPENDENCIA Y COMPLEJIDAD EN EL DISEÑO DE APLICACIONES BASADAS EN MICROSERVICIOS

3.2.1 ATRIBUTOS, MÉTRICAS, HERRAMIENTAS Y DESAFÍOS DE LA EVALUACIÓN DE LA CALIDAD DE MICROSERVICIOS

En esta sección se sustenta la primera pregunta de investigación:

- PI.1. Qué desafíos y problemas se presentan en la evaluación de la calidad de los microservicios, asociados a los atributos, métricas y herramientas aplicables a la medición de la calidad de este tipo de software.

A través de una revisión sistemática de la literatura se analizó la calidad de los microservicios (MS), el objetivo principal fue identificar los atributos y métricas clave, así como, los retos investigativos asociados a este campo. Los resultados de esta investigación se publicaron en la revista internacional "Journal of Advances in Information Technology", en el artículo denominado Research Opportunities in Microservices Quality Assessment: A Systematic Literature Review" [27].

Para el estudio bibliográfico se plantearon cinco preguntas (Research Questions - RQ):

- **RQ1.** ¿Cuáles son los atributos de calidad aplicables a los microservicios?
- **RQ2.** ¿Qué métricas de calidad son aplicables a los microservicios?
- **RQ3.** ¿Qué desafíos se han identificado en la evaluación de la calidad de los microservicios?
- **RQ4.** ¿Cuáles son las herramientas más utilizadas para evaluar microservicios?
- **RQ5.** ¿Qué métricas emplean estas herramientas?

3.2.1.1 *Proceso de Búsqueda*

El proceso de revisión se realizó basado en una adaptación de la metodología propuesta por Barbara Kitchenham [51]. Se ejecutaron las revisiones, tomando en cuenta que son cinco preguntas de investigación, a partir de la etapa 4 el proceso se realizó de forma iterativa por interrogante (Figura 4).

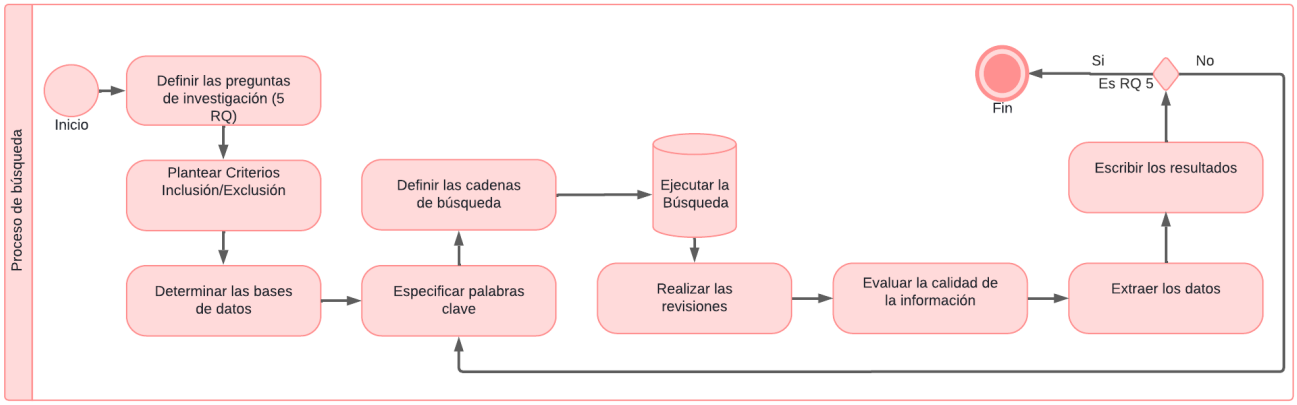


Figura 4: Proceso de revisión sistemática de la literatura. Fuente: [27]

Criterios de inclusión y exclusión

En los criterios de inlusión se plantean los principios requeridos para considerar fuentes válidas, relevantes y actualizadas. El rango de antigüedad de las publicaciones es de cinco años, se observa el número de citas y se toman en cuenta solo las versiones completas de los artículos.

Se excluyen del estudio a la literatura gris (reportes técnicos, resúmenes y presentaciones) y los artículos no relacionados con la calidad del software y los microservicios (Tabla 3).

Tabla 3. Criterios de inclusión y exclusión

CRITERIOS DE INCLUSIÓN	CRITERIOS DE EXCLUSIÓN
IC1: Artículos publicados en los últimos cinco (5) años (2017-2021). IC2: Artículos publicados que tienen el mayor número de citaciones. IC3: Si un artículo describe más de un estudio, cada estudio es evaluado individualmente. IC4: Si hay versiones corta y completa	EC1: Reportes técnicos y documentos que están disponibles en forma de resúmenes o presentaciones literatura gris). EC2: Artículos que no presentan estudios relacionados con software y microservicios.

3.2.1.2 *Resultados de la revisión sistemática de la literatura:*

La búsqueda generó 965 fuentes, luego de la evaluación de la calidad se obtuvieron 163 válidas. Las bases de datos en donde se ejecutaron las búsquedas fueron ACM, IEEE XPLORE, SCIENCEDIRECT, SCOPUS, SPRINGER, y WEB of SCIENCE (WOS), (Figura 5).

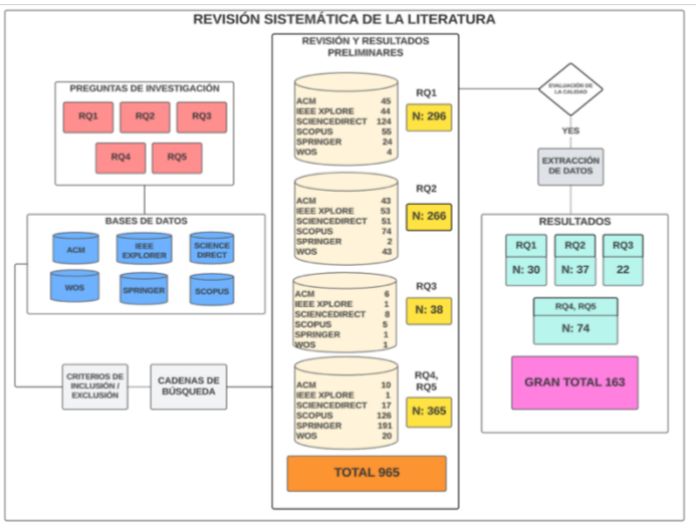


Figura 5: Resultados del proceso de revisión sistemática. Fuente: [27]

- **Atributos aplicables a microservicios**

Se consideraron 30 de 296 publicaciones, los resultados indican que los atributos aplicables a microservicios que con mayor frecuencia se reportan son el rendimiento, escalabilidad, seguridad, mantenibilidad y acoplamiento; mientras que, con menor frecuencia están la cohesión, granularidad, fiabilidad, elasticidad, flexibilidad, complejidad e interoperabilidad (Tabla 4, Figura 6).

Tabla 4. Resultados LRS – Atributos de microservicios

Atributo	Nº	Referencia
Rendimiento	14	[1], [6], [11], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62]
Escalabilidad	13	[1], [11], [52], [53], [54], [55], [57], [58], [60], [62], [63], [64], [65]
Seguridad	10	[1], [52], [53], [57], [58], [61], [62], [63], [64], [66], [67]
Mantenibilidad	9	[1], [6], [11], [53], [56], [59], [68], [69], [70]
Acoplamiento	8	[9], [52], [53], [55], [68], [69], [71], [72]

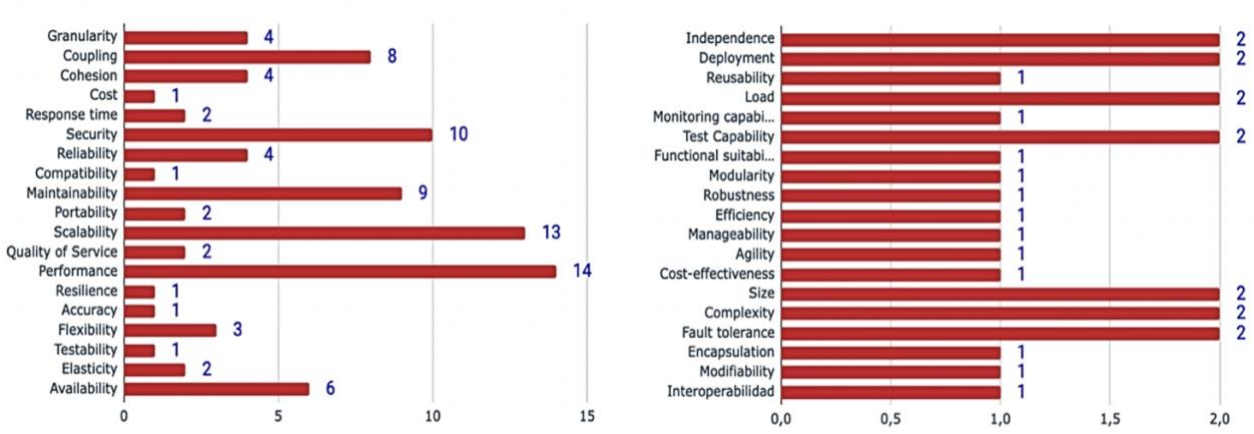


Figure 6. Resultados del proceso de búsqueda – Pregunta 1 (RQ1). Fuente [27]

- **Métricas de calidad de microservicios**

En el caso de la pegunta 2, se consideraron 37 de 266 fuentes bibliográficas. Según la LRS, las métricas aplicables a microservicios son la cohesión, acoplamiento, granularidad, rendimiento y complejidad (Tabla 5, Figura 7).

Tabla 5. Resultados LRS – Métricas de microservicios

Métricas	No	Referencias
Rendimiento	14	[53], [66], [67], [69], [70], [71], [72], [73], [74], [75]
Cohesión	8	[53], [59], [74], [76], [77], [78], [79], [80]
Acoplamiento	8	[53], [70], [73], [76], [77], [81], [82], [83]
Granularidad	4	[9], [70], [75], [76]
Complejidad	4	[53], [61], [77], [83]

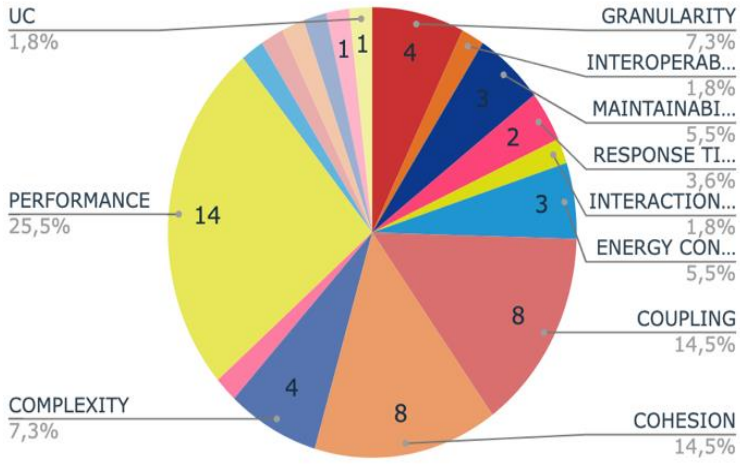


Figure 7. Resultados del proceso de búsqueda – Pregunta 2 (RQ2). Fuente [27]

- **Desafíos de investigación en la evaluación la calidad de los microservicios**

Se seleccionaron 22 fuentes de las 38 identificadas. En mayor medida se reportan como desafíos a la seguridad, granularidad, rendimiento, monitorización, estrategia organizativa, orquestación, coreografía, escalabilidad, descomposición y refactorización de monolitos (Figura 8).

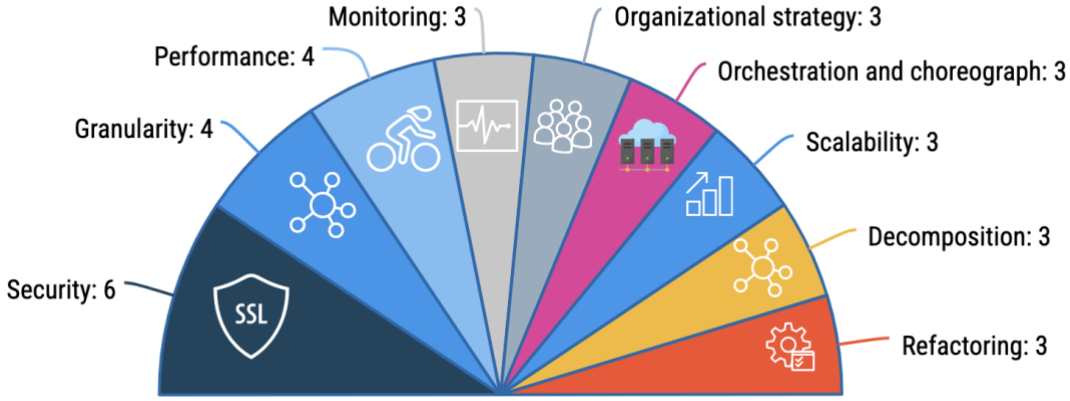


Figure 8. Resultados del proceso de búsqueda – Pregunta 3 (RQ3). Fuente [27]

- **Herramientas de evaluación de microservicios**

Se reportan varias herramientas de evaluación de microservicios, la mayoría relacionadas con el atributo de rendimiento. De las 365 fuentes encontradas se seleccionaron 74. La generalidad de los estudios se centra en implementar o investigar una única herramienta con la que realizan las evaluaciones y aplican las métricas de calidad. En este contexto, se podrían citar más de 70 herramientas diferentes, a continuación, en la figura 9 se citan algunos ejemplos y en la tabla 6 se pueden apreciar las referencias.

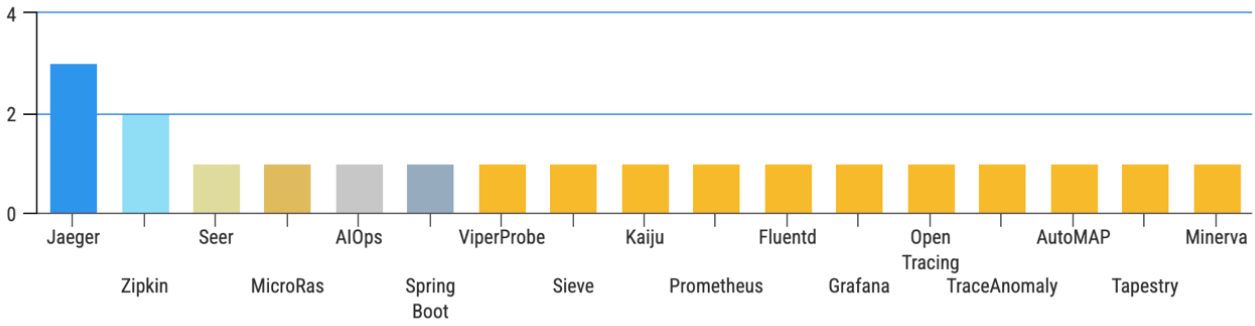


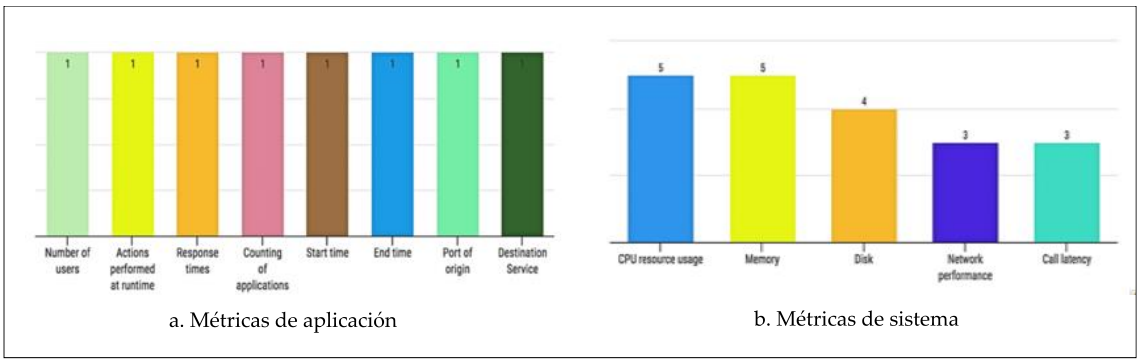
Figure 9. Resultados del proceso de búsqueda – Pregunta 4 (RQ4). Fuente [27]

Tabla 6. Herramientas de evaluación de microservicios

Herramienta	No	Referencias
Jaeger	3	[84], [85], [86]
Zipking	2	[84], [85]
Grafana	1	[87]
Kaiju	1	[85]
Prometheus	1	[85]
Opentracing	1	[85]
Aroma	1	[88]
MAAT	1	[89]
MSA Nose+	1	[90]
Prevant	1	[91]
Docker Compose Rule	1	[92]

- **Métricas de evaluación que utilizan las herramientas**

A partir de la búsqueda realizada para la pregunta 4, se obtuvieron también los resultados para la pregunta 5. Se identificaron 74 fuentes y se determinó que las herramientas utilizan dos tipos de métricas: a) métricas de aplicación como el número de usuarios, tiempos de respuesta, horas de inicio y finalización y, b) métricas de sistema como el uso de recursos de CPU, usos de memoria y usos de disco (Figura 10).



a. Métricas de aplicación

b. Métricas de sistema

Figure 10. Resultados del proceso de búsqueda – Pregunta 5 (RQ5). Fuente [27]

A través de la revisión de la literatura se analizaron los principales desafíos reportados en el ámbito de los microservicios, incluyendo seguridad, granularidad, rendimiento, monitoreo, estrategia organizacional, orquestación, coreografía, escalabilidad, descomposición y refactorización de monolitos. Como oportunidades de investigación, se proponen estudios empíricos sobre tácticas específicas para el desarrollo de microservicios, análisis de los compromisos entre distintas garantías de calidad y una ampliación continua de revisiones que integren nuevas herramientas y perspectivas.

Asimismo, se enfatiza la carencia de una definición clara sobre la interrelación entre los atributos de calidad, métricas y patrones. También se identifica la necesidad de desarrollar evaluaciones automáticas, realizar análisis sobre el impacto de las decisiones arquitectónicas, implementar diseños basados en dominios y mejorar aspectos como las pruebas, monitoreo y el modelado de rendimiento.

Finalmente, se destacan desafíos adicionales como la orquestación, gestión de la comunicación, consistencia entre microservicios, evolución independiente y la escalabilidad, subrayando la importancia de abordar estos aspectos para fortalecer la calidad en arquitecturas basadas en microservicios..

3.2.2 DEPENDENCIA Y COMPLEJIDAD EN EL DISEÑO DE MICROSERVICIOS: UN ENFOQUE INTEGRAL

En esta sección se sustenta la segunda pregunta de investigación:

- PI.2. Cuál es la relación que se presenta entre la dependencia y la complejidad de las arquitecturas de los microservicios con los problemas de flexibilidad y escalabilidad de los mismos.

Se examinó de manera detallada la relación entre dependencia y complejidad dentro de las arquitecturas basadas en microservicios, haciendo especial énfasis en cómo impactan en la flexibilidad y escalabilidad de los sistemas. Los resultados de esta parte de la ivestigación se publicaron en la revista internacional "Revista Ibérica de Sistemas e Tecnologias de Informação", en el artículo "Evaluando Dependencia y Complejidad en el Diseño de Microservicios: Un Enfoque Integral", [93].

Para abordar esta cuestión, se llevó a cabo una revisión exhaustiva de la literatura complementada con una encuesta dirigida a profesionales del campo del desarrollo de microservicios con el fin de comprender cómo estos factores influyen en la flexibilidad y escalabilidad de dichas arquitecturas.

Los hallazgos de la revisión de literatura revelan que una proporción significativa de los participantes enfrenta dificultades relacionadas con la dependencia (72.7%) y la complejidad (63.6%), lo que repercute negativamente en actividades esenciales como el mantenimiento, el despliegue y la detección de fallos. En particular, el mantenimiento se identificó como el aspecto más afectado (68.2%), seguido por la escalabilidad (63.6%) y el rendimiento del sistema (54.5%) [93].

El estudio subraya la necesidad de abordar estos retos con un enfoque equilibrado que permita gestionar las interdependencias sin comprometer la flexibilidad del sistema. Para lograrlo, es fundamental implementar un diseño arquitectónico riguroso, acompañado de una planificación meticulosa, una documentación comprensiva y una gestión eficiente de las dependencias. Esto garantizará niveles óptimos de mantenibilidad y escalabilidad en las arquitecturas de microservicios. En síntesis, la interacción entre la dependencia

y la complejidad constituye un factor determinante para la flexibilidad y la escalabilidad, lo que demanda un enfoque sistemático y equilibrado para maximizar los beneficios inherentes a este paradigma arquitectónico.

En los siguientes apartados de esta sección, se presentan las descripciones de los resultados más significativos de esta parte de la investigación:

- **Dependencia**

La dependencia en software se refiere al grado en que un componente del sistema necesita la colaboración de otro para funcionar. Un alto nivel de acoplamiento indica que un elemento no puede operar correctamente sin otro; por lo tanto, cualquier modificación en un componente requiere ajustes en los módulos que dependen de el [94]. Según [95], las dependencias entre microservicios son esenciales para su operación dentro de un sistema distribuido, ya que un microservicio necesita de otro para completar tareas específicas. Sin embargo, estas interrelaciones pueden causar problemas: si un microservicio falla o se actualiza, puede impactar a otros, generando un efecto dominó que afecta el rendimiento y la estabilidad general del sistema. Existen diferentes tipos de dependencias, como las llamadas de red, las dependencias de datos y las interacciones entre microservicios. Es crucial distinguir entre dos categorías principales: las dependencias cíclicas, donde los microservicios se invocan mutuamente, y las dependencias paralelas, donde múltiples microservicios acceden simultáneamente a los mismos servicios.

- **Complejidad**

La complejidad en la arquitectura de microservicios es tanto inherente como fundamental, ya que implica descomponer aplicaciones en componentes independientes. Esta independencia puede beneficiar o dificultar la flexibilidad,

que es crucial para adaptarse a cambios y ajustarse a entornos dinámicos, al integrar diversas funcionalidades [96]. Aunque la segmentación puede simplificar teóricamente las tareas y facilitar el desarrollo, existe un límite delicado: si se lleva la fragmentación al extremo, los microservicios pueden presentar desafíos significativos debido a las interdependencias entre ellos, lo que puede llevar a situaciones insostenibles. Esto pone en riesgo la escalabilidad, que en el contexto de los microservicios se clasifica en dos tipos: horizontal y vertical, ambos afectados por cambios en la demanda que requieren estrategias de gestión efectivas [52].

- **Relación entre dependencia y escalabilidad.**

El análisis sintetizado en la Tabla 7 revela la compleja relación entre la dependencia y la escalabilidad en arquitecturas de microservicios, destacando elementos teóricos y prácticos fundamentales. Según la Ley de Conway, los sistemas diseñados por organizaciones tienden a reflejar las dinámicas internas de estas, lo que, en el contexto de los microservicios, implica que las dependencias entre ellos pueden ocasionar desafíos significativos en términos de ajuste y evolución arquitectónica [97] [52].

La escalabilidad, como un atributo esencial de las aplicaciones modernas, permite que los microservicios se adapten dinámicamente a las variaciones en la carga de trabajo, mientras que las dependencias bien gestionadas fomentan la reutilización de componentes, fortaleciendo la modularidad del sistema [52]. No obstante, para garantizar una escalabilidad efectiva, se requiere un diseño caracterizado por un acoplamiento débil y una minimización de las dependencias interservicios, ya que estas características reducen los riesgos asociados con la propagación de errores y la degradación del rendimiento [98].

En este contexto, la capacidad de escalar servicios de manera independiente adquiere un papel central, particularmente en aplicaciones sometidas a altos volúmenes de tráfico, dado que permite una optimización precisa de recursos sin comprometer la estabilidad del sistema global [5][99][100]. Este equilibrio entre dependencia y escalabilidad constituye un aspecto clave para maximizar la eficiencia y adaptabilidad de los microservicios en entornos dinámicos y de alta demanda.

Tabla 7. Relación dependencia y escalabilidad en microservicios

Aspecto	Descripción	Referencias
Ley de Conway	Los sistemas diseñados por organizaciones tienden a reflejar las dinámicas internas de estas, lo que, en el contexto de los microservicios, implica que las dependencias entre ellos pueden ocasionar desafíos significativos en términos de ajuste y evolución arquitectónica.	[48]
Impacto de la dependencia en los microservicios	La dependencia entre servicios puede afectar a otros, generando desafíos para realizar ajustes.	[29]
Escalabilidad independiente	Cada servicio puede escalarse individualmente, lo que es esencial en aplicaciones modernas con alto tráfico.	[50]; [51]; [5]
Condiciones para escalabilidad	La escalabilidad depende de un acoplamiento débil y dependencias mínimas.	[49]

- **Relación entre dependencia y flexibilidad**

La relación entre dependencia y flexibilidad en las arquitecturas de microservicios es un aspecto crítico que influye significativamente en su calidad, escalabilidad y mantenibilidad. Una alta dependencia puede restringir la capacidad de escalar y actualizar los sistemas, mientras que una excesiva flexibilidad tiende a complicar el mantenimiento y la depuración, aumentando la complejidad general del sistema. Este equilibrio, aunque desafiante, puede

lograrse mediante el uso de patrones de diseño específicos como el Front Controller o Gateway, que fomentan la agilidad y estabilidad de los servicios.

La gestión adecuada de las dependencias no solo mejora la flexibilidad durante el desarrollo e implementación, sino que también permite abordar de manera eficaz los desafíos inherentes al diseño y mantenimiento de los microservicios. Este proceso, sin embargo, requiere una comunicación y colaboración constante entre los desarrolladores, lo que asegura una comprensión clara de las interrelaciones y fomenta la resiliencia del sistema. Por tanto, al adoptar un enfoque integrado que contemple la dependencia y flexibilidad, las organizaciones pueden diseñar arquitecturas más robustas, adaptables y sostenibles, optimizando el desempeño general y la resiliencia frente a cambios.

Tabla 8. Relación entre dependencia y flexibilidad

Aspecto	Descripción	Referencias
Impacto de la dicotomía	La interacción entre dependencia y flexibilidad influye directamente en la calidad de los sistemas de microservicios, afectando escalabilidad, mantenimiento y depuración.	[52]; [53]
Efectos de alta dependencia	Limita la escalabilidad y actualización del sistema, generando ramificaciones en cascada que comprometen estabilidad e integridad.	[53]
Efectos de excesiva flexibilidad	Complica el mantenimiento y la depuración, dificultando la identificación de relaciones entre microservicios.	[54]
Recomendaciones de diseño	Se sugiere emplear patrones como Front Controller o Gateway para equilibrar agilidad y estabilidad en la arquitectura.	[29]
Gestión eficaz de dependencias	La adecuada gestión de dependencias incrementa la flexibilidad en el desarrollo e implementación de los servicios.	[55]
Colaboración entre desarrolladores	Comunicación efectiva entre desarrolladores es crucial para lograr sistemas escalables, resilientes y eficientes.	[49]
Enfoque integrado	Un diseño consciente de las interdependencias permite superar los retos inherentes y asegurar el éxito de las arquitecturas de microservicios.	[49]; [56]

- **Relación entre complejidad y flexibilidad**

La relación entre complejidad y flexibilidad es un eje central en el diseño de arquitecturas basadas en microservicios. Mientras la flexibilidad se posiciona como un requisito fundamental para la adaptabilidad ante cambios constantes en los entornos tecnológicos, la complejidad actúa como una fuerza tanto restrictiva como habilitadora. Una complejidad excesiva puede limitar severamente la flexibilidad, incrementando los costos de escalabilidad y dificultando la gestión integral del sistema. Por el contrario, una complejidad insuficiente puede generar un diseño deficiente, caracterizado por una baja cohesión y una falta de coordinación entre los servicios, lo que socava la eficiencia operativa.

La implementación de mecanismos de comunicación y coordinación robustos se identifica como un elemento crítico para mitigar los impactos adversos de la complejidad, asegurando una integración eficiente y minimizando la posibilidad de errores en la interacción entre servicios. Además, la escalabilidad y la facilidad de gestión del sistema surgen como factores determinantes para garantizar que la flexibilidad inherente a los microservicios se traduzca en beneficios reales, como una mayor adaptabilidad y capacidad de respuesta.

En última instancia, el diseño exitoso de microservicios exige un equilibrio meticuloso entre complejidad y flexibilidad. Este balance permite a los desarrolladores crear sistemas que no solo sean resilientes y escalables, sino también eficientes y fáciles de mantener, alineándose con las demandas de las aplicaciones modernas y los entornos altamente dinámicos.

Tabla 9. Relación entre complejidad y flexibilidad

Aspecto	Descripción	Referencias
Importancia de la flexibilidad	Es fundamental para adaptarse a los cambios en el entorno y los requerimientos del sistema.	[57]
Impacto de la complejidad excesiva	Puede restringir la flexibilidad, dificultando la escalabilidad y gestión del sistema.	[57]
Riesgos de baja complejidad	Resulta en una falta de cohesión y coordinación entre los servicios, lo que afecta la eficiencia del sistema.	[57]
Coordinación y comunicación	Implementar mecanismos efectivos de comunicación entre servicios es clave para garantizar la integración y funcionalidad del sistema.	[53]
Escalabilidad y gestión	La escalabilidad y facilidad de gestión son factores esenciales para mantener la adaptabilidad y rendimiento en sistemas complejos.	[54]
Equilibrio entre complejidad y flexibilidad	Alcanzar un balance adecuado permite el desarrollo de arquitecturas adaptables, eficientes y cohesivas.	[57]; [5]

- **Relación entre complejidad y escalabilidad**

La relación entre complejidad y flexibilidad es un eje central en el diseño de arquitecturas basadas en microservicios. Mientras la flexibilidad se posiciona como un requisito fundamental para la adaptabilidad ante cambios constantes en los entornos tecnológicos, la complejidad actúa como una fuerza tanto restrictiva como habilitadora. Una complejidad excesiva puede limitar severamente la flexibilidad, incrementando los costos de escalabilidad y dificultando la gestión integral del sistema. Por el contrario, una complejidad insuficiente puede generar un diseño deficiente, caracterizado por una baja cohesión y una falta de coordinación entre los servicios, lo que socava la eficiencia operativa.

La implementación de mecanismos de comunicación y coordinación robustos se identifica como un elemento crítico para mitigar los impactos adversos de la complejidad, asegurando una integración eficiente y minimizando la posibilidad de errores en la interacción entre servicios. Además, la escalabilidad y la facilidad

de gestión del sistema surgen como factores determinantes para garantizar que la flexibilidad inherente a los microservicios se traduzca en beneficios reales, como una mayor adaptabilidad y capacidad de respuesta.

En última instancia, el diseño exitoso de microservicios exige un equilibrio meticuloso entre complejidad y flexibilidad. Este balance permite a los desarrolladores crear sistemas que no solo sean resilientes y escalables, sino también eficientes y fáciles de mantener, alineándose con las demandas de las aplicaciones modernas y los entornos altamente dinámicos.

Tabla 10. Relación entre complejidad y escalabilidad

Aspecto	Descripción	Referencias
Importancia de la flexibilidad	Es fundamental para adaptarse a los cambios en el entorno y los requerimientos del sistema.	[5]
Impacto de la complejidad excesiva	Puede restringir la flexibilidad, dificultando la escalabilidad y gestión del sistema.	[49]
Riesgos de baja complejidad	Resulta en una falta de cohesión y coordinación entre los servicios, lo que afecta la eficiencia del sistema.	[49]
Coordinación y comunicación	Implementar mecanismos efectivos de comunicación entre servicios es clave para garantizar la integración y funcionalidad del sistema.	[43]
Escalabilidad y gestión	La escalabilidad y facilidad de gestión son factores esenciales para mantener la adaptabilidad y rendimiento en sistemas complejos.	[44]
Equilibrio entre complejidad y flexibilidad	Alcanzar un balance adecuado permite el desarrollo de arquitecturas adaptables, eficientes y cohesivas.	[49];

3.2.2.1 *Encuesta aplicada a profesionales con experiencia en la implementación de microservicios.*

La encuesta se aplicó a un conjunto de 22 profesionales del desarrollo de aplicaciones basadas en microservicios. El instrumento compuesto por 14 preguntas, indaga sobre el tiempo de experiencia del encuestado, el manejo de la calidad de los microservicios y, sobre todo, investiga sobre los problemas identificados a causa de arquitecturas dependientes y complejas, se enfoca en

problemas que atacan a los atributos de escalabilidad y flexibilidad de microservicios (Anexo 1).

En los siguientes apartados, se describen los resultados más relevantes:

- **Técnicas utilizas para el diseño de aplicaciones basadas en microservicios**

El 31.8% de los encuestados señala utilizar el Modelado de Procesos de Negocio como técnica para el diseño de aplicaciones basadas en microservicios. Las técnicas de Diagramas de Flujo de Datos y Diseño Dirigido por el Dominio, cada una tiene el 27.3%, el 13.6% responde que no usan ninguna técnica, finalmente, la técnica del Modelo C4, tiene un 0%.

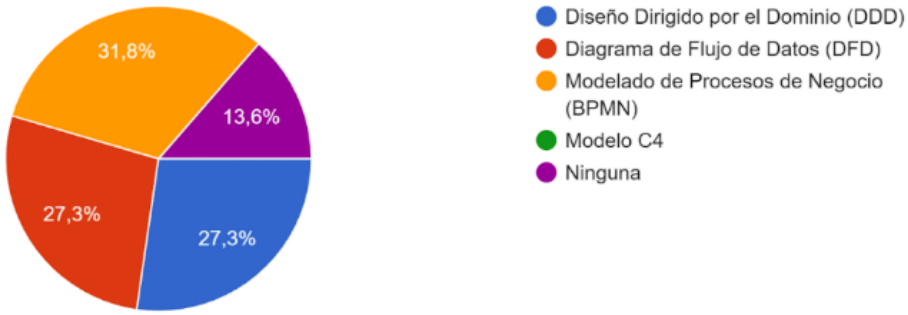


Figure 11. Técnicas utilizadas para el diseño de microservicios. Fuente [93]

- **Atributos de calidad afectados por arquitecturas dependientes y complejas**

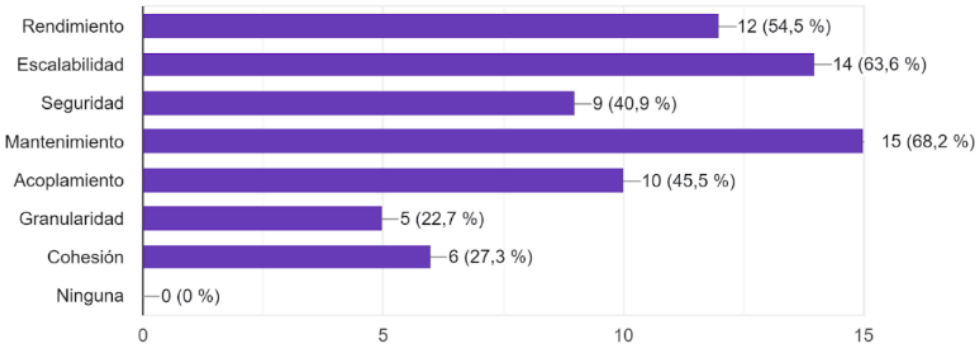


Figura 12. Atributos de calidad afectados por arquitecturas dependientes y complejas. Fuente [93]

En esta pregunta los participantes podían escoger varias respuestas, es así que, el 68.2% de profesionales menciona que el mantenimiento es el atributo más afectado por las arquitecturas dependientes y complejas, sigue la escalabilidad con un 63.6%, luego el rendimiento con el 54.5%, sigue el acoplamiento con el 45.5%, la seguridad con el 40.9%, la cohesión con el 27.3% y con el 22.7% la granularidad.

- **Problemas de dependencia en arquitecturas de microservicios**

El 72.7% de participantes han identificado problemas de dependencia al desarrollar microservicios, el 27.3% no ha tenido problemas en el desarrollo de ete tipo de software.

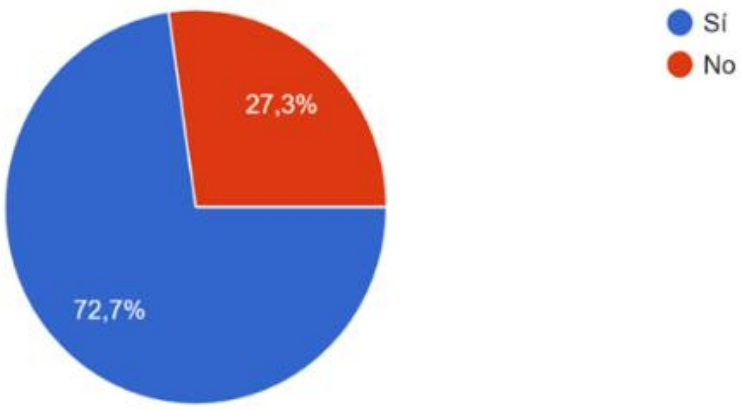


Figura 13. Problemas de dependencia en arquitecturas de microservicios. Fuente [93]

Los problemas que se reportan son la flexibilidad, dificultad para detectar el origen de errores en componentes, análisis, complejidad en el proceso de despliegue, tiempo adicional al unir información de distintos repositorios, dependencia de volúmenes de almacenamiento NFS y posibles problemas al realizar cambios sin un estándar de comunicación entre operaciones.

- **Problemas de complejidad en arquitecturas de microservicios**

El 63.6% de participantes sí han identificado problemas de complejidad al desarrollar microservicios, el 36.4% menciona que no han tenido problemas de complejidad.

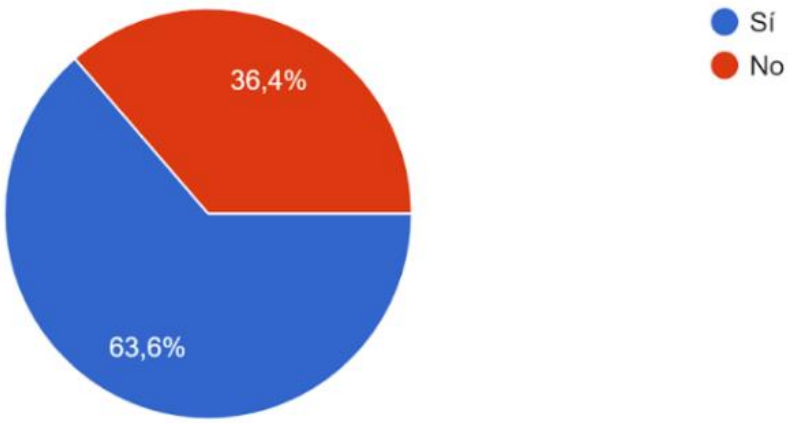


Figura 14. Problemas de complejidad en arquitecturas de microservicios. Fuente [93]

Los problemas de complejidad reportados son mantenibilidad, dificultad en la interoperabilidad de componentes, complejidad en la gestión y coordinación de equipos de trabajo, limitaciones de infraestructura o interconexión, mayor tiempo para procesos transaccionales, necesidad de coordinadores y mayor comunicación en red.

- **Métricas de calidad que utilizan las empresas para evaluar la calidad de microservicios**

Las métricas de calidad que se reportan son el rendimiento con el 72.7%, la seguridad con el 63.6%, la escalabilidad con el 45.5%, el acoplamiento y la granularidad con el 27.3% cada una y, con el 18.2%, cohesión y personas que no usan ninguna métrica de calidad.

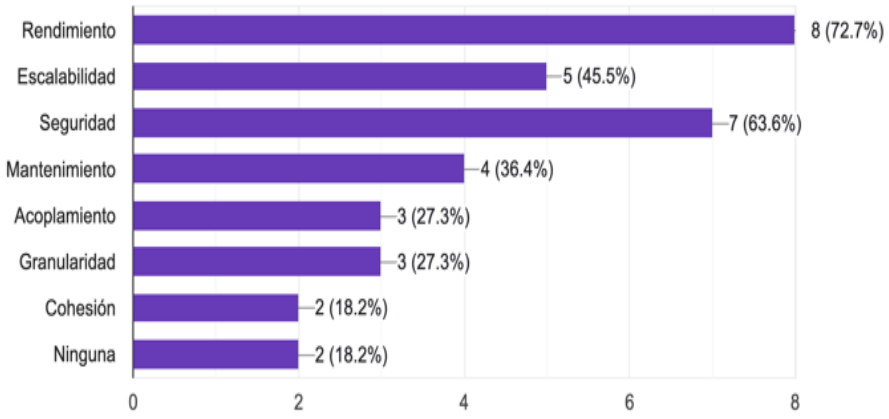


Figura 15. Métricas de calidad que utilizan las empresas. Fuente [93]

- **Desafíos que se presentan en la evaluación de la calidad de los microservicios**

Los desafíos mencionados son el rendimiento con el 63.6%, seguridad el 54.5%, diseño el 45.5%, monitoreo el 36.4%, escalabilidad el 27.3%, para finalizar, la granularidad el 18.2%.

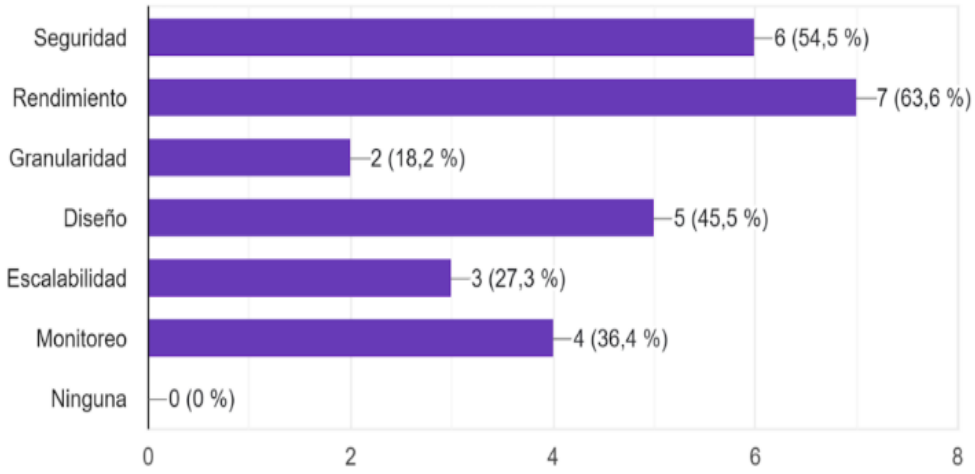


Figura 16. Desafíos que se presentan en la evaluación de la calidad de los microservicios. Fuente [93]

- **Herramientas de modelado utiliza para el diseño de aplicaciones basada en microservicios**

Las herramientas de modelado que más se mencionan son Draw.io y Lucid chart, 11 encuestados no responden esta pregunta.

La información obtenida de la encuesta subraya la relevancia de asegurar la calidad en el diseño de arquitecturas de microservicios, destacando que esta calidad se evalúa desde múltiples perspectivas. Los parámetros fundamentales incluyen mantenimiento, escalabilidad, rendimiento y seguridad. Estudios indican que un diseño efectivo de microservicios debe equilibrar estos aspectos, desafiando la noción de que su implementación perjudica el desarrollo. No obstante, se observa que una flexibilidad excesiva puede generar complicaciones que obstaculizan el logro de objetivos, lo que resalta la necesidad de establecer criterios claros para gestionar dicha flexibilidad.

El análisis revela que el rendimiento, la escalabilidad y la dependencia son factores críticos en estas arquitecturas. Los encuestados identificaron problemas relacionados con la dependencia (72.7%), tales como flexibilidad, detección de errores, despliegue y seguridad. En cuanto a los problemas de complejidad (63.6%), se mencionaron la mantenibilidad, la coordinación entre equipos, la infraestructura y la falta de documentación. La calidad en los microservicios se define por la capacidad de equilibrar flexibilidad y dependencia, minimizando los riesgos asociados con la complejidad.

En conclusión, se enfatiza la necesidad de un diseño riguroso, patrones específicos y una gestión adecuada de las interdependencias para maximizar los beneficios de estas arquitecturas. Se sugiere el desarrollo de una herramienta inteligente que asista automáticamente en la fase de diseño, ayudando a los equipos técnicos a prevenir problemas arquitectónicos que puedan comprometer

la flexibilidad y escalabilidad de los microservicios. Este enfoque busca abordar desafíos críticos y mejorar la evaluación de la calidad en este ámbito emergente de la ingeniería de software.

3.2.3 ESPECIFICACIÓN DEL MODELO INTELIGENTE

El modelo inteligente para especificar dependencia y complejidad en el diseño de aplicaciones de microservicios pretende brindar apoyo a los equipos de desarrollo, en la toma de deciciones de carácter arquitectónico para prevenir futuros problemas de implementación, mantenimiento, flexibilidad y escalabilidad.

El modelo denominado "MOAM" (Microservices Optimization Aquitectures Model), se basa en la implementación de algoritmos inteligentes que a partir de las entradas que son las historias de usuario de un caso, genera el diagrama arquitectrónico de los microservicios. Posteriormente, utiliza una serie de métricas para calcular de manera automática la dependencia y complejidad de la arquitectura, finaliza con el reporte de los cálculos correspondientes y una serie de recomendaciones que fueron determinadas a través de umbrales definidos en los resultados de las diferentes métricas.

3.2.3.1 *Descripción abstracta del modelo*

El modelo se enfoca en la generación y evaluación de la calidad de arquitecturas de aplicaciones basadas en microservicios, a través de la integración de análisis funcional, especificación arquitectónica y evaluación métrica, el modelo debe permitir: crear arquitecturas desde un nivel conceptual inicial, facilitar decisiones informadas durante la evolución y mantenimiento de

la arquitectura, proveer retroalimentación continua para optimizar la calidad de diseño en términos de flexibilidad y escalabilidad.

Se definen dos características principales: la primera, relacionada con la generación automática de las arquitecturas y, la segunda, relacionada con la evaluación de la calidad a través del cálculo de un conjunto de métricas de dependencia y complejidad (Figura 17).

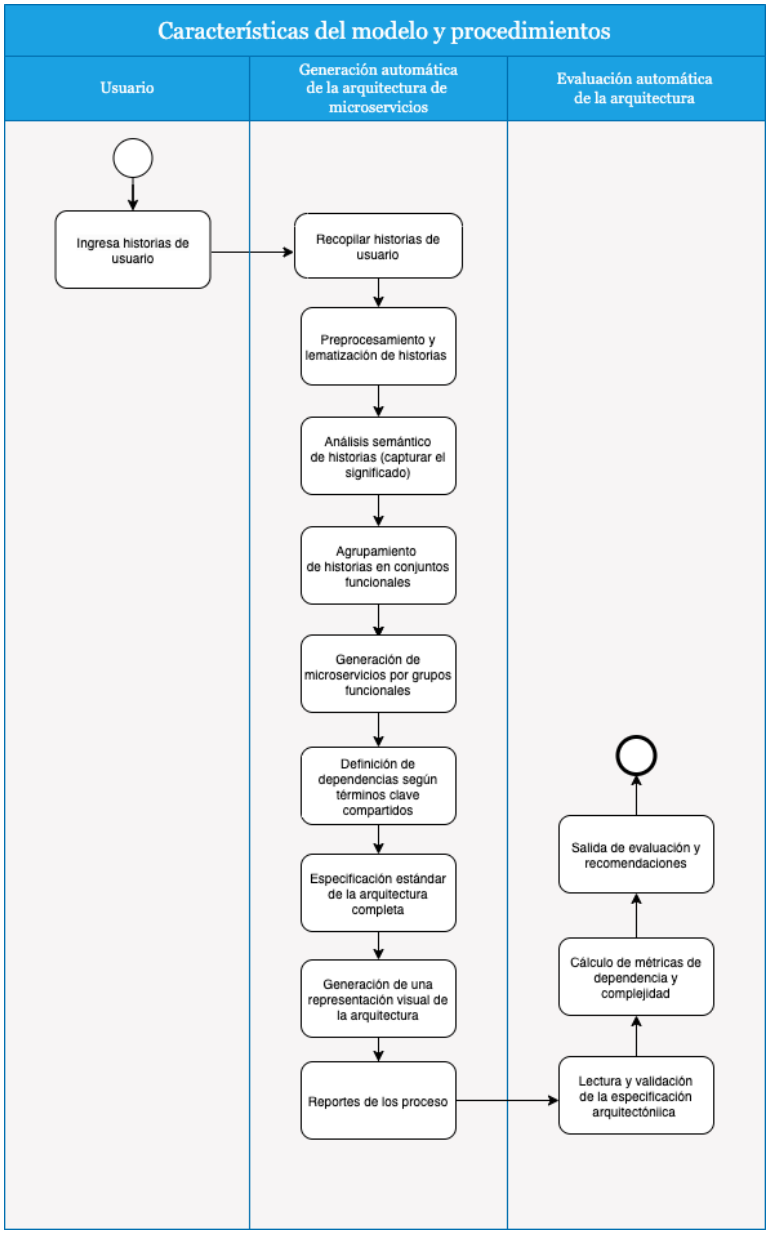


Figura 17. Características del modelo y procedimientos

En los siguientes apartados se explica cada una de las características:

- **Generación automática de la arquitectura**

A partir de las descripciones funcionales de un caso, expresadas en forma de historias de usuario, como insumos para identificar y agrupar funcionalidades relacionadas, se beben emplear técnicas avanzadas de análisis semántico y métodos de agrupamiento para determinar dinámicamente los dominios funcionales más adecuados. El resultado debe ser una especificación arquitectónica que:

  - Asigna las funcionalidades a microservicios, asegurando cohesión interna y autonomía.
  - Identifica dependencias internas y externas, minimizando acoplamientos innecesarios.
  - Genera diagramas arquitectónicos visuales que representan las relaciones y dependencias entre los microservicios.
  - Produce una representación estructurada y estandarizada de la arquitectura generada, adecuada para análisis posteriores.

- **Pasos para generar la arquitectura:**

Paso 1: Ingreso de historias de usuario. El usuario ingresa las historias de un caso o proceso, las historias describen las funcionalidades o requerimientos del caso en lenguaje natural.

Paso 2: Recopilar historias de usuario. Se recopilan historias de usuario como entradas al sistema.

Paso 3: Preprocesamiento y lematización de historias. Cada historia se preprocesa para eliminar elementos irrelevantes como las palabras vacías y los signos de puntuación, luego es necesario normalizar mediante técnicas como

lematización. El objetivo de este paso es garantizar que las historias estén en un formato estandarizado y adecuado para realizar análisis semántico.

Paso 4: Análisis semántico de historias de usuario. Se deben utilizar técnicas de procesamiento de lenguaje natural para capturar el significado de las historias y realizar el análisis semántico, el proceso consiste en descomponer y examinar su contenido lingüístico para entender con precisión su significado, intención, y las necesidades que describen.

Paso 5: Agrupamiento de las historias de usuario. En este paso es necesario emplear algoritmos de agrupamiento para identificar relaciones entre las historias y organizarlas en clusters, el número de clusters se determina dinámicamente para maximizar métricas de calidad. El resultado de este proceso debe ser las historias agrupadas en conjuntos funcionales que representarán dominios lógicos dentro del sistema.

Paso 6: Generación de dominios y microservicios. Usando técnicas de procesamiento de lenguaje natural, a cada grupo funcional se le asigna un nombre representativo basado en palabras clave extraídas de las historias de usuario. Los grupos funcionales se transforman en microservicios, con cada microservicio representando un dominio lógico independiente.

Paso 7: Definición de las dependencias. Se definen las relaciones iniciales de dependencia entre microservicios basándose en los recursos o términos clave compartidos. El resultado debe ser un conjunto de microservicios definidos con funcionalidades bien agrupadas y relaciones iniciales identificadas.

Paso 8: Generación de la especificación arquitectónica. Se debe crear una representación estructurada de la arquitectura en formato estandarizado para garantizar la precisión y automatización del análisis arquitectónico. Sin esta

estandarización, los cálculos serían más propensos a errores y más difíciles de implementar. La especificación tiene que incluir: nombres y dominios de los microservicios, funcionalidades asignadas a cada microservicio, dependencias internas (entre microservicios), dependencias externas (con sistemas externos).

Paso 9: Visualización de la arquitectura. Es necesario generar una representación visual de la arquitectura, esta representación permitirá comprender las relaciones y dependencias de forma clara e identificar problemas como alto acoplamiento o dependencias cíclicas. El diagrama debe otorgar la visualización de los microservicios agrupados por dominios funcionales, historias de usuario contenidas en cada microservicio, dependencia internas y externas.

Paso 10: Reportes de los procesos generados. Es necesario documentar los resultados de los distintos procesos, por tanto, se requiere del modelo los siguientes reportes: procesamiento inicial de historias de usuario, agrupación en dominios funcionales, definición de microservicios y sus dependencias, generación de la especificación estándar de la arquitectura, diagrama visual,

- **Evaluación de la Arquitectura**

A partir de la representación estructurada de la arquitectura, se evalúa la calidad de la misma desde las perspectivas de dependencia y complejidad, utilizando un conjunto de métricas específicas. Con base en estas métricas, se genera un diagnóstico cuantitativo de la arquitectura y recomendaciones para mejorar su diseño. Esto incluye sugerencias para reducir el acoplamiento, mejorar la cohesión y simplificar las interacciones internas.

- **Pasos para la evaluación de la arquiectura:**

Paso 1: Entrada de la Arquitectura. Se recibe como entrada la especificación estructurada que describe la arquitectura generada, es necesario validar que la especificación contiene todos los elementos requeridos (microservicios agrupados y sus dependencias), el objetivo es tener todos los datos para el análisis mediante métricas.

Paso 2: Cálculo de Métricas. Se calculan las métricas para evaluar la calidad de la arquitectura desde las perspectivas de dependencia y complejidad, los resultados describen cuantitativamente el estado de la arquitectura. Las métricas son los siguientes:

1. Dependencias Externas (ND): Se cuenta cuántos microservicios dependen de sistemas externos. Un número alto puede indicar poca autonomía.

2. Dependencias Internas (NI): Se calcula la cantidad de relaciones internas entre microservicios. Muchas relaciones pueden indicar acoplamiento innecesario.

3. Profundidad de Dependencias (PD): Se mide la longitud de la cadena de dependencias internas más larga. Una mayor profundidad puede incrementar la complejidad y el riesgo de fallos en cascada.

4. Acoplamiento Interno (AC_est): Se analiza la proporción de conexiones entre microservicios en relación con el número máximo posible de conexiones y se ajusta esta proporción a una escala para obtener un valor numérico de acoplamiento.

5. Falta de cohesión (LCOH_est): Se evalúa la similitud semántica entre las funcionalidades dentro de un microservicio. Una cohesión baja puede indicar que las responsabilidades de un microservicio están mal definidas.

6. Complejidad Ciclomática (CC_est): Se analiza la cantidad de nodos (microservicios) y aristas (dependencias) en el grafo de dependencias. Se mide el número de ciclos fuertes en el grafo.

7. Acoplamiento por Recursos Compartidos (CA_est): Se calcula cuántos recursos son compartidos por múltiples microservicios, lo cual puede aumentar el acoplamiento.

8. Complejidad Estructural (CE_est): Se evalúa la cantidad total de microservicios y dependencias. Se combinan estos valores para obtener una métrica de complejidad estructural.

9. Puntos de historia estimados (PH_est): Se suma el número total de funcionalidades asignadas a todos los microservicios. Valores altos pueden indicar una distribución desigual de responsabilidades.

10. Índice de Arquitectura estimado (AI_est): Se combina la dependencia (D_est) y la complejidad (C_est) para calcular un índice general de calidad arquitectónica.

Paso 3: Salida de evaluación y recomendaciones. Para esto, se debe crear un diagnóstico que incluye métricas y recomendaciones personalizadas que represente una valoración específica para mejorar la arquitectura. Las recomendaciones deben incluir: la reducción de dependencias externas para mejorar la autonomía, simplificación de las dependencias internas para reducir el acoplamiento, incremento de la cohesión mediante una mejor distribución

funcional, disminución de la complejidad estructural reorganizando dependencias y un equilibrio de la carga funcional entre microservicios.

Este diagnóstico debe representar un análisis completo que facilite el refinamiento continuo de la arquitectura, el objetivo es permitir a los diseñadores arquitectónicos tomar decisiones informadas sobre cómo mejorar su sistema.

3.2.3.2 *Representación matemática del modelo*

En esta sección se sustenta la tercera pregunta de investigación:

PI.3 ¿Cómo diseñar un modelo matemático de evaluación de la calidad para determinar dependencia y complejidad en la arquitectura de aplicaciones basadas en microservicios?

El diseño de arquitecturas basadas en microservicios presenta desafíos significativos en términos de dependencia entre servicios y complejidad del sistema. Evaluar estos factores durante la fase de diseño es crucial para asegurar que el sistema sea escalable, mantenible y resiliente. Un mal manejo de las dependencias y la complejidad puede conducir a sistemas frágiles, difíciles de mantener y propensos a fallos.

El modelo se fundamenta en el nivel de importancia que tienen tanto la dependencia como la complejidad en la calidad de los microservicios, [101] manifiesta que, las arquitecturas de microservicios ofrecen escalabilidad y flexibilidad, pero si no se diseñan correctamente, pueden generar problemas como alto acoplamiento, anti-patrones y complejidad excesiva, por ello, la identificación y corrección de anti-patrones es crucial para mantener la calidad y mantenibilidad del sistema, los métodos existentes no son completamente automatizados ni ofrecen suficientes métricas o visualizaciones basadas en teoría

de grafos, siendo necesario un enfoque más eficiente para analizar las dependencias y detectar anti-patrones en sistemas complejos.

Los sistemas de microservicios son esenciales para manejar aplicaciones modernas debido a su flexibilidad y escalabilidad, sin embargo, las dependencias complejas y la coexistencia de múltiples instancias de microservicios dificultan la optimización del tiempo de respuesta y la calidad del servicio [98].

Según [102] las dependencias de microservicios se refiere a la dependencia de un microservicio de otro para su correcto funcionamiento. Las dependencias pueden ser explícitas (como llamadas directas entre servicios) o implícitas (como modelos de datos compartidos, sistemas de mensajes o cumplimiento de políticas globales). Las dependencias son un aspecto fundamental de los microservicios pero introducen complejidad, especialmente en términos de capacidad de mantenimiento, escalabilidad e integridad arquitectónica. Comprender y gestionar estas dependencias de manera eficaz es fundamental para garantizar la resiliencia, la modificabilidad y la escalabilidad de los sistemas basados en microservicios.

La dependencia para [5] se refiere a las relaciones e interacciones entre los microservicios. Estas dependencias pueden clasificarse en dos tipos principales: 1) explícita cuando la conexión es directa entre dos microservicios, se da cuando un microservicio depende directamente de otro para intercambiar datos o ejecutar una función específica y, 2) implícita cuando la conexión es indirecta y se establece a través de uno o más microservicios intermediarios, aunque no existe una interacción directa, un microservicio puede depender de otro debido al flujo de información que pasa por intermediarios. Otro de los conceptos importantes que introduce el trabajo de J. McZara (2020) es el bajo acoplamiento como un objetivo deseable que se refleja en la necesidad de diseñar

microservicios con interfaces claras que limiten las dependencias rígidas entre ellos para evitar que fallos en un servicio afecten significativamente a otros; se menciona explícitamente que la gestión de las interdependencias entre los microservicios es un desafío clave, ya que estas conexiones aumentan el riesgo de co-evolución, donde los cambios en un microservicio pueden requerir modificaciones en otros.

Por otro lado, sobre la complejidad [5] manifiesta que, de una aplicación basada en microservicios la complejidad aumenta con el número de microservicios y los enlaces de interdependencia entre ellos, ya sean explícitos o implícitos. Este aumento complica el análisis, recuperación y pruebas durante el ciclo de vida de la aplicación

La complejidad surge de la autonomía y heterogeneidad de los microservicios, donde cada servicio opera independientemente pero interactúa a través de una interfaz de programación de aplicaciones (API), recursos compartidos o comunicación indirecta. La alta complejidad dificulta definir claramente los límites del servicio y diseñar sistemas escalables y mantenibles. El uso del diseño impulsado por el dominio ayuda a dividir las aplicaciones en servicios bien definidos, lo que reduce la ambigüedad y la interdependencia, asimismo, actividades como la evaluación arquitectónica se centran en la escalabilidad, el rendimiento y la disponibilidad para gestionar la complejidad [103], por tanto, poner énfasis en el diseño de alto nivel y el desarrollo iterativo va a permitir una mejor adaptación a los requisitos cambiantes y la identificación de puntos críticos. En esencia, la complejidad de los microservicios proviene de su naturaleza distribuida e independiente, que, si bien ofrece escalabilidad y flexibilidad, exige estrategias rigurosas de diseño, monitoreo y prueba para gestionarla de manera efectiva.

Al acoplamiento [104] lo refiere como el grado de interdependencia entre los microservicios, aborda este tema, a través de la reducción de dependencias entre microservicios con el método de clustering, agrupando clases que tienen dependencias fuertes entre sí, de esta manera, minimiza las dependencias entre diferentes microservicios. Adicionalmente, promueve la separación de responsabilidades, al identificar microservicios con clases que comparten un propósito común, buscando reducir el número de interacciones necesarias entre microservicios, lo que contribuye a un bajo acoplamiento.

Para [105] el acoplamiento de microservicios es un indicador crítico de calidad interna en la arquitectura de software. Un bajo acoplamiento favorece la independencia de los microservicios, lo que mejora su mantenibilidad, escalabilidad y capacidad de evolución. El trabajo introduce el Microservice Coupling Index (MCI), que incluye métricas específicas para medir la influencia de los microservicios entre sí: MCI (Microservice Coupling Index), cuantifica el acoplamiento entre dos microservicios; aMCI (Afferent Microservice Coupling Index), mide cuánto influye un microservicio en otros dentro del sistema; eMCI (Efferent Microservice Coupling Index), mide cuánto es influenciado un microservicio por otros.

La cohesión es otro elemento importante de análisis, [105] manifiesta que, la cohesión se refiere a qué tan bien las partes internas de un microservicio (sus clases, métodos e interfaces) trabajan juntas para cumplir con una responsabilidad o funcionalidad específica. Un microservicio con alta cohesión tiene una funcionalidad clara y no depende innecesariamente de otros microservicios, es decir, cuantificar este parámentro en términos de falta de cohesión, permitira identificar problemas arquitectónicos en una aplicación de microservicios.

Sobre las técnicas utilizadas para la identificación de microservicios, en el trabajo [96] se destacan: 1) el análisis estático (inspección del código fuente, métodos, clases, dependencias y estructuras estáticas del sistema), siendo esta una de las técnicas más utilizadas; 2) análisis dinámico, basado en la ejecución del sistema (trazas de uso, logs y pruebas de usuario), su adopción ha aumentado debido a la capacidad para capturar comportamientos reales en producción; 3) domain-driven analysis (utiliza principios de modelado de dominio, contextos delimitados y procesos de negocio) y; 4) machine learning/optimization (uso de algoritmos de aprendizaje automático y optimización para descomponer sistemas en microservicios), ha mostrado crecimiento reciente, destacándose como un enfoque emergente.

Una referencia del tipo de métricas que se usan para la evaluación de las denpendencias de microservicios, se puede encontar en el trabajo [45]. Este trabajo aplica métricas específicas para evaluar la granularidad de los microservicios en términos de acoplamiento, cohesión y complejidad: métricas de acoplamiento (AIS "Absolute Importance of the Service", ADS "Absolute Dependence of the Service", SIY "Microservices Interdependence", CpT "Coupling Total"); métricas de cohesión (LC "Lack of Cohesion", CohT "Degree of Cohesion"); métricas de complejidad (WSIC "Weighted Service Interface Count"), story points, development time; métrica compuesta (GM "Granularity Metric"). Estas métricas permiten analizar y optimizar la descomposición en microservicios desde el diseño, evalúan las dependencias entre los microservicios y cómo estas afectan la calidad del diseño.

Otro trabajo que especifica métricas para evaluar microservicios desde la perspectiva de la arquitectura es [83]. Este estudio propone un nuevo conjunto de métricas para aplicaciones que utilizan arquitecturas de microservicios. Entre las métricas introducidas se encuentran la Métrica de Granularidad del Servicio

(SGM), la Métrica de Falta de Cohesión (LCOM) y el Número de Operaciones (NOO). Estas herramientas se utilizan para medir la granularidad, cohesión y complejidad de microservicios individuales mediante el análisis de sus interfaces de programación de aplicaciones (API). Gracias a estas métricas, es posible evaluar la calidad general del diseño de aplicaciones basadas en microservicios.

El artículo [106] propone una métrica de complejidad para evaluar el costo de migrar aplicaciones monolíticas a una arquitectura basada en microservicios. Este enfoque busca facilitar la toma de decisiones al analizar la dificultad de rediseñar funcionalidades distribuidas y ofrece herramientas para medir la cohesión, el acoplamiento y la complejidad de las descomposiciones de monolitos en microservicios. La complejidad de una funcionalidad mide la dificultad de implementar esa funcionalidad en un sistema distribuido. Se basa en el número de entidades de dominio que la funcionalidad lee y escribe y en las dependencias entre funcionalidades que comparten entidades de dominio.

El trabajo [107] propone un modelo semiautomático para determinar la granularidad de los microservicios durante la fase de diseño utilizando historias de usuario como entrada. Para lograrlo, se desarrolla un algoritmo genético que optimiza las descomposiciones basándose en métricas específicas de acoplamiento, cohesión, granularidad, complejidad y similitud semántica. El artículo propone métricas clave para evaluar la granularidad de microservicios, centradas en acoplamiento, cohesión, granularidad y complejidad. Entre ellas destacan: AIS (dependencias entrantes), ADS (dependencias salientes), SIY (interdependencias bidireccionales), LC (falta de cohesión), CohT (grado de cohesión), WsIC (número de historias por microservicio) y CxT (complejidad cognitiva). Una métrica compuesta, GM, combina estas dimensiones para evaluar la calidad global de la descomposición. Las pruebas realizadas en casos de estudio como Cargo Tracking, JPet Store y Foristom Conferences demostraron

que el modelo genera microservicios con menor acoplamiento, mayor cohesión y granularidad óptima, superando métodos como el Diseño Dirigido por el Dominio (DDD) y Service Cutter, gracias al uso de un algoritmo genético que optimiza agrupaciones basadas en dependencias y similitud semántica entre historias de usuario.

Con estos antecedentes, la especificación formal del modelo matemático para evaluar la calidad arquitectónica de aplicaciones basadas en microservicios, se enfoca en las métricas de dependencia y complejidad aplicables en tiempo de diseño. El modelo utiliza diagramas arquitectónicos para estimar estas métricas, su objetivo principal es proporcionar una evaluación cuantitativa de la arquitectura de microservicios para identificar áreas problemáticas y guiar decisiones de refactorización o rediseño. Estas métricas incluyen:

- ND (Número de Dependencias Externas): Cuenta los microservicios que tienen dependencias externas.

- NI (Número de Interacciones Internas): Suma el número total de dependencias internas entre microservicios.

- PD (Profundidad de Dependencias): Determina la longitud del camino más largo de dependencias internas.

- AC_est (Acoplamiento Interno Estimado): Calcula el acoplamiento interno basado en el número de dependencias internas en relación con el número máximo posible de conexiones.

- LCOH_est (Falta de Cohesión Estimada): Mide la cohesión dentro de un microservicio utilizando similitud coseno entre las descripciones textuales de sus funcionalidades.

• CC_est (Complejidad Ciclomática Estimada): Calcula la complejidad ciclomática de la arquitectura basada en sus dependencias internas.

• CA_est (Acoplamiento por Recursos Compartidos Estimado): Evalúa el grado de acoplamiento debido a recursos compartidos entre microservicios.

• CE_est (Complejidad Estructural Estimada): Combina el número de microservicios y sus interacciones internas para medir la complejidad estructural.

• PH_est (Puntos de Historia Estimados): Suma las funcionalidades de todos los microservicios para estimar el esfuerzo de desarrollo.

• D_est (Dependencia Estimada): Agrega ND, NI, PD y LCOH_est para obtener una medida global de dependencia.

• C_est (Complejidad Estimada): Combina CC_est, CA_est, CE_est y PH_est para obtener una medida global de complejidad.

• AI_est (Índice de Arquitectura Estimado): Suma D_est y C_est para obtener un índice global de la arquitectura.

Con base en la contextualización anterior, se propone el siguiente modelo matemático:

Representación del modelo matemático

Sea $M = \{M_{1,} M_{1,} \dots, M_N\}$ el conjunto de microservicios que conforman la arquitectura, donde $N = |\mathrm{M}|$ es el número total de microservicios.

Para cada microservicio $M_{i=}M$:

- Funcionalidades: $F_i = \{f_{i1,}\, f_{i2,}\, \dots, f_{in}\}$, donde $n_{i=}|F_i|$ es el número de fincionalidades de M_i
- Dependencias Internas: $D_{int,i} \subseteq M \backslash \{M_{i,}\}$, el conjunto de microservicios de los que M_i depende internamente
- Dependencias Externas: $D_{ext,i}$, el conjunto de sistemas externos de los que M_i depende
- Recursos Compartidos: $R_i \subseteq R$, donde R es el conjunto total de recursos compartidos en el sistema

Grafo dirigido $\boldsymbol{G}$ $(\boldsymbol{M}, \boldsymbol{E})$, donde:

- Nodos: Los microservicios M
- Aristas: $E = \{(M_{i,}\, M_j) | M_j \in D_{int,i}\}$, representando las dependencias internas

En los siguientes apartados se definen las métricas:

Dependencias Extenas (ND)

La fórmula evalúa cuántos microservicios están conectados a sistemas externos y, por lo tanto, mide un aspecto de la integración externa en la arquitectura del sistema; *0* es el valor mínimo, ocurre cuando ningún microservicio tiene dependencias hacia sistemas externos; *N* es el valor máximo, ocurre cuando todos los microservicios dependen de al menos un sistema externo, siendo *N* el número total de microservicios en el sistema.

Definición formal: $ND = |\{M_i \in D_{ext,i} \neq \emptyset\}$ (1)

Dominio: $0 \leq ND \leq N$ (2)

Dependencias Internas (NI)

Representa el grado de acoplamiento interno entre microservicios. Un bajo NI indica que los microservicios tienen pocas dependencias internas, lo que puede significar que están diseñados para ser más independientes (bajo acoplamiento interno). Esto suele ser deseable en arquitecturas de microservicios bien diseñadas. Un alto NI sugiere un sistema donde los microservicios tienen muchas dependencias entre ellos, lo que puede dificultar la escalabilidad y aumentar la complejidad del mantenimiento, puede señalar problemas de diseño, como una descomposición ineficaz; 0 representa el caso en el que no hay dependencias internas entre microservicios (todos son completamente independientes); N(N−1) es el valor máximo, que ocurre si cada microservicio está conectado con todos los demás (N−1 conexiones por microservicio).

Definición formal: $\mathrm{NI} = \sum_{i=1}^{N} d_i$ (3)

Donde $d_i = |D_{int,i}|$ es el número de dependencias internas del microservicio $M_{i,}$

Dominio: $0 \leq NI \leq N\,(N-1)$ (4)

Produndidad de dependencias (PD)

Representa la cadena más larga de dependencias internas, proporciona una medida importante de la complejidad estructural de las dependencias internas en un sistema de microservicios. Un valor alto podría indicar problemas de diseño que dificultan la escalabilidad, el mantenimiento y el rendimiento del sistema; 0 indica que no hay dependencias internas en el sistema (los microservicios son completamente independientes); N−1 es el valor máximo de PD que ocurre cuando todos los microservicios están conectados en una única cadena lineal (es decir, un microservicio depende del siguiente en una secuencia sin ramificaciones).

Definición formal: $PD = \max_{p \in P}(|p|)$ (5)

Donde:

P es el conjunto de todos los caminos dirigidos en G

$|p|$ es la longitud del camino p (numero de aristas).

Dominios: $0 \leq PD \leq N - 1$ (6)

Acoplamiento interno estimado AC_{est}

Evalua el acoplamiento interno de un sistema de microservicios, la métrica mide el grado de densidad de conexiones internas entre microservicios, asignando un valor escalado entre 3 y 10. Este enfoque permite interpretar fácilmente el nivel de acoplamiento interno del sistema; bajo AC_{est} significa que el sistema tiene pocas dependencias internas y que los microservicios están diseñados con alta independencia, facilitando su escalabilidad y mantenibilidad; alto AC_{est} significa que el sistema tiene muchas dependencias internas, esto puede indicar un diseño excesivamente acoplado, lo que aumenta la complejidad y reduce la resiliencia del sistema.

Definición formal: $AC_{est} = 3 + (\frac{NI}{N(N-1)}\ x\ 7)$ (7)

Dominio: $3 \leq AC_{est} \leq 10$ (8)

Falta de Cohesión Estimada $LCOH_{est}$

La métrica Falta de Cohesión Estimada ($LCOH_{est}$) mide el grado de cohesión interna de un microservicio basado en la similitud entre las funcionalidades que contiene. Cuanto menor sea la cohesión, más dispersas son las responsabilidades dentro del microservicio, lo que sugiere un mal diseño. Esta métrica se calcula a

nivel individual para cada microservicio y como un promedio global para todo el sistema.

Definición formal:

Para cada microservicio M_i con $n_i \geq 2$:

1. Similitud entre funcionalidades: Evalúa la similitud entre dos funcionalidades f_{ik} y f_{il} dentro de un mismo microservicio M_i utilizando el cálculo del coseno entre sus vectores TF-IDF (Term Frequency-Inverse Document Frequency). Valores cercanos a 1 indican alta similitud semántica entre las funcionalidades, mientras que valores cercanos a 0 indican baja similitud.

$$Sim\left(f_{ik,}f_{il,}\right) = \cos(\theta_{kl,}) \quad (9)$$

Donde θ_{kl} es el ángulo entre los vectores TF-IDF de las funcionalidades $f_{ik}\ Y\ f_{il}$

Se emplea TF-IDF y cosine similarity para analizar la cohesión entre funcionalidades asignadas a un mismo microservicio, asegurando que las historias agrupadas compartan un propósito común.

2. Similitud promedio: Calcula el promedio de las similitudes entre todas las combinaciones de pares de funcionalidades en el microservicio M_i que tiene al menos $n_i \geq 2$funcionalidades:

$$\overline{Sim_i} = \frac{2}{n_i\,(n_i - 1)} \sum_{k<l}^{Sim} (f_{il}, f_{ik,}) \quad (10)$$

n_i : Número de funcionalidades en el microservicio Mi

Esto considera todas las posibles combinaciones de pares de funcionalidades en Mi

3. Cálculo de $LCOHest$ para M_i: Calcula la falta de cohesión de Mi usando la similitud promedio $\overline{Sim_i}$. Valores de $\overline{Sim_i}$ más altos (alta similitud) reducen el valor de LCOHest,i indicando mayor cohesión; el valor mínimo de LCOHest,i es 1.0 (máxima cohesión), y el máximo es 2.0 (mínima cohesión).

$$LCOH_{est,i} = \max(1.0, 2.0 - 1.5 \times \overline{Sim_i} \quad (11)$$

4. LCOH global: Calcula el promedio de la falta de cohesión de todos los microservicios con al menos 2 funcionalidades (M'). M′ conjunto de microservicios con $n_i \geq 2$ funcionalidades; 1.0 indica máxima cohesión interna (alta similitud promedio entre las funcionalidades del microservicio), 2.0 indica mínima cohesión interna (baja similitud promedio entre funcionalidades).

$$LCOH_{est} = \frac{1}{|M'|}\sum_{M_i \in M'} LCOH_{est,i} \quad (12)$$

Donde $M' = \{M_i \in M | \ n_i \geq 2\}$

Dominio individual: $1.0 \leq LCOH_{est,i} \leq 2.0$ (13)

Valores Más Altos: Indican menor cohesión interna.

Complejidad Ciclomática Estimada (CC_{est})

La complejidad ciclomática estimada mide la complejidad estructural de una arquitectura en términos de ciclos y caminos independientes en el grafo de dependencias entre microservicios. Es una métrica comúnmente utilizada en

ingeniería de software para analizar la complejidad de sistemas y componentes, adaptada aquí para sistemas de microservicios. Representa la complejidad estructural en términos de ciclos y caminos independientes; el valor mínimo de $CCest$ es 2, que ocurre en un sistema con una configuración básica (al menos dos microservicios conectados de forma lineal); en sistemas más complejos, $CCest$ aumentará en función de las conexiones adicionales (E) y los ciclos presentes (P).

Definición formal: $CC_{est} = E - N + P$ (13)

Donde:

$E = |E| = NI$ es el número total de aristas en G

$N = |M|$ es el número de nodos

P es el número de componentes fuertemente conectados en G

Dominio: $CC_{est} \geq 2$

Acoplamiento por recursos compartidos estimado (CA_{est})

La métrica Acoplamiento por Recursos Compartidos Estimado (CA_{est}) mide el grado de acoplamiento entre microservicios debido al uso de recursos compartidos (por ejemplo, bases de datos, archivos o cualquier entidad que sea accedida por más de un microservicio). Esta métrica refleja cómo las dependencias derivadas de compartir recursos pueden influir en la calidad del diseño y la modularidad de la arquitectura. Bajo CA_{est} indica que el sistema tiene pocos recursos compartidos en relación con el número de microservicios. Esto es deseable porque reduce el acoplamiento y mejora la independencia de los microservicios, alto CA_{est} refleja que hay una gran cantidad de recursos compartidos, lo que incrementa el acoplamiento entre microservicios y puede generar problemas como: mayores dependencias entre servicios, riesgo de

conflictos de acceso concurrente, mayor complejidad en la sincronización o coordinación.

Definición formal:

$$\text{Sea } R_{shared} = \{r \in R \mid |\{M_i | r \in R_i\}| \geq 2\} \quad (14)$$

$$CA_{est} = \frac{|R_{shared}|}{N} \quad (15)$$

R_{shared}. Es el subconjunto de recursos compartidos que son accedidos por al menos dos microservicios.

R: Conjunto total de recursos del sistema.

R_i: Recursos accedidos por el microservicio M_i

r: Un recurso específico.

$|\{M_i | r \in R_i\}|$: Número de microservicios que acceden al recurso r.

Dominios: $0 \leq CA_{est} \leq \frac{|R_{shared}|}{N}$ (16)

Complejidad Estructural Estimada (CE_{est})

La Complejidad Estructural Estimada ($CEest$) mide la complejidad de una arquitectura de microservicios en función del número de microservicios y sus interacciones internas. Esta métrica combina ambos factores utilizando coeficientes de ponderación, lo que permite ajustar la importancia relativa de cada componente según las necesidades del sistema. Alto $CEest$ indica que el sistema tiene un diseño simple con un número razonable de microservicios y pocas interacciones internas, suele reflejar una arquitectura bien diseñada, fácil de mantener y escalar; bajo $CEest$ sugiere que el sistema tiene una complejidad significativa debido a un gran número de microservicios, muchas interacciones

internas, o ambos, esto puede reflejar problemas de diseño, como microservicios excesivamente pequeños (granularidad fina) o dependencias internas excesivas.

Definición formal:

$$(CE_{est}) = \alpha\, x\, C + \beta\, x\, I \quad (17)$$

Donde:

$C = N$ es el número de microservicios

$I = NI$ es el número de interacciones internas

$\alpha, \beta \in \mathbb{R}^+$ son coeficientes de ponderación

Dominio: $CE_{est} \geq 0$ (18)

Personalización: Ajustable según la importancia relativa de C e I.

Puntos de historia estimados PH_{est}

La métrica Puntos de Historia Estimados ($PHest$) mide el volumen total de funcionalidades presentes en un sistema de microservicios. Representa una suma acumulada de todas las funcionalidades asignadas a los microservicios, reflejando la carga funcional global del sistema. Bajo $PHest$ indica que el sistema tiene un volumen reducido de funcionalidades, lo que sugiere que es menos complejo o que está en una etapa temprana de desarrollo; alto $PHest$ refleja un sistema con muchas funcionalidades, lo que puede implicar mayor complejidad, tanto en términos de implementación como de mantenimiento.

Definición formal:

$$PH_{est} = \sum_{i=1}^{N} n_i \quad (19)$$

Dominio: $PH_{est} \geq 0$ (20)

Dependencia estimada (D_{est})

La métrica Dependencia Estimada ($Dest$) mide la complejidad general de las dependencias dentro de un sistema de microservicios. Combina múltiples métricas de dependencia para proporcionar una visión integral del acoplamiento y la cohesión entre los microservicios. Es una métrica compuesta que abarca tanto la cantidad como la calidad de las dependencias internas y externas. Bajo $Dest$ indica que el sistema tiene pocas dependencias externas e internas, cadenas de dependencias cortas y microservicios altamente cohesivos, es deseable porque refleja una arquitectura bien diseñada, con bajo acoplamiento y alta cohesión; alto $Dest$ sugiere un sistema con muchas dependencias internas y externas, cadenas largas de dependencias y falta de cohesión interna en los microservicios. Puede reflejar un diseño problemático que requiere refactorización para reducir la complejidad y mejorar la modularidad.

Definicón formal:

$$D_{est} = ND + NI + PD + AC_{est} + LCOH_{est} \quad (21)$$

Dominio: $D_{est} \geq 0$ (22)

Complejidad estimada C_{est}

La métrica Complejidad Estimada ($Cest$) mide la complejidad total de un sistema de microservicios combinando diversas métricas que capturan diferentes aspectos de la estructura y funcionalidad del sistema. Este enfoque permite evaluar la complejidad de manera integral, teniendo en cuenta dependencias, recursos compartidos, estructura del diseño y el volumen funcional. Baja $Cest$ indica que el sistema tiene una baja complejidad estructural, funcional y de

dependencia, es deseable, ya que implica una arquitectura más sencilla de mantener, escalar y comprender; alta $Cest$ sugiere que el sistema tiene alta complejidad debido a múltiples factores, como muchas dependencias cíclicas, recursos compartidos, interacciones internas complejas o un gran volumen de funcionalidades, puede reflejar problemas de diseño que dificultan la escalabilidad, mantenibilidad y desarrollo.

Definición formal:

$$C_{est} = CC_{est} + CA_{est} + CE_{est} + PH_{est} \quad (23)$$

Dominio: $C_{est} \geq 0$ (24)

Índice de Arquitectura Estimado $AI \geq 0$

El Índice de Arquitectura Estimado (AI_{est}) mide la carga total de dependencia y complejidad en un sistema de microservicios. Es una métrica compuesta que combina las métricas de dependencia ($Dest$) y complejidad (C_{est}) para proporcionar una visión integral del diseño arquitectónico del sistema. Bajo A$Iest$ indica que el sistema tiene una arquitectura bien diseñada con baja complejidad y pocas dependencias, esto es deseable porque refleja una arquitectura más fácil de mantener, escalar y desarrollar; Alto A$Iest$ sugiere que el sistema enfrenta alta complejidad y/o un alto nivel de dependencias, esto puede reflejar problemas de diseño que dificultan el mantenimiento, la escalabilidad y la resiliencia del sistema.

Definición formal:

$$AI_{est} = D_{est} + C_{est} \quad (25)$$

Dominios: $AI_{est} \geq 0$ (26)

A medida que AI_{est} aumenta significa menor calidad, representa una métrica única para evaluar la calidad arquitectónica. Este modelo proporciona una base matemática sólida para evaluar arquitecturas de microservicios, identificar áreas con alta dependencia o complejidad, comparar diferentes diseños o refactorizaciones.

Esta especificación formal presentada una notación matemática precisa para definir las métricas y sus propiedades en el contexto de arquitecturas de microservicios. Al establecer definiciones rigurosas y describir las relaciones y límites de las métricas, se facilita un análisis matemático profundo y una comprensión clara de cómo cada aspecto de la arquitectura contribuye a la dependencia y complejidad del sistema.

3.2.3.3 *Implementación del modelo*

En esta sección se describe el proceso de implementación del modelo y se sustenta la cuarta pregunta de investigación:

PI.4 ¿Qué técnicas de Machine Learning para Procesamiento de Lenguaje Natural se pueden aplicar para definir un modelo inteligente que especifique dependencia y complejidad en el diseño de aplicaciones basadas en microservicios?

En el ámbito de las arquitecturas de microservicios, garantizar la calidad del diseño es esencial para lograr sistemas escalables, mantenibles y robustos. Sin embargo, las arquitecturas complejas presentan desafíos relacionados con la complejidad, dependencia, acoplamiento y cohesión de los microservicios, que pueden comprometer la eficiencia operativa y la evolución del sistema.

En este contexto, se desarrolló el modelo inteligente utilizando el marco de la ciencia del diseño para el desarrollo de modelos en Sistemas de Información [49]. A través de un enfoque sistemático e iterativo, la herramienta fue diseñada para procesar las entradas en forma de historias de usuario, agruparlas según los dominios funcionales y posteriormente generar el diagrama de la aplicación y las respectivas definiciones arquitectónicas en formato JSON. Una vez se tienen las definiciones arquitectónicas, calcula las métricas clave y proporcionar recomendaciones prácticas para optimizar la calidad de la arquitectura.

- **Identificar el problema**

La determinación del problema se deriva de la necesidad de los equipos de desarrollo, de contar con herramientas automáticas que permitan identificar rápidamente los cuellos de botella arquitectónicos, como servicios con alta dependencia o baja cohesión. Sin una herramienta automatizada, los equipos deben realizar análisis manuales que son propensos a errores y costosos en términos de tiempo.

Desde la perspectiva del desarrollo ágil, los equipos necesitan evaluar la calidad arquitectónica de forma continua durante el ciclo de desarrollo. Sin una herramienta automática, las evaluaciones de dependencia y complejidad pueden retrasar los sprints y dado que requieren análisis manuales extensos, las métricas críticas, como cohesión y acoplamiento, pueden quedar subestimadas, lo que puede comprometer decisiones de diseño.

Así mismo, el diseño de microservicios busca fomentar la escalabilidad horizontal, pero si las dependencias no se gestionan correctamente el costo del mantenimiento aumenta exponencialmente y la escalabilidad se ve comprometida por arquitecturas con alto acoplamiento interno y dependencias profundas, con una herramienta automática se puede ayudar a los

desarrolladores a medir y controlar estos factores, asegurando que las arquitecturas escalen de manera eficiente.

Las decisiones arquitectónicas tomadas durante el diseño inicial tienen un impacto significativo en la evolución del sistema. Sin herramientas que calculen de forma automática aspecto de calidad como la dependencia y complejidad, los equipos enfrentan dificultades para predecir el impacto de nuevos servicios o funcionalidades, se incrementa el riesgo de incurrir en deuda técnica, que puede ser costosa y difícil de abordar en etapas avanzadas del desarrollo.

Una herramienta automática puede permitir analizar la calidad arquitectónica en tiempo real, previniendo problemas antes de que se conviertan en riesgos críticos, además, proporciona una representación clara y unificada de la arquitectura, facilitando la colaboración y la alineación entre equipos.

Adicionalmente, se analizaron otras investigaciones sobre arquitecturas de microservicios y métricas relevantes, destacando la falta de herramientas específicas que evalúen las dependencias internas y externas, así como, la cohesión y el acoplamiento. De igual manera, se encontraron problemas como las limitaciones que tienen las herramientas existentes ya que se enfocan más en el código que en las arquitecturas.

- **Proponer el modelo**

De acuerdo al análisis descrito en la fase anterior, se propuso y se desarrolló el siguiente modelo inteligente con los componentes funcionales que a continuación se describen:

  - Ingreso del product backlog (conjunto de historias de usuario de un caso)
  - Preprocesamiento de historias de usuario

- Agrupamiento de historias de usuario en dominios funcionales
- Inferencia de dependencias internas y externas
- Generación del diagrama arquitectónico que visualiza los microservicios y sus dependencias
- Calculo de métricas para especificar dependencia y complejidad en el diseño de aplicaciones basadas en microservicios.

- **Diseño del modelo**

El modelo se diseñó en dos iteraciones, cada iteración permitió generar un componente funcional: 1) componente de arquitectura de microservicios y, 2) componente de aplicación del modelo matemático para evaluar el diseño arquitectónico de microservicios (Figura 15).

El componente 1 se encarga de generar el diagrama arquitectónico de las aplicaciones basadas en microservicios, lo hace a partir del procesamiento de las historias de usuario de un caso o proceso, organiza las historias en microservicios utilizando técnicas avanzadas de Procesamiento de Lenguaje Natural (NLP) y aprendizaje automático, finaliza el proceso creando un archivo JSON que contiene las especificaciones arquitectónicas de la aplicación de microservicios (nombre, funcionalidades, dependencias internas y externas, y recursos compartidos).

El componente 2 analiza la especificación arquitectónica de la aplicación de microservicios a través del archivo JSON proporcionado por el usuario, posteriormente, calcula las métricas para determinar dependencia y complejidad en el diseño de los microservicios, finaliza entregando los resultados de cada una de las métricas del modelo matemático. La figura 18 presena una visión conceptual de la arquitectura de MOAM.

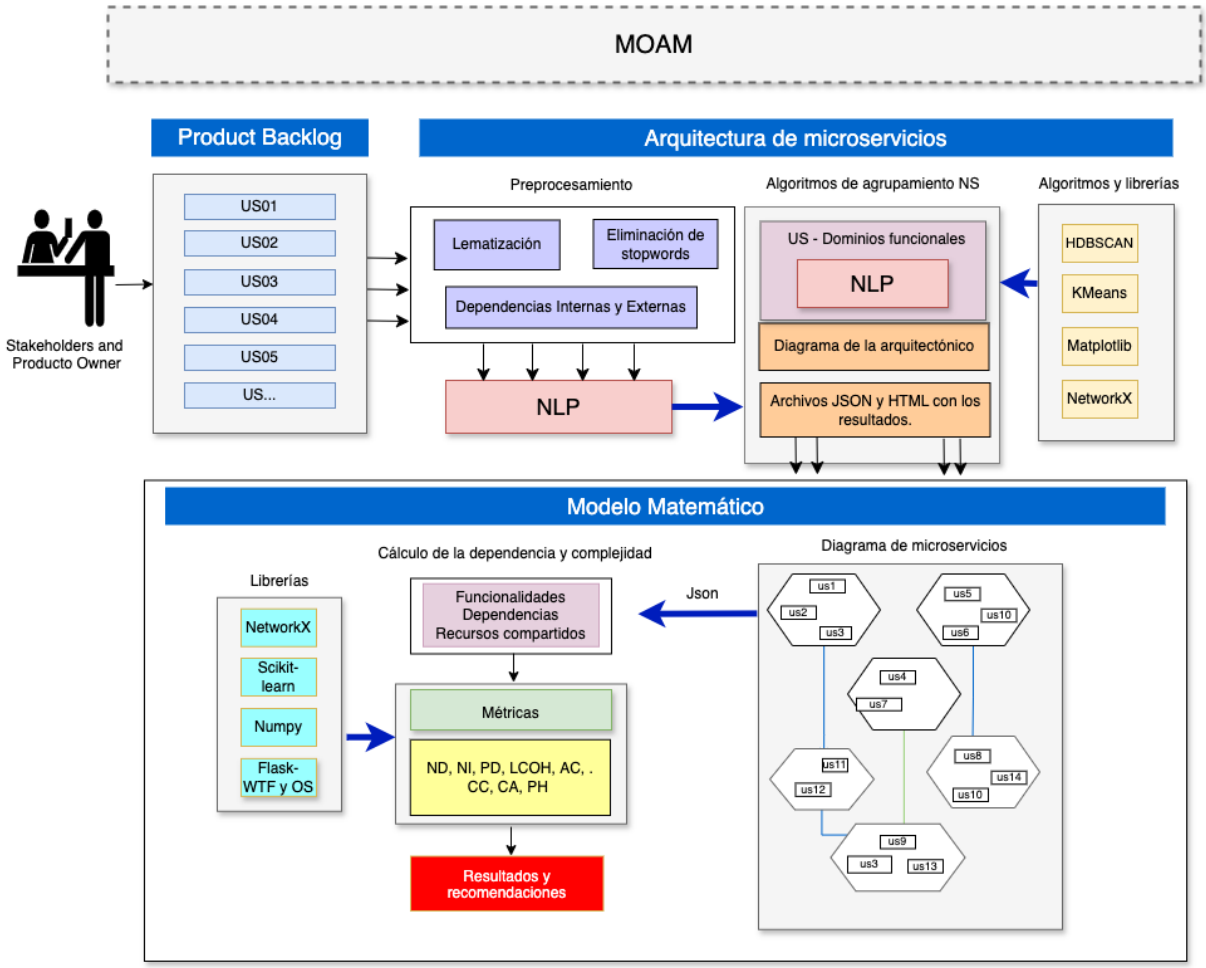


Figura 18. Microservices Optimization Architectures Model "MOAM"

- **Componente: Arquitectura de microservicios**

Este componente es una solución diseñada para procesar historias de usuario, agruparlas en funcionalidades relacionadas y organizar estas funcionalidades en una arquitectura basada en microservicios. Se utiliza un enfoque centrado en técnicas avanzadas de Procesamiento de Lenguaje Natural y aprendizaje automático para analizar, agrupar y optimizar estas historias en un contexto técnico. En las siguientes subsecciones, se detallan las funcionalidades clave del componente, divididas en módulos específicos. La figura 19 presenta el modelo de proceso del componente de arquitectura de microservicios.

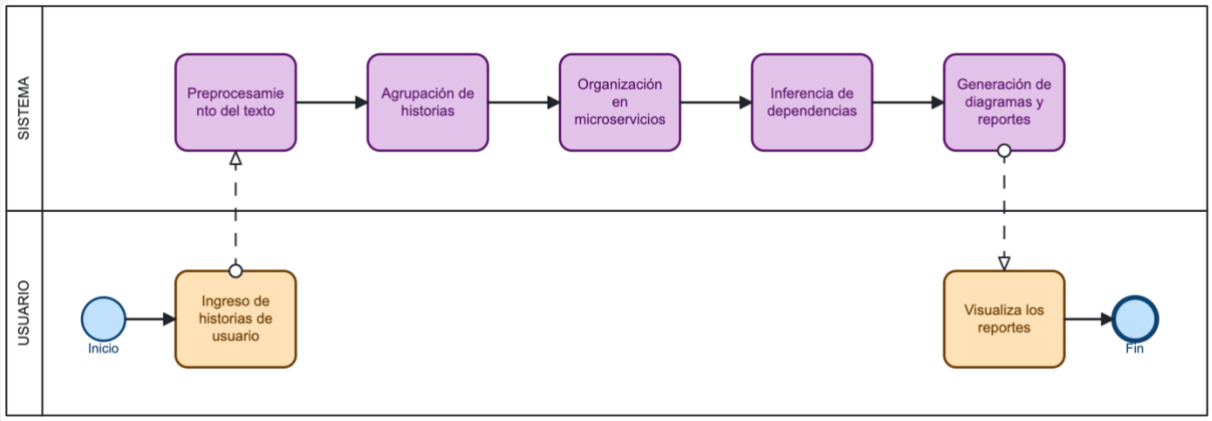


Figura 19. Proceso del componente de arquitectura de microservicios

Preprocesamiento del texto: Lematización y eliminación de ruido

- Función: (preprocess_text), el objetivo de esta funcionalidad es facilitar el análisis semántico, elimina palabras vacías, puntuación y realiza lematización del texto original utilizando la biblioteca SpaCy (en_core_web_lg), proporciona textos normalizados para análisis posterior.
- Función: (format_as_identifier), el objetivo es crear nombres de dominios o identificadores claros y únicos para microservicios, convierte cadenas de texto en identificadores válidos en formato CamelCase, eliminando caracteres no alfanuméricos.

La figura 20 presenta la interfaz de ingreso de datos del componente de arquitectura de microservicios, los datos de ingreso son el caso y el conjunto de historias de usuario (Product Backlog)

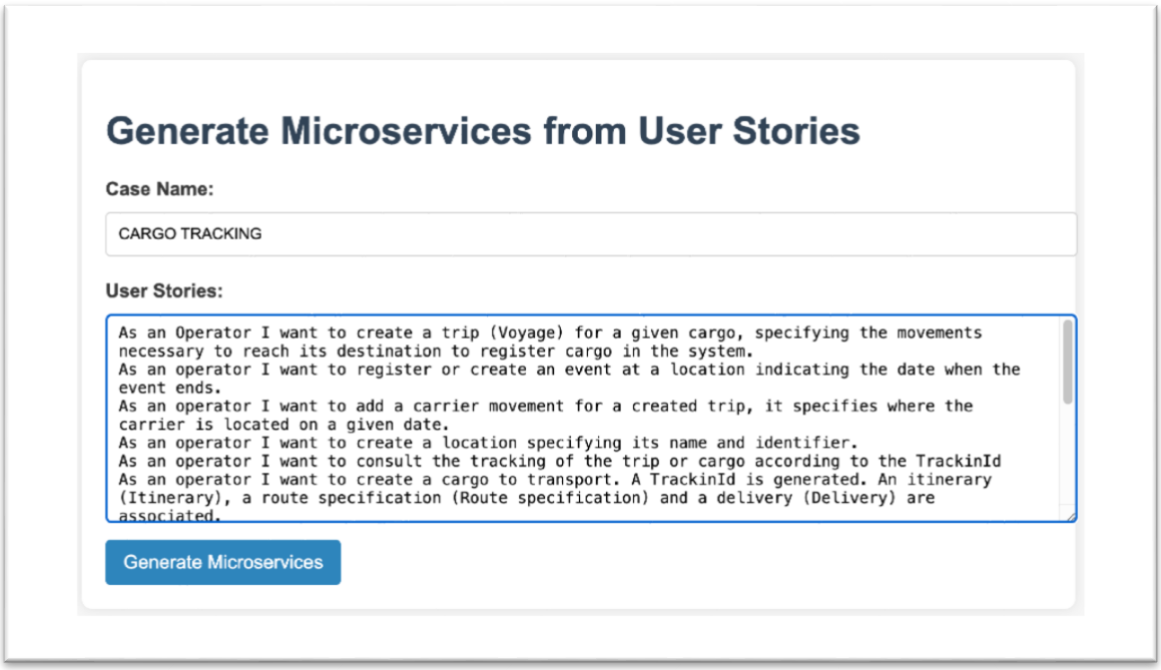


Figura 20. Ingreso de historias de usuario

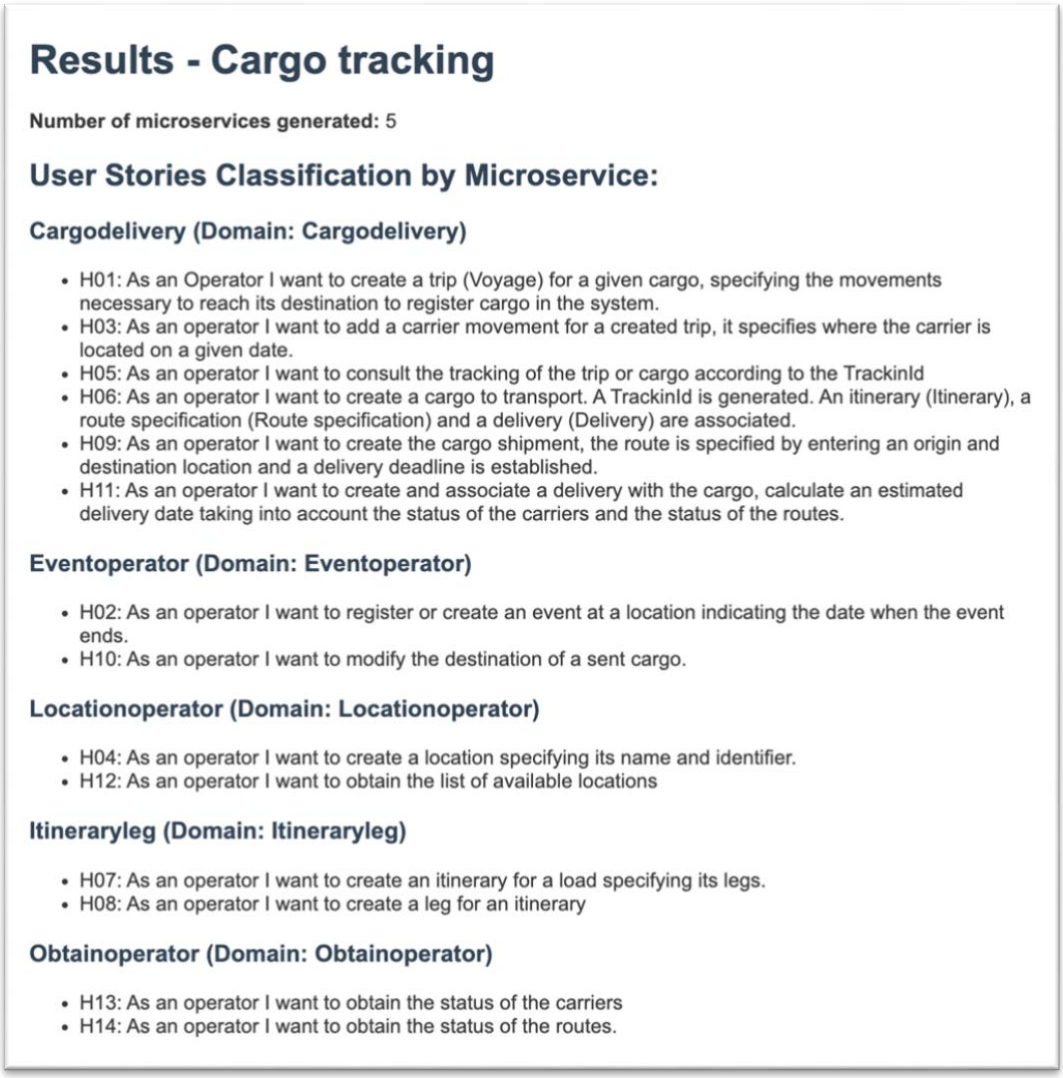
Agrupación de historias de usuario

- Función: (group_stories_with_advanced_nlp), su objetivo es identificar relaciones semánticas entre historias de usuario para organizarlas en funcionalidades.
    - Agrupa historias de usuario en dominios funcionales relacionados utilizando embeddings semánticos generados con el modelo preentrenado SentenceTransformer (all-mpnet-base-v2) para generar representaciones vectoriales de las historias de usuario. Estas representaciones codifican relaciones semánticas y contextuales entre textos.
    - Utiliza los algoritmos de clustering: HDBSCAN (basado en densidad utilizado para agrupar historias de usuario en dominios funcionales, evalúan diferentes configuraciones para determinar el número óptimo de clusters) y Kmeans (alternativa cuando

HDBSCAN no logra un agrupamiento óptimo, evalúa la calidad del agrupamiento mediante el coeficiente de silueta (silhouette_score).

- Función: (generate_dynamic_domain_name)
    - Asigna nombres representativos a los dominios generados utilizando análisis de palabras clave basado en puntajes TF-IDF (Term Frequency-Inverse Document Frequency).
    - Si el análisis TF-IDF no genera términos significativos, utiliza técnicas de resumen basadas en SpaCy.
- Función: (summarize_cluster_stories)
    - Genera resúmenes basados en fragmentos nominales de SpaCy para representar dominios funcionales cuando no se encuentran palabras claves significativas.

La figura 21 presenta un ejemplo de resultados de agrupamiento que se genera en el componente de arquitectura, permite la visualización del número de microservicios y cada microservicio con sus respectivas historias agrupadas por el dominio funcional.



Results - Cargo tracking

Number of microservices generated: 5

User Stories Classification by Microservice:

Cargodelivery (Domain: Cargodelivery)

- H01: As an Operator I want to create a trip (Voyage) for a given cargo, specifying the movements necessary to reach its destination to register cargo in the system.
- H03: As an operator I want to add a carrier movement for a created trip, it specifies where the carrier is located on a given date.
- H05: As an operator I want to consult the tracking of the trip or cargo according to the TrackinId
- H06: As an operator I want to create a cargo to transport. A TrackinId is generated. An itinerary (Itinerary), a route specification (Route specification) and a delivery (Delivery) are associated.
- H09: As an operator I want to create the cargo shipment, the route is specified by entering an origin and destination location and a delivery deadline is established.
- H11: As an operator I want to create and associate a delivery with the cargo, calculate an estimated delivery date taking into account the status of the carriers and the status of the routes.

Eventoperator (Domain: Eventoperator)

- H02: As an operator I want to register or create an event at a location indicating the date when the event ends.
- H10: As an operator I want to modify the destination of a sent cargo.

Locationoperator (Domain: Locationoperator)

- H04: As an operator I want to create a location specifying its name and identifier.
- H12: As an operator I want to obtain the list of available locations

Itineraryleg (Domain: Itineraryleg)

- H07: As an operator I want to create an itinerary for a load specifying its legs.
- H08: As an operator I want to create a leg for an itinerary

Obtainoperator (Domain: Obtainoperator)

- H13: As an operator I want to obtain the status of the carriers
- H14: As an operator I want to obtain the status of the routes.

Figura 21. Resultados del agrupamiento de historias de usuario en microservicios

Organización en microservicios

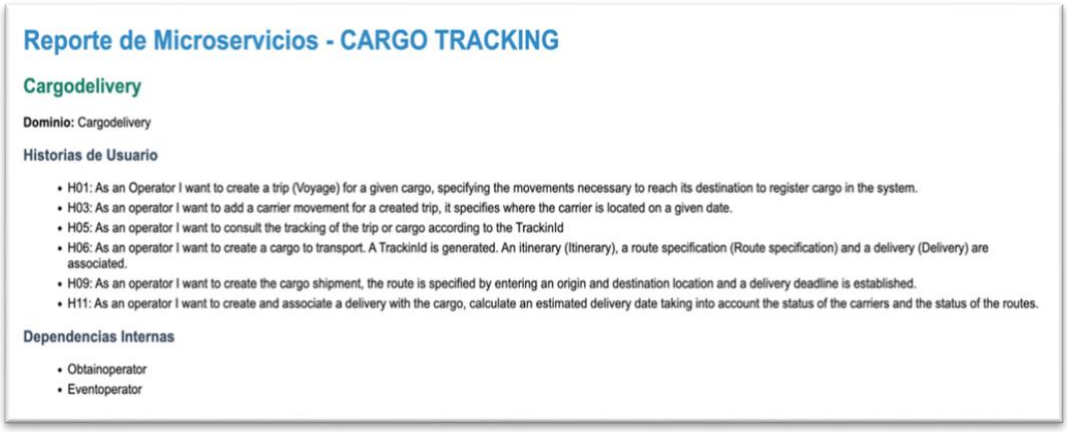
- Función: (group_into_microservices), su objetivo es crear una estructura inicial de microservicios a partir de las funcionalidades.
    - Asigna funcionalidades agrupadas a microservicios, generando una estructura que incluye:
        - Nombre del microservicio.
        - Dominio funcional asociado.
        - Historias de usuarios asignados.

- Dependencias externas y recursos compartidos (inicialmente vacíos).

Inferencia de dependencias

- Función principal: (infer_dependencies), su objetivo es Identificar dependencias lógicas entre microservicios para minimizar el acoplamiento y mejorar la modularidad.
    - Analiza dependencias internas entre microservicios utilizando:
        - Puntajes TF-IDF para identificar términos claves compartidos entre microservicios.
        - Relaciones semánticas en historias de usuario para determinar dependencias basadas en acciones o recursos comunes.
    - Calcula un umbral dinámico para identificar relaciones significativas entre microservicios.

La figura 22 presenta cómo se visualiza el reporte textual de las epecificaciones arquitectónicas, el agrupamiento de las historias en microservicios y las dependencias.


Reporte de Microservicios - CARGO TRACKING

Cargodelivery

Dominio: Cargodelivery

Historias de Usuario

- H01: As an Operator I want to create a trip (Voyage) for a given cargo, specifying the movements necessary to reach its destination to register cargo in the system.
- H03: As an operator I want to add a carrier movement for a created trip, it specifies where the carrier is located on a given date.
- H05: As an operator I want to consult the tracking of the trip or cargo according to the TrackinId
- H06: As an operator I want to create a cargo to transport. A TrackinId is generated. An itinerary (Itinerary), a route specification (Route specification) and a delivery (Delivery) are associated.
- H09: As an operator I want to create the cargo shipment, the route is specified by entering an origin and destination location and a delivery deadline is established.
- H11: As an operator I want to create and associate a delivery with the cargo, calculate an estimated delivery date taking into account the status of the carriers and the status of the routes.

Dependencias Internas

- Obtainoperator
- Eventoperator

Figura 22. Resultados de la inferencia de dependencias.

Generación de representaciones visuales y documentos

- Función: (generate_diagram), su objetivo es facilitar la comprensión visual de la arquitectura generada
    - Genera un diagrama arquitectónico visualizando microservicios y sus dependencias internas/externas.
    - Utiliza NetworkX y Matplotlib para crear gráficos dirigidos.
- Función: (generate_json), su objetivo es proveer una estructura de archivo para su uso en otras herramientas o plataformas
    - Exporta la estructura detallada de microservicios y sus historias asociadas en un archivo JSON.
- Funcionalidad: (generate_html_report), su objetivo es documentar la arquitectura generada de manera clara y presentable
    - Crea un informe HTML detallado que describe cada microservicio, sus funcionalidades y sus dependencias.

La figura 23 describe la arquitectura de microservicios para el sistema "CARGO TRACKING". La arquitectura cuenta con cinco microservicios, cada uno con sus respectivas historias, los microservicios representan los dominios funcionales, es decir, un área de negocio o responsabilidad específica dentro del sistema, las historias están agrupadas por dominios funcionales. Las características representadas son: 1) nodos (microservicios), cada nodo es un microservicio con su nombre y las historias asignadas, ejemplo (Cargodelivery tiene asignadas las historias: H01, H03, H05, H06, H09, H11); y 2) aristas (dependencias), las flechas indican dependencias entre microservicios, ejemplo (Cargodelivery tiene una dependencia con Eventoperator).

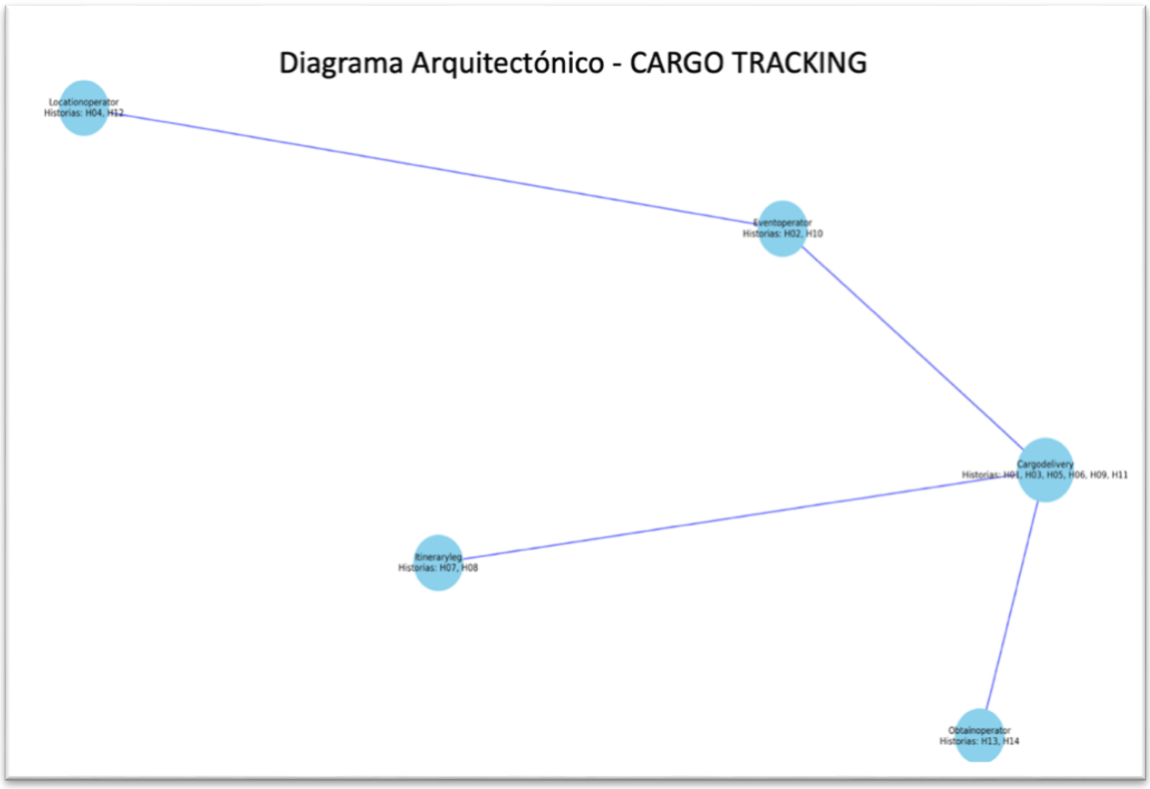


Figura 23. Diagrama arquitectónico de la aplicación

En la figura 24 se puede apreciar un ejemplo de archivo JSON que genera el componte de arquitectura con las especificaciones arquitectónicas de una aplicación.


```
"microservices": [
    {
        "name": "Cargodelivery",
        "domain": "Cargodelivery",
        "functionalities": [
            {
                "code": "H01",
                "text": "As an Operator I want to create a trip (Voyage) for a given cargo, specifying the movements necessary to reach its destination to register cargo in the system."
            },
            {
                "code": "H03",
                "text": "As an operator I want to add a carrier movement for a created trip, it specifies where the carrier is located on a given date."
            },
            {
                "code": "H05",
                "text": "As an operator I want to consult the tracking of the trip or cargo according to the TrackinId"
            },
            {
                "code": "H06",
                "text": "As an operator I want to create a cargo to transport. A TrackinId is generated. An itinerary (Itinerary), a route specification (Route specification) and a delivery (Deliv
ry) are associated."
            },
            {
                "code": "H09",
                "text": "As an operator I want to create the cargo shipment, the route is specified by entering an origin and destination location and a delivery deadline is established."
            },
            {
                "code": "H11",
                "text": "As an operator I want to create and associate a delivery with the cargo, calculate an estimated delivery date taking into account the status of the carriers and the status
of the routes."
            }
        ],
        "external_dependencies": [],
        "shared_resources": [],
        "internal_dependencies": [
            "Obtainoperator",
            "Eventoperator"
        ]
    },
    {
        "name": "Eventoperator",
        "domain": "Eventoperator",
        "functionalities": [
            {
                "code": "H02",
                "text": "As an operator I want to register or create an event at a location indicating the date when the event ends."
            },
            {
                "code": "H10",
                "text": "As an operator I want to modify the destination of a sent cargo."
            }
        ],
        "external_dependencies": [],
        "shared_resources": [],
        "internal_dependencies": [
            "Cargodelivery"
        ]
    },
```


Figura 24. Especificación arquitectónica – archivo JSON

Flujo completo de procesamiento

- Función principal: (process_stories), su objetivo es actuar como punto de entrada principal para procesar historias de usuario.
  - Combina todos los pasos anteriores para procesar historias de usuario desde el inicio hasta la organización en microservicios:
    - Asigna códigos únicos a cada historia de usuario.
    - Agrupa historias en funcionalidades relacionadas.
    - Organiza funcionalidades en microservicios.
    - Infiere dependencias internas entre microservicios.
    - Devuelve el número total de microservicios generados.

Tecnologías y bibliotecas utilizadas

- Procesamiento de lenguaje natural:
  - SpaCy para lematización, análisis de dependencias y extracción de entidades.
  - SentenceTransformer para generar incrustaciones semánticas.
- Agrupamiento:
  - HDBSCAN para agrupamiento basado en densidad.
  - KMeans para agrupamiento basado en particiones.
- Análisis del texto:
  - TfidfVectorizer para calcular la relevancia de términos en documentos.
- Visualización:
  - NetworkX y Matplotlib para generación de diagramas arquitectónicos.
- Otras herramientas:
  - JSON y HTML para exportar datos y reportes.

- **Componente: Aplicación del modelo matemático**

Componente web interactivo diseñado para analizar arquitecturas basadas en microservicios a partir de un archivo JSON que describe los microservicios y sus relaciones. Utiliza Flask como framework principal, combinando funcionalidades avanzadas de análisis y evaluación de métricas arquitectónicas para proporcionar recomendaciones prácticas y visualizaciones que ayudan a optimizar la arquitectura. En las siguientes subsecciones, se detallan las funcionalidades clave del componente:

Carga y validación de archivos

La aplicación permite a los usuarios cargar un archivo JSON que describe la arquitectura de microservicios. Este archivo se valida para asegurarse de que sea del tipo correcto (.json) y que cumpla con los requisitos del sistema.

- Módulo relacionado: (UploadForm, allowed_file)
- Flujo del proceso:
  - El formulario verifica la extensión del archivo.
  - Si el archivo es válido, se guarda temporalmente en el servidor.
  - Si es inválido, se muestra un mensaje de error al usuario.
  - Propósito: Garantizar que los usuarios suban archivos válidos para el análisis.

Carga de microservicios desde archivo JSON

Este módulo procesa el archivo JSON cargado, deserializando los datos y organizándolos en objetos de microservicios con atributos como dependencias internas, externas, funcionalidades y recursos compartidos; es decir, su propósito es transformar los datos del archivo en una estructura de objetos adecuada para el análisis.

- Módulo relacionado: (load_microservices_from_json)
- Flujo del proceso:
  - Crea instancias de la clase Microservice para representar cada microservicio.
  - Establece las relaciones entre microservicios basándose en sus dependencias internas.

La figura 25 muestra cómo el componente de aplicación permite cargar el archivo JSON con las especificaciones arquitectónicas para proceder a calcular las distintas métricas de dependencia y complejidad.

Figura 25. Carga del archivo JSON

Cálculo de métricas arquitectónicas

Este módulo evalúa las diversas métricas arquitectónicas, su propósito es proporcionar una visión cuantitativa de la arquitectura.:

- Módulo relacionado: calculate_metrics y sus submódulos:
  - calculate_ND, calculate_NI, calculate_PD, calculate_AC_est, calculate_LCOH_est
  - calculate_CC_est, calculate_CA_est, calculate_CE_est, calculate_PH_est

- Métricas:
  - **ND:** Número de Dependencias Externas
  - **NI:** Número de Dependencias Internas
  - **PD:** Profundidad de Dependencias
  - **AC_est:** Acoplamiento Interno Estimado
  - **LCOH_est:** Falta de Cohesión
  - **CC_est:** Complejidad Ciclomática
  - **CA_est:** Acoplamiento por Recursos Compartidos
  - **CE_est:** Complejidad Estructural
  - **PH_est:** Puntos de Historia
- Métricas globales:
  - **D_est:** Dependencia estimada.
  - **C_est:** Complejidad estimada.
  - **AI_est:** Índice de Arquitectónica.

La figura 26 muestra un ejemplo del código de las métricas del modelo matemático, en este caso, el código corresponde a la métrica de falta de cohesión.

```
def calculate_LCOH_est(microservices):
    LCOH_list = []
    for ms in microservices:
        all_texts = [preprocess_text(func['text']) for func in ms.functionalities if 'text' in func]
        if not all_texts:
            continue

        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(all_texts)

        if tfidf_matrix.shape[0] > 1:
            similarity_matrix = cosine_similarity(tfidf_matrix)
            upper_tri_indices = np.triu_indices_from(similarity_matrix, k=1)
            mean_similarity = similarity_matrix[upper_tri_indices].mean()
        else:
            mean_similarity = 1.0  # Cohesión máxima con una funcionalidad

        # Ajuste para ampliar el rango de LCOH_est y asegurar un valor mínimo de 1
        LCOH_est = max(1.0, 2.0 - mean_similarity * 1.5)  # Factor de ajuste expandido
        LCOH_list.append(LCOH_est)

    # Calcula y retorna el promedio redondeado, asegurando mínimo de 1
    return round(sum(LCOH_list) / len(LCOH_list), 1) if LCOH_list else 1.0
```

Figura 26. Cálculo de la métrica Falta de cohesión LCOH

Generación de recomendaciones

Su propósito es orientar a los usuarios en la optimización de su arquitectura.

- Módulo relacionado: (analyze_metrics)
- Analiza las métricas calculadas y genera recomendaciones específicas para mejorar la arquitectura, tales como:
  - Reducir dependencias externas o internas.
  - Mejorar la cohesión y disminuir el acoplamiento.
  - Simplificar la complejidad de la arquitectura.
  - Balancear la distribución de funcionalidades.

Visualización de resultados

Presenta los resultados del análisis y las recomendaciones en una interfaz web, su propósito es hacer que los resultados sean accesibles y fáciles de interpretar para los usuarios.

- Módulo relacionado: Plantillas HTML (index.html, resultados.html) y Flask-Bootstrap.
- Utiliza plantillas HTML para mostrar:
  - Métricas detalladas.
  - Recomendaciones para mejorar la arquitectura.
  - Mensajes de error claros en caso de problemas durante el análisis.

La figura 27 muestra un ejemplo del reporte de los resultados de las métricas calculadas y la generación de recomendaciones.

Figura 27. Visualización de resultados y recomendaciones

CAPÍTULO 4

4 EVALUACIÓN DEL MODELO

Esta etapa se centra en analizar y medir qué tan eficaz resultó el modelo como apoyo en la solución del problema planteado, permite sustentar la quinta pregunta de investigación.

P.I. 5: ¿Cómo demostrar que MOAM representa una alternativa eficaz para ayudar a los desarrolladores en la toma de decisiones sobre la definición de las arquitecturas de microservicios?

La estrategia aplicada fue comparar los resultados obtenidos al utilizar el modelo con los objetivos previamente establecidos. Esta tarea exige el uso de técnicas de análisis y métricas adecuadas. Una vez completada, los investigadores tienen la potestad de decidir si es necesario regresar al paso de "diseño" para perfeccionar la solución, o si pueden avanzar al siguiente paso, que consiste en la comunicación, destacando de manera propositiva las mejoras realizadas y sugiriendo posibles áreas de exploración para futuros estudios [108].

Siguiendo la propuesta de la Ciencia del Diseño de [109], el proceso de evaluación de un modelo debe seguir ciertas directrices clave que garantizan la calidad y relevancia tanto del modelo como de los resultados de la investigación. Desde la perspectiva de [108], los modelos se evalúan según la armonía que guardan con los hechos del mundo real, por tanto, una forma de evaluación es a

través del método observacional aplicando los métodos analíticos y casos de estudios.

4.1 PROCESO DE EVALUACIÓN

4.1.1 MÉTODOS DE EVALUACIÓN

Se utilizaron métodos analíticos para abordar tanto las relaciones estructurales entre los componentes (métricas de dependencias) como las métricas que reflejan la complejidad del sistema.

Se hizo análisis a través de implementaciones en casos de estudio hipotéticos y reales en los que se utilizó MOAM, posteriormente se aplicaron métodos analíticos no solo para estudiar los resultados y compararlos con otros trabajos relacionados, sino también, de forma dinámica para verificar los algoritmos de preprocesamiento de texto, de agrupamiento y de cálculo de las métricas de dependencia y complejidad.

La evaluación se llevó a cabo de la siguiente manera:

1. Análisis de los casos de estudio
2. Especificación del conjunto de historias de usuario de cada caso (product backlog)
3. Ejecución del primer componente de MOAM para obtener el diagrama arquitectónico de cada caso con las historias agrupadas (técnicas de NLP y clustering) y la inferencia de las dependencias internas y externas.
4. Descargar el archivo JSON con la especificación arquitectónica de la arquitectura

5. Ejecución del segundo componente de MOAM para obtener el cálculo de las métricas de dependencia y complejidad de la arquitectura (aplicación del modelo matemático)
6. Comparación de resultados con otros trabajos relacionados
7. Verificación de resultados

4.1.2 MÉTRICAS DE EVALUACIÓN

Para el proceso de evaluación únicamente se tomaron en cuenta aquellas fórmulas que son similares a las utilizadas por las otras propuestas, estas son: número de microservicios en el sistema, acoplamiento total, falta de cohesión, mayor número de historias asociadas a un microservicio, número total de llamadas. Sin embargo, la representación matemática completa del modelo para la especificación de la dependencia y complejidad en arquitecturas de microservicios, se encuentra en la sección 3.2.3.2.

A continuación, se describen las métricas utilizadas para la evaluación del modelo.

- **Número de microservicios en el sistema:** mide la cantidad total de microservicios identificados y definidos en el sistema.
- **Acoplamiento total**: mide el nivel de dependencia entre los microservicios, evaluando cuántos interactúan directamente entre sí.
- **Falta de cohesión**: Evalúa qué tan estrechamente relacionadas están las funcionalidades dentro de un microservicio.
- **Mayor número de historias asociadas a un microservicio**: Indica el microservicio que tiene el mayor número de historias de usuario asociadas
- **Número total de llamadas**: Representa el total de interacciones (llamadas) entre microservicios dentro del sistema.

Las técnicas avanzadas de procesamiento de lenguaje natural (NLP) y clustering para organizar historias de usuario en microservicios. Emplea spaCy para lematización y limpieza de texto, junto con SentenceTransformer para generar embeddings semánticos que capturan el contexto de las historias. Adicionalmente, utiliza TF-IDF para extraer palabras clave y, cuando es necesario, chunks nominales para identificar términos representativos. Estas representaciones semánticas se integran con algoritmos de clustering como HDBSCAN y K-Means, evaluando la calidad de los clusters mediante el puntaje de silueta, y asegurando una asignación efectiva de historias a microservicios.

Se automatiza la generación de nombres de dominio, organización de dependencias internas y representación visual de arquitecturas mediante NetworkX. Destaca por su robustez en el preprocesamiento textual y uso de técnicas modernas de embeddings y clustering, siendo ideal para el diseño de arquitecturas orientadas a microservicios. Aunque presenta una infraestructura sólida, puede beneficiarse de métricas adicionales como Davies-Bouldin index y reglas enriquecidas para nombres de dominio. Es una herramienta avanzada para el análisis y diseño basado en historias de usuario.

Para evaluar el modelo se utilizó la estrategia descrita en [107], los casos (Cargo Tracking y JPet Store) como dos ejemplos que otorga el estado del arte y dos proyectos típicos seleccionados de manera especifica para esta validación (Bank of Anthos y Hipster Shop). La evaluación comparó los cálculos obtenidos por MOAM con los resultados de las métricas similares dadas por otros métodos, como son: (MB - Genetic Algorithm) [107], Diseño basado en el dominio (DDD) [110], Service Cuttuer [111], Identificación de Microservicios a través de Análisis de Interfaz (MITIA) [112] e Identificación de Candidatos a Servicios de Sistemas Monolíticos basados en Trazas de Ejecución (Execution Traces) [111].

El procedimiento de evaluación para todos los casos es el siguinte:

Con el product backlog definido (conjunto de historias de usuario), se carga el Módulo de Arquitectura y se procede a generar el diagrama arquitectónico de la aplicación, de forma automática, a través de las técnicas de NLP y de clustering, se obtienen: 1) la visualización del diagrama arquitectónico, 2) archivo JSON con las especificaciones de la arquitectura (nombre, dominio, dependencias internas y externas) y, 3) reporte HTML con un resumen del agrupamiento más la inferencia de las dependencias internas y externas. Con el archivo JSON que contiene la especificación de la arquitetura, se carga el Módulo del Modelo Matemático, el módulo lee el archivo JSON y procede al cálculo de cada una de las métricas de dependencia y complejidad, aplica para ello, las fórmulas matemáticas, técnicas de NLP y teoría de grafos, finalmente se obtienen los resultados y las recomendaciones.

4.1.3 CASOS DE ESTUDIO

La selección de los casos de estudio para validar el modelo MOAM combina una estrategia basada en la relevancia científica y la representatividad práctica. Por un lado, los casos Cargo Tracking y JPetStore han sido utilizados ampliamente en investigaciones previas para evaluar herramientas similares, lo que garantiza una base sólida para la comparabilidad directa con otras propuestas. Al trabajar con estos casos, se asegura la validez metodológica del modelo, permitiendo que los resultados sean contrastables y replicables, un factor clave para fomentar la aceptación académica y la credibilidad del enfoque.

Por otro lado, los casos Bank of Anthos y Hipster Shop representan arquitecturas típicas de microservicios en entornos reales, con características como escalabilidad, múltiples dependencias y flexibilidad. Estos casos permiten

evaluar la aplicabilidad del modelo en contextos industriales, simulando desafíos prácticos y asegurando que los resultados puedan generalizarse a sistemas complejos del mundo real. Al combinar ambos enfoques, se obtiene un balance ideal entre rigor científico y relevancia práctica, fortaleciendo la robustez del modelo MOAM y su utilidad tanto en la academia como en la industria.

4.1.3.1 *Cargo Tracking*

La aplicación Cargo Tracking tiene finalidad de trasladar una carga, identificada por un TrackingId, desde un punto de origen hasta un destino siguiendo una ruta específica. Cuando la carga está disponible, se asigna a uno de los itinerarios, que consisten en listas de movimientos de transportistas seleccionados entre los viajes disponibles. Los eventos de manipulación (HandlingEvents) supervisan el avance de la carga a lo largo del itinerario, mientras que el proceso de entrega proporciona información sobre su estado actual, la hora estimada de llegada y si se encuentra en tránsito [112] .

Una aplicación de cargo tracking permite gestionar y registrar cargas con identificadores únicos, planificar rutas optimizadas asignando itinerarios específicos, y monitorear en tiempo real el progreso de las cargas a lo largo de su trayecto, registrando eventos clave como carga, descarga o inspección. Además, ofrece visualización detallada del estado de la carga, predicción de tiempos estimados de llegada (ETA), generación de alertas y notificaciones sobre cambios, retrasos o incidentes, así como reportes y análisis de desempeño logístico. También facilita la integración con sistemas externos mediante APIs, asegurando trazabilidad completa y control de acceso para proteger los datos. La tabla 11 muestra el Product Backlog de caso.

Tabla 11. Product Backlog - Cargo Tracking

ID.	DETALLE DE LA HISTORIA
HU01	As an Operator I want to create a trip (Voyage) for a given cargo, specifying the movements necessary to reach its destination to register cargo in the system.
HU02	As an operator I want to register or create an event at a location indicating the date when the event ends.
HU03	As an operator I want to add a carrier movement for a created trip, it specifies where the carrier is located on a given date.
HU04	As an operator I want to create a location specifying its name and identifier.
HU05	As an operator I want to consult the tracking of the trip or cargo according to the TrackinId
HU06	As an operator I want to create a cargo to transport. A TrackinId is generated. An itinerary (Itinerary), a route specification (Route specification) and a delivery (Delivery) are associated.
HU07	As an operator I want to create an itinerary for a load specifying its legs.
HU08	As an operator I want to create a leg for an itinerary
HU09	As an operator I want to create the cargo shipment, the route is specified by entering an origin and destination location and a delivery deadline is established.
HU10	As an operator I want to modify the destination of a sent cargo.
HU11	As an operator I want to create and associate a delivery with the cargo, calculate an estimated delivery date taking into account the status of the carriers and the status of the routes.
HU12	As an operator I want to obtain the list of available locations
HU13	As an operator I want to obtain the status of the carriers
HU14	As an operator I want to obtain the status of the routes.

En el caso "Cargo tracking", se generaron 5 microservicios relacionados con los dominios funcionales de la aplicación, el agrupamiento se dió de la siguiente manera (Tabla 12):

Tabla 12. Resumen de agrupamiento por microservicios – Cargo Tracking

Agrupamiento de historias (microservicio y dominio relacionado)				
Cargodelivery (Domain: Cargodelivery)	Eventoperator (Domain: Eventoperator)	Locationoperator (Domain: Locationoperator)	**Itineraryleg (Domain: Itineraryleg)**	**Obtainoperator (Domain: Obtainoperator)**
HU01	HU02	HU04	HU07	H13
HU03	HU010	HU12	HU08	H13
HU05				
HU06				

HU09	
HU11	

La tabla 13 indica que, en el caso "Cargo Tracking", el método MOAM se distingue como el más modular, con un total de 5 microservicios, superando a los demás métodos en esta característica. Sin embargo, este aumento en la cantidad de microservicios viene acompañado de un acoplamiento total alto (6.5), lo que indica una mayor interdependencia entre los componentes del sistema. Comparado con métodos como Genetic y Service Cutter, que presentan acoplamientos significativamente menores (3.16), MOAM podría introducir desafíos en la gestión de dependencias. Por otro lado, el número de historias asociadas al microservicio más demandado en MOAM es de 6, una cifra moderada y comparable con Genetic y DDD, lo que sugiere que la carga funcional está mejor distribuida en comparación con métodos como Service Cutter, donde esta métrica alcanza un valor de 10. Además, MOAM presenta un total de 6 llamadas entre microservicios, que es el segundo valor más bajo después de Genetic, lo que implica un nivel razonable de comunicación entre componentes sin llegar a la complejidad observada en MITIA (12 llamadas).

Tabla 13. Comparación de resultados con otros métodos – Cargo Tracking

Proyecto	Método	Número de microservicios	Falta de cohesión	Acoplamiento total	Mayor número de historias asociadas a un microservicio	Número total de llamadas
		N	***CohT***	**CpT**	**W**	**Cl**
Cargo Tracking	Genetic	3	1.16	3.16	6	3
	DDD	4	1.50	5.29	6	9
	Sevice Cutter	3	1.15	3.16	10	8
	MITIA	4	1.06	6.78	5	12
	MOAM	**5**	**1.5**	**6.5**	**6**	**6**

Se puede concluir que MOAM prioriza la modularidad al incrementar la cantidad de microservicios, pero esto viene acompañado de un acoplamiento

elevado. Este método podría ser más adecuado en contextos donde se busca escalabilidad y segmentación de funciones, siempre que se gestionen adecuadamente las dependencias para evitar complicaciones en la integración y mantenimiento del sistema.

4.1.3.2 *JPet-Store*

La aplicación JPet-Store es una tienda en línea especializada en productos para mascotas. Permite a los usuarios explorar y buscar en su catálogo, seleccionar artículos para añadir al carrito de compras, realizar modificaciones en el carrito y proceder con los pedidos de los productos seleccionados. Muchas de estas acciones pueden realizarse sin necesidad de registrarse o iniciar sesión. Sin embargo, para completar un pedido, es obligatorio iniciar sesión, lo que requiere haber creado previamente una cuenta en la aplicación mediante el proceso de registro. A continuación, se detallan las funcionalidades principales de la JPet-Store [113].

Una tienda en línea permite a los usuarios navegar por su catálogo de productos para mascotas, buscar artículos específicos, añadirlos al carrito de compras, modificar su contenido y realizar pedidos. Aunque muchas acciones, como la exploración del catálogo y la gestión del carrito, pueden realizarse sin registrarse, para completar un pedido es necesario crear una cuenta y acceder mediante inicio de sesión. Estas funcionalidades principales hacen que la JPet-Store ofrezca una experiencia completa y flexible para la compra de productos para mascotas. La tabla 14 muestra el Product Backlog del caso JPet-Store.

Tabla 14. Product Backlog - JPetStore

ID.	DETALLE DE LA HISTORIA
HU01	As a user I need to list the products or pets that belong to a category
HU02	As a user I need to list the categories available in the system
HU03	As a user I need to search for products or pets specifying a name
HU04	As a user I need to see the items of a selected product
HU05	As a user I need to see the details of an item, a text or name and a visual description of the item and the number of that item in stock are displayed
HU06	As a user I need to add an item or item to the shopping cart
HU07	As a user I need to remove an item from the shopping cart
HU08	As a user I need to adjust the quantity of an item in the shopping cart
HU09	As a user I need to obtain the items from the shopping cart
HU10	As a user I need to create a new order from the items selected in the shopping cart
HU11	As a user I need to obtain the data of an order by its id
HU12	As a user I need to obtain and assign a new number or ID of the order
HU13	As a user I need to list the orders that I have made specifying the username
HU14	As a system I need to verify if the user is authenticated in order to create an order, otherwise he must register and authenticate
HU15	As a user I need to register in the system providing a username and password
HU16	As a user I need to obtain my account information in the system by providing my username
HU17	As a user, I need to log out of the system
HU18	As a user I need to modify my account data (username or password) in the system
HU19	As a user I need to obtain the details of a category by entering its id
HU20	As a user I need to obtain the details of a product by entering its id
HU21	As a user I need to verify if there are available units of the item or article
HU22	As a user I need to authenticate myself in the system by providing my username and password

En el caso "JPetStore", se generaron 5 microservicios relacionados con los dominios funcionales de la aplicación, el agrupamiento se dió de la siguiente manera (Tabla 15):

Tabla 15. Resumen de agrupamiento por microservicios – JPetStore

Agrupamiento de historias (microservicio y dominio relacionado)				
Categorylist (Domain: Categorylist)	**Cartshopping (Domain: Cartshopping)**	**Logarticle (Domain: Logarticle)**	**Obtaindatum (Domain: Obtaindatum)**	**Usernamepassword (Domain: Usernamepassword)**
HU01	HU04	HU05	HU11	HU14
HU02	HU06	HU17	HU12	HU15
HU03	HU07	HU20	HU13	HU16
	HU08			HU18
	HU09			HU22
	HU10			
	HU19			
	HU20			

En la Tabla 16. "Comparación de resultados con otros métodos – JPetStore", el método MOAM destaca como el más modular, con 5 microservicios, igualando al método Genetic, pero superando a DDD y Execution Traces, que cuentan con 4 microservicios cada uno. Sin embargo, MOAM también presenta el mayor nivel de acoplamiento total (CpT = 5.33), lo que indica una mayor interdependencia entre los servicios. Este nivel de acoplamiento podría generar desafíos en la flexibilidad y mantenimiento del sistema, especialmente en comparación con el método Genetic, que tiene el acoplamiento más bajo (CpT = 1.41).

En cuanto al mayor número de historias asociadas a un microservicio (W), MOAM mantiene un equilibrio con DDD (8 historias), aunque es superado por Genetic, que concentra hasta 9 historias, lo que podría indicar un potencial problema de sobrecarga funcional en ese método. Por otro lado, el número total de llamadas entre microservicios (Cl) en MOAM es intermedio (6), menor que en DDD (9) y Execution Traces (8), pero superior a Genetic (3), lo que refleja un diseño de comunicación moderado que busca un balance entre eficiencia y modularidad.

Tabla 16. Comparación de resultados con otros métodos – JPetStore

Proyecto	Método	Número de microservicios	Falta de cohesión	Acoplamiento total	Mayor número de historias asociadas a un microservicio	Número total de llamadas
		N	***CohT***	**CpT**	**W**	**Cl**
Jpet Store	Genetic	5	1.79	1.41	9	3
	DDD	4	1.50	3.46	8	9
	Execution Traces	4	1.50	3.46	7	8
	MOAM	**5**	**1.50**	**5.33**	**8**	**6**

En conclusión, MOAM prioriza la modularidad y la distribución funcional, pero a costa de un mayor acoplamiento, lo que podría limitar la capacidad del sistema para adaptarse a cambios o manejar escalabilidad. Dependiendo de los requisitos del sistema, como la necesidad de mayor independencia entre microservicios, podría ser necesario optimizar su diseño para reducir el acoplamiento sin comprometer la funcionalidad.

4.1.3.3 *Bank of Anthos*

El sistema Bank of Anthos, es uno de los casos que se seleccionó para aplicar MOAM, éste a comparación de Cargo Tranking y JpetStore no fueron probados por los otros métodos, al igual que Hipster Shop, sin embargo, es importante considerar que, siempre que se cuente con el Product Backlog de un caso, éste se puede probar con MOAM.

Bank Anthos es una aplicación bancaria en línea basada en microservicios que permite realizar transferencias, ver transacciones y administrar cuentas. Creada para probar sistemas distribuidos y prácticas de DevOps. La tabla 17 presenta el Product Backlog del caso Bank of Anthos.

Tabla 17. Product Backlog del caso Bank of Anthos

ID.	DETALLE DE LA HISTORIA
HU01	As a user, I want to create a bank account so that I can start using the bank's services.
HU02	As a user, I want to log in to access my account securely.
HU03	As a user, I want to see the balance of my accounts to know my financial situation.
HU04	As a user, I want to transfer funds between my accounts to manage my finances.
HU05	As a user, I want to transfer money to other accounts to make payments or send money to friends.
HU06	As a user, I want to see a history of my transactions to review my expenses and income.
HU07	As a user, I want to receive notifications when transactions are made on my account to keep track of movements.
HU08	As a user, I want to report suspicious transactions to protect my funds.
HU09	As a user, I want to update my personal information to keep my details up to date.
HU10	As an administrator, I want to monitor transactions to detect and prevent fraud.
HU11	As an administrator, I want to access transaction reports to evaluate the bank's performance.
HU12	As an administrator, I want to manage the configuration of services to keep the infrastructure up and running.

MOAM, agrupó a Bank of Anthos en 3 microservicios: Wantbank (Domain: Wantbank), con 5 historias de usuario, Wanttransaction (Domain: Wanttransaction), con 5 historias de usuario y Administratortransaction (Domain: Administratortransaction), con 2 historias de usuario (Tabla 18).

Tabla 18. Resumen de agrupamiento por microservicios – Bank of Anthos

Agrupamiento de historias (microservicio y dominio relacionado)		
Wantbank (Domain: Wantbank)	**Wanttransaction (Domain: Wanttransaction)**	**Administratortransaction (Domain: Administratortransaction)**
HU01	HU02	HU10
HU03	HU07	HU11
HU04	HU08	
HU05	HU09	
HU06	HU12	

4.1.3.4 *Hipster Shop*

La aplicación "Hipster Shop" es una aplicación de demostración creada por Google Cloud Platform para mostrar cómo implementar y gestionar un sistema basado en microservicios. La aplicación simula una tienda en línea y está compuesta por múltiples microservicios que colaboran para proporcionar funcionalidades como el catálogo de productos, el carrito de compras, la gestión de pagos y el procesamiento de pedidos. Cada servicio se implementa utilizando una variedad de lenguajes y tecnologías, lo que demuestra la flexibilidad de la arquitectura de microservicios. La tabla 19 presenta el Product Backlog del caso Hipster Shop.

Tabla 19. Product Backlog del caso Hipster Shop

ID.	DETALLE DE LA HISTORIA
HU01	As a user, I want to browse the product catalog to see the different items available.
HU02	As a user, I want to see the details of a specific product to get more information about it.
HU03	As a user, I want to see related product recommendations as I browse, to discover additional items that interest me.
HU04	As a user, I want to add products to the shopping cart to save the items I am interested in purchasing.
HU05	As a user, I want to review and modify the products in my cart to make sure I have the items I want to purchase.
HU06	As a user, I want to complete the purchase of the products in my cart to complete the transaction.
HU07	As a user, I want to receive an email confirmation of my order to have a record of my purchase.
HU08	As a user, I want to be able to choose between different payment methods to conveniently complete my purchase.
HU09	As a user, I want to select a shipping method to receive my order according to my time and cost preferences.
HU10	As a user, I want to be able to track my order after I make my purchase so I know when it will arrive.
HU11	As a user, I want to see product prices in my local currency to better understand costs.
HU12	As a user, I want to read other users' reviews and ratings of a product to make an informed buying decision

HU13	As a user, I want to leave a review and rating for a product I have purchased to share my experience.
HU14	As a user, I want to create an account and access my profile to manage my orders and shopping preferences.
HU15	As a user, I want to log in and log out of my account to securely access my personal information and orders.
HU16	As an administrator, I want to manage product inventory to ensure catalog information is up to date.
HU17	As an administrator, I want to see a report of sales made to understand the store's performance.

En el caso de Hipster Shop, MOAM generó 7 microservicios, según los dominios funcionales agrupó las historias entre 2 y 4 por microservicio (Tabla 20).

Tabla 20. Comparación de resultados con otros métodos – Hipster Shop

Agrupamiento de historias (microservicio y dominio relacionado)						
Wantspecific (Domain: Wantspecific)	Wantunderstand (Domain: Wantunderstand)	Wantcomplete (Domain: Wantcomplete)	Purchasewant (Domain: Purchasewant)	Methodwant (Domain: Methodwant)	Ratingreview (Domain: Ratingreview)	Logwant (Domain: Logwant)
HU01	HU03	HU04	HU07	HU08	HU12	HU14
HU02	HU11	HU05	HU10	HU09	HU13	HU15
	HU16	HU06				
	HU17					

4.1.4 ANÁLISIS DE RESULTADOS

En esta sección se analizan los resultados de cada métrica comparada, el análisis se desprende de la Figura 24:

- **Número de Microservicios:** La cantidad de microservicios varía según el método y el caso de estudio. En Cargo Tracking, los métodos DDD y MOAM generan más microservicios (4), mientras que Genetic y Service Cutter producen menos (3). En Jpet Store, el número de microservicios es

más alto en general, con Genetic y MOAM alcanzando el máximo de 5, lo que indica que algunos métodos tienden a segmentar más funcionalidad en sistemas más complejos.

- **Falta de Cohesión:** La falta de cohesión ($CohT$) refleja las diferencias en cómo los métodos manejan la cohesión interna. En ambos casos de estudio, Service Cutter tiene la menor falta de cohesión (1.06), lo que indica un diseño más cohesivo. Genetic muestra la mayor falta de cohesión en Jpet Store (1.79), sugiriendo mayor fragmentación en este contexto. DDD, MOAM y Execution Traces mantienen valores intermedios, indicando una estrategia balanceada entre cohesión y separación funcional.
- **Acoplamiento Total**: Los valores de acoplamiento (CpT) varían significativamente entre los métodos. En Cargo Tracking, MITIA y MOAM tienen los acoplamientos más altos (6.78 y 6.5, respectivamente), lo que sugiere una mayor interdependencia entre servicios. En Jpet Store, el acoplamiento tiende a ser más bajo en general, con Genetic alcanzando el valor mínimo (1.41), reflejando un diseño con menor conexión entre microservicios.
- **Mayor Número de Historias Asociadas**: Esta métrica (W) es relativamente consistente en ambos casos de estudio, con la mayoría de los métodos manejando entre 6 y 8 historias asociadas. En Cargo Tracking, MOAM soporta más historias (8), mientras que otros métodos como Genetic y Service Cutter soportan menos (6). En Jpet Store, todos los métodos alcanzan valores similares, reflejando una capacidad uniforme para manejar requisitos funcionales en este sistema.
- **Número Total de Llamadas:** En Cargo Tracking, los métodos MOAM y MITIA tienen los valores más altos de llamadas totales (12 y 12), lo que indica una mayor interacción entre servicios. En Jpet Store, los valores son

más moderados, con MOAM y Execution Traces manejando 6 llamadas, mientras que Genetic muestra el menor número (3) en ambos casos, lo que podría reflejar un diseño más simple o menos interdependiente.

La figura 28 muestra un conjunto de gráficos que permiten comparar los resultados de cada propuesta por caso de estudio.

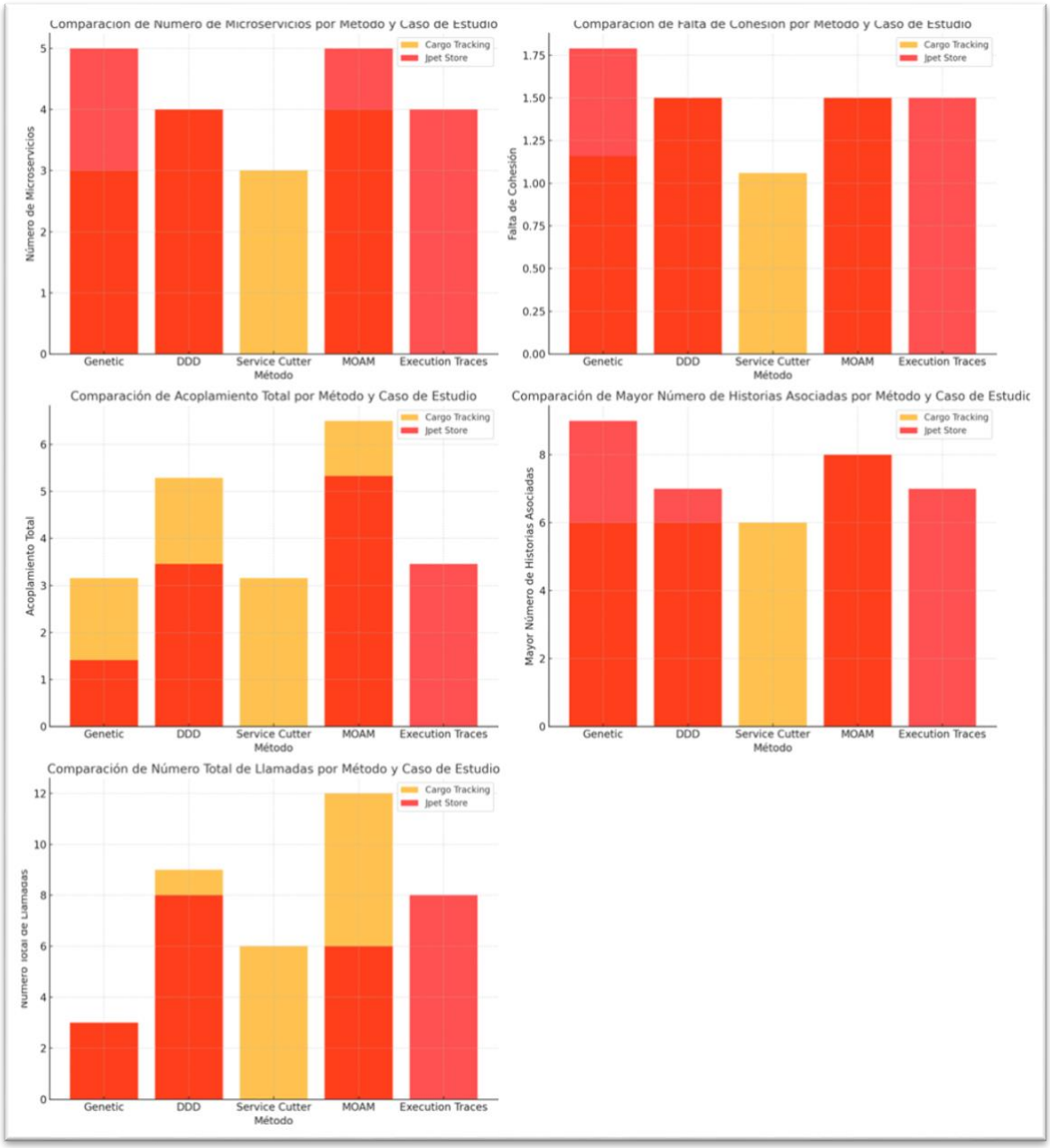


Figura 28. Comparación de resultdos por métrica y caso

4.1.4.1 *Análisis Detallado de Acoplamiento y Falta de Cohesión*

Acoplamiento Total (CpT): El acoplamiento total mide la interdependencia entre los microservicios dentro del sistema. En Cargo Tracking, los métodos MITIA (6.78) y MOAM (6.5) tienen los valores más altos, indicando una arquitectura con conexiones más frecuentes o necesarias entre los componentes. Esto puede generar mayor complejidad en el mantenimiento del sistema, pero podría facilitar la integración si los servicios están bien diseñados. En contraste, Genetic y Service Cutter tienen un acoplamiento más bajo (3.16), lo que puede significar menor interacción entre microservicios, aunque a costa de menor cohesión funcional. En Jpet Store, los valores de acoplamiento disminuyen en general; Genetic muestra el acoplamiento más bajo (1.41), mientras que MOAM y Execution Traces se mantienen en niveles intermedios (5.33 y 3.46, respectivamente), lo que puede reflejar una arquitectura más equilibrada para este sistema.

Falta de Cohesión (CohT): La falta de cohesión refleja el grado de fragmentación funcional en los microservicios. En ambos casos de estudio, Service Cutter tiene la menor falta de cohesión (1.06), lo que sugiere que sus microservicios están bien definidos y son altamente cohesivos. En el otro extremo, Genetic tiene el mayor valor de falta de cohesión en Jpet Store (1.79), indicando una fragmentación significativa y, posiblemente, servicios menos relacionados funcionalmente. DDD, MOAM y Execution Traces mantienen valores intermedios (1.5), lo que refleja un equilibrio entre cohesión y separación funcional. Esto indica que estos métodos pueden ser adecuados para casos donde es necesario dividir funciones complejas sin perder cohesión interna.

Un análisis conjunto de estas métricas revela un patrón interesante: los métodos con mayor acoplamiento (CpT) tienden a mantener una falta de

cohesión moderada ($CohT$). Por ejemplo, MOAM y DDD muestran valores relativamente altos en ambas métricas, lo que podría indicar que están diseñados para manejar casos de estudio más complejos, donde las interdependencias y la separación funcional son inevitables. Por otro lado, Service Cutter destaca por su capacidad para mantener bajo el acoplamiento y la falta de cohesión, lo que lo convierte en una opción viable para sistemas donde la simplicidad y la cohesión interna son prioritarias.

CAPÍTULO 5

5 CONCLUSIONES

5.1 CONCLUSIONES Y CONTRIBUCIONES

La investigación logró diseñar y validar el MOAM (Microservices Optimization Architectures Model), un modelo inteligente basado en técnicas de Machine Learning y Procesamiento de Lenguaje Natural. Este modelo permite identificar y medir de manera efectiva la dependencia y la complejidad en el diseño de microservicios, abordando así un vacío crítico en el campo. MOAM demostró ser eficaz para guiar decisiones arquitectónicas que optimizan tanto la flexibilidad como la escalabilidad de los sistemas.

A través de la revisión sistemática de la literaturna y las encuestas aplicadas a los desarrolladores de microservicios, se identificaron los principales desafíos relacionados con dependencia, complejidad y atributos de calidad en microservicios. Estos hallazgos confirman que los problemas de acoplamiento excesivo y la falta de cohesión interna en los microservicios, afectan negativamente la flexibilidad y escalabilidad de los sistemas, estableciendo de esta manera una base fundamentada para desarrollar soluciones específicas como la propuesta en esta tesis.

Los resultados del estudio evidencian que la complejidad y la dependencia están directamente relacionadas con la flexibilidad y escalabilidad de las aplicaciones. Un diseño que optimiza estas variables mejora significativamente

el desempeño general del sistema y su capacidad para adaptarse a cambios, lo que es crucial en entornos dinámicos y de alta demanda.

La validación del modelo mediante estudios de caso y experimentos controlados confirmó su utilidad en entornos reales. MOAM no solo proporciona métricas precisas y comprensibles para evaluar arquitecturas, sino que también es aplicable en diversas etapas del ciclo de vida del software, facilitando la transición de diseños monolíticos a arquitecturas basadas en microservicios o, la decisión, desde la concepción de un sistema, de desarrollarlo con esta arquitectura.

Este trabajo no solo ha cumplido con los objetivos planteados, sino que ha dejado una serie de contribuciones adicionales que amplían su impacto en el campo de las arquitecturas de software y microservicios. Estas son:

- Establece una base teórica y práctica para la mejora continua en el diseño de microservicios, combinando métodos ágiles y herramientas automatizadas. Las conclusiones refuerzan la importancia de una evaluación estructurada y adaptable, integrando prácticas de DevOps, análisis de calidad y toma de decisiones informada en arquitecturas distribuidas.
- La propuesta de un modelo matemático para medir la dependencia y la complejidad constituye una contribución importante que puede inspirar nuevas investigaciones orientadas a la cuantificación y optimización de otros criterios arquitectónicos.
- Al integrar Machine Learning y NLP en el diseño y evaluación de microservicios, este trabajo demuestra el potencial de la inteligencia artificial para transformar la manera en que se diseñan y gestionan sistemas complejos.

- Al destacar cómo las decisiones arquitectónicas en las etapas iniciales impactan significativamente en la calidad y mantenibilidad de los sistemas, este trabajo sensibiliza a los desarrolladores y arquitectos sobre la importancia de un diseño bien fundamentado.

5.2 LIMITACIONES Y TRABAJOS FUTUROS

- El modelo MOAM utiliza técnicas de Machine Learning y Procesamiento de Lenguaje Natural (NLP) que requieren datos bien etiquetados y representativos. Una limitación es la necesidad de disponer de descripciones arquitectónicas o historias de usuario de calidad para garantizar resultados precisos, lo que puede no estar disponible en todos los proyectos.
- Aunque el modelo ha sido validado en casos de estudio, su aplicación en proyectos con arquitecturas altamente heterogéneas o que empleen tecnologías emergentes podría requerir ajustes adicionales. Esto limita su capacidad de generalización sin modificaciones.
- La evaluación de sistemas extremadamente complejos o con miles de microservicios puede presentar desafíos en términos de tiempo de procesamiento y precisión, debido a la cantidad de relaciones y dependencias que deben analizarse.
- El modelo está diseñado para ser más útil en las etapas iniciales del diseño arquitectónico, lo que puede limitar su efectividad para proyectos ya implementados o en fases avanzadas de desarrollo.
- A pesar de proporcionar métricas precisas, la interpretación de los resultados y la implementación de las recomendaciones arquitectónicas aún dependen del juicio experto, lo que puede dificultar su adopción en equipos con menos experiencia.

Las siguientes propuestas de trabajos futuros proporcionan una hoja de ruta clara para expandir y mejorar las contribuciones del trabajo:

- Se propone expandir el conjunto de datos de historias de usuario y descripciones arquitectónicas utilizadas para entrenar y validar el modelo, incluyendo proyectos de diversas industrias, tecnologías y escalas.
- Aunque desarrollado específicamente para microservicios, el enfoque metodológico presentado en esta tesis, que combina revisiones sistemáticas, encuestas y experimentos controlados, puede ser adaptado para evaluar otros paradigmas arquitectónicos y tecnológicos.
- La investigación aporta nuevas métricas para evaluar atributos de calidad específicos de los microservicios, como granularidad, elasticidad y modularidad. Estas métricas enriquecen el repertorio existente y pueden ser usadas en estudios futuros.
- Mejorar las capacidades de automatización del modelo para que no solo evalúe dependencias y complejidad, sino que también sugiera directamente posibles refactorizaciones o cambios arquitectónicos.
- Investigar cómo MOAM puede integrarse directamente con pipelines de DevOps para ofrecer evaluaciones continuas y en tiempo real de la arquitectura durante el ciclo de vida del desarrollo.
- Diseñar interfaces gráficas para que equipos menos técnicos puedan interactuar con el modelo y beneficiarse de sus evaluaciones sin necesidad de expertos.
- Analizar cómo la implementación de MOAM afecta el ciclo de vida completo de aplicaciones basadas en microservicios, desde el diseño hasta el despliegue y la evolución.

5.3 AMENAZAS A LA VALIDEZ

5.3.1 VALIDEZ INTERNA

- MOAM depende de descripciones arquitectónicas y datos de entrenamiento que puedan contener sesgos derivados del dominio o del contexto del proyecto. Si los datos no son representativos o están incompletos, los resultados del modelo pueden ser inconsistentes o poco fiables.
- Las métricas utilizadas para evaluar dependencia y complejidad fueron seleccionadas a partir de revisiones sistemáticas y validaciones experimentales. Sin embargo, su aplicabilidad podría estar influenciada por interpretaciones subjetivas o suposiciones implícitas no aplicables a todos los escenarios.

5.3.2 VALIDEZ EXTERNA

- Aunque el modelo fue validado mediante estudios de caso, estos se centraron en un conjunto limitado de contextos y tecnologías. Esto podría restringir la aplicabilidad del modelo en proyectos con arquitecturas significativamente diferentes o en dominios industriales no evaluados.
- El rendimiento y la precisión del modelo no fueron probados exhaustivamente en arquitecturas extremadamente grandes o altamente distribuidas. Esto puede limitar su eficacia en escenarios donde la cantidad de microservicios y sus interdependencias aumenten significativamente.

BIBLIOGRAFÍA

[1] J. A. Valdivia, X. Limon, and K. Cortes-Verdin, "Quality attributes in patterns related to microservice architecture: A Systematic Literature Review," *Proceedings - 2019 7th International Conference in Software Engineering Research and Innovation, CONISOFT 2019*, pp. 181–190, 2019, doi: 10.1109/CONISOFT.2019.00034.

[2] M. Fowler, "martinFowler.com," Micriservices. Accessed: Nov. 24, 2020. [Online]. Available: https://martinfowler.com/articles/microservices.html

[3] L. Coppolino, S. D'Antonio, G. Mazzeo, and L. Romano, "A comparative analysis of emerging approaches for securing java software with Intel SGX," *Future Generation Computer Systems*, vol. 97, pp. 620–633, 2019, doi: 10.1016/j.future.2019.03.018.

[4] D. R. F. Apolinário and B. B. N. de França, "A method for monitoring the coupling evolution of microservice-based architectures," *Journal of the Brazilian Computer Society*, vol. 27, no. 1, Dec. 2021, doi: 10.1186/s13173-021-00120-y.

[5] J. McZara, S. Kafle, and D. Shin, "Modeling and Analysis of Dependencies between Microservices in DevSecOps," in *2020 IEEE International Conference on Smart Cloud (SmartCloud)*, IEEE, Nov. 2020, pp. 140–147. doi: 10.1109/SmartCloud49737.2020.00034.

[6] J. Ghofrani and D. Lübke, "Challenges of microservices architecture: A survey on the state of the practice," *CEUR Workshop Proc*, vol. 2072, no. October, pp. 1–8, 2018.

[7] R. Heinrich *et al.*, "Performance engineering for microservices: Research challenges & directions," *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, no. February 2018, pp. 223–226, 2017, doi: 10.1145/3053600.3053653.

[8] M. Fowler, "Who needs an architect?," *IEEE Softw*, vol. 20, no. 5, pp. 11–13, 2003, doi: 10.1109/MS.2003.1231144.

[9] M. D. M.-D. Cojocaru, A. Oprescu, and A. Uta, "Attributes assessing the quality of microservices automatically decomposed from monolithic applications," *Proceedings - 2019 18th International Symposium on Parallel and Distributed Computing, ISPDC 2019*, no. 1, pp. 84–93, 2019, doi: 10.1109/ISPDC.2019.00021.

[10] A. H. Osses Felipe, Márquez Gastón, "An Exploratory Study of Academic Architectural Tactics and Patterns in Microservices: A systematic literature review," 2019, [Online]. Available: https://www.researchgate.net/publication/331022065_An_Exploratory_Study_of_Academic_Architectural_Tactics_and_Patterns_in_Microservices_A_systematic_literature_review

[11] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández, "Patterns Related to Microservice Architecture: a Multivocal Literature Review," *Programming and Computer Software*, vol. 46, no. 8, pp. 594–608, Dec. 2020, doi: 10.1134/S0361768820080253.

[12] Inc. Red Hat, "Microservicios," ¿Qué son los microservicios? Accessed: Jul. 02, 2021. [Online]. Available: https://www.redhat.com/es/topics/microservices/what-are-microservices

[13] MEDIACLOUD, "¿Qué son los microservicios? Definición, características y retos." Accessed: Jul. 02, 2021. [Online]. Available: https://blog.mdcloud.es/que-son-los-microservicios-definicion-caracteristicas-y-retos/

[14] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, vol. 9783642290. 2012. doi: 10.1007/978-3-642-29044-2.

[15] F. H. Vera-Rivera, C. M. Gaona Cuevas, and H. Astudillo, "Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación," *Revista Ibérica de Sistemas e Tecnologias de Informação*, vol. E23, no. January 2019, pp. 107–120, 2019.

[16] M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," *Journal of Systems and Software*, vol. 170, 2020, doi: 10.1016/j.jss.2020.110798.

[17] C. Richardson, "Microservice Architecture." [Online]. Available: https://microservices.io/patterns/microservices.html

[18] M. Fowler, "martinFowler.com," Micriservices. Accessed: Nov. 25, 2020. [Online]. Available: https://martinfowler.com/articles/microservices.html

[19] V. Velepucha, P. Flores, and J. Torres, "MOMMIV : Modelo para descomposición de una arqui- tectura monolítica hacia una arquitectura de microservi- cios bajo el Principio de Ocultación de Información," pp. 1000–1010, 2019.

[20] C. W. B. R. De la Torre, *Microservicios de .NET: Arquitectura para aplicaciones .NET en contenedo*, V5.0: actu. Microsoft Corporation, 2021.

[21] A. F. and C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices," *IEEE Softw*, pp. 63–72, 2018.

[22] Y. Wang, H. Kadiyala, and J. Rubin, *Promises and challenges of microservices: an exploratory study*, vol. 26, no. 4. Empirical Software Engineering, 2021. doi: 10.1007/s10664-020-09910-y.

[23] ISO 25000, "ISO 25OOO Calidad de software y datos." [Online]. Available: https://iso25000.com/index.php/normas-iso-25000

[24] S. Jeon, M. Han, E. Lee, and K. Lee, "Quality attribute driven agile development," *Proceedings - 2011 9th International Conference on Software Engineering Research, Management and Applications, SERA 2011*, no. August 2011, pp. 203–210, 2011, doi: 10.1109/SERA.2011.24.

[25] T. Schirgi and E. Brenner, "Quality Assurance for Microservice Architectures," *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, vol. 2021-Augus, pp. 76–80, 2021, doi: 10.1109/ICSESS52187.2021.9522227.

[26] G. Arcos-Medina and D. Mauricio, *Aspects of software quality applied to the process of agile software development: a systematic literature review*, vol. 10, no. 5. Springer India, 2019. doi: 10.1007/s13198-019-00840-7.

[27] V. C. Tapia and C. M. Gaona, "Research Opportunities in Microservices Quality Assessment: A Systematic Literature Review," *Journal of Advances in Information Technology*, vol. 14, no. 5, pp. 991–1002, 2023, doi: 10.12720/jait.14.5.991-1002.

[28] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Softw Pract Exp*, vol. 50, no. 9, pp. 1651–1681, 2020, doi: 10.1002/spe.2869.

[29] J. W. Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, 4Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, "Performance Engineering for Microservices: Research Challenges and Directions," in *Third Workshop on Challenges in Performance Methods for Software Development (WOSP-C 2017) at 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*, 2017. doi: 10.1145/3053600.3053653.

[30] M. Cardarelli, A. Di Salle, L. Iovino, I. Malavolta, P. Di Francesco, and P. Lago, "An extensible data-driven approach for evaluating the quality of microservice architectures," in *34th Annual ACM Symposium on Applied Computing, SAC 2019*, University of L'Aquila, L'Aquila, Italy: Association for Computing Machinery, 2019, pp. 1225–1234. doi: 10.1145/3297280.3297400.

[31] A. Baškarada, S., Nguyen, V., Koronios, "Architecting Microservices: Practical Opportunities and Challenges," *Journal of Computer Information Systems*, vol. 60, no. 5, 2, pp. 428–436, doi: 10.1080/08874417.2018.1520056.

[32] G. Márquez and H. Astudillo, "Helping novice architects to manage architectural technical debt in microservices architecture," in *13th Ibero-American Conference on Software Engineering and Knowledge Engineering, JIISIC 2018*, Toeska research group, Universidad Técnica Federico Santa María Valparaíso, Chile: Escuela Superior Politecnica de Chimborazo, 2018, pp. 97–108.

[33] A. A. Durán Chinchilla, Claudia Marcela; García Quintero, Carmen Liceth G; Rosado Gómez, "APLICACIÓN DEL PROCESAMIENTO DELLENGUAJE NATURAL COMO TÉCNICADEANÁLISIS EN LAPRODUCCIÓN TEXTUAL,CASO ESTUDIANTES DE INGENIERÍADESISTEMAS UFPS," *Revista Boletín Redipe*, vol. 11, 2022, doi: 10.36260/rbr.v11i1.1656.

[34] T. Bell and T. Olavsrud, "What is natural language processing? The business benefits of NLP explained | CIO," *Cio*, pp. 1–5, 2020.

[35] A. R. Martinez, "Natural language processing," *WIREs Computational Statistics*, vol. 2, no. 3, pp. 352–357, May 2010, doi: 10.1002/wics.76.

[36] J. I. Bagnato, "Aprende Machine Learning." [Online]. Available: https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/

[37] C. Stryker and J. Holdsworth, "IBM, Natural Language Processing (NLP)," Procesamiento del lenguaje natural. Accessed: Dec. 04, 2024. [Online]. Available: https://www.ibm.com/es-es/topics/natural-language-processing.

[38] J. C. Blandón Andrade, "Aplicaciones del Procesamiento de Lenguaje Natural," *Entre Ciencia e Ingeniería*, vol. 16, no. 2, pp. 440–454, 2022, doi: https://doi.org/10.31908/19098367.2847.

[39] R. Y. B. Murdoch, James; Singh, Chandan; Kumbier, Karl; Abbasi-Asl, "Interpretable machine learning: definitions, methods, and applications," 2019. doi: 10.1073/pnas.1900654116.

[40] G. Spanakis, G. Weiss, B. Boh, L. Lemmens, and A. Roefs, "Machine learning techniques in eating behavior e-coaching: Balancing between generalization and personalization," *Pers Ubiquitous Comput*, vol. 21, no. 4, pp. 645–659, 2017, doi: 10.1007/s00779-017-1022-4.

[41] M. J. Issam El Naqa; Murphy, *What Is Machine Learning?* Springer, Cham, 2015. doi: https://doi.org/10.1007/978-3-319-18305-3_1.

[42] R. V. Dirk P. Kroese; Zdravko I. Botev;Thomas Taimre, *Data Science and Machine Learning Mathematical and Statistical Methods*. 2024.

[43] Revista robots, "Qué es el Machine Learning y el Aprendizaje Automático." [Online]. Available: https://revistaderobots.com/inteligencia-artificial/que-es-el-machine-learning-y-el-aprendizaje-automatico/

[44] J. I. Bagnato, "Aprende Machine Learning." [Online]. Available: https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/

[45] F. H. Vera-Rivera, E. G. Puerto-Cuadros, H. Astudillo, and C. M. Gaona-Cuevas, "Microservices Backlog - A Model of Granularity Specification and Microservice Identification," in *17th International Conference on Services Computing, SCC 2020, held as part of the Services Conference Federation, SCF 2020*, vol. 12409 LNCS, W. Q., X. Y., S. S., and Z. L.-J., Eds., Universidad Francisco de Paula Santander, San José de Cúcuta, Colombia: Springer Science and Business Media Deutschland GmbH, 2020, pp. 85–102. doi: 10.1007/978-3-030-59592-0_6.

[46] B. Yang, H. Guo, and H. Liu, "Evaluation and assessment of machine learning based user story grouping: A framework and empirical studies,"

Sci Comput Program, vol. 227, p. 102943, Apr. 2023, doi: 10.1016/j.scico.2023.102943.

[47] A. Engel, Thomas; Langermeier, Melanie; Bauer, Bernhard; Hofmann, *Information Systems in the Big Data Era*, vol. 317. in Lecture Notes in Business Information Processing, vol. 317. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-92901-9.

[48] M. Cardarelli, L. Iovino, P. Di Francesco, A. Di Salle, I. Malavolta, and P. Lago, "An extensible data-driven approach for evaluating the quality of microservice architectures," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, New York, NY, USA: ACM, Apr. 2019, pp. 1225–1234. doi: 10.1145/3297280.3297400.

[49] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[50] V. C. Tapia and C. M. Gaona, "Automation of Software Development Stages with the OpenAI API," 2024, doi: 10.32604/csse.2024.056979.

[51] B. Kitchenham, "Procedures for Performing Systematic Reviews," *Keele University Technical Report TR/SE-0401*, 2004.

[52] S. Li *et al.*, "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Inf Softw Technol*, vol. 131, p. 106449, 2021, doi: https://doi.org/10.1016/j.infsof.2020.106449.

[53] M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," *Journal of Systems and Software*, vol. 170, p. 110798, 2020, doi: https://doi.org/10.1016/j.jss.2020.110798.

[54] P. H. M. M. and M. I. C. M. Filho, E. Pimentel, W. Pereira, "Self-Adaptive Microservice-based Systems - Landscape and Research Opportunities," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021. doi: 10.1109/SEAMS51251.2021.00030.

[55] F. H. Vera-Rivera, C. Gaona, and H. Astudillo, "Defining and measuring microservice granularity—a literature overview," *PeerJ Comput Sci*, vol. 7, p. e695, Sep. 2021, doi: 10.7717/peerj-cs.695.

[56] A. Yamuç and U. M. Sürme, "A service-oriented architecture to microservice architecture migration strategy for satellite ground software systems ," in *7th Turkish National Software Architecture Conference, UYMK 2018*, T. B. and D. A.H., Eds., Türk Uzay ve Havacılık Sanayii, Turkish Aerospace Industries, Ankara, Turkey: CEUR-WS, 2018, pp. 8–19. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060224472&partnerID=40&md5=7cc7338e3792390df121a640c6c03e94

[57] S. A. K. A. K. Ghayyur, A. Razzaq, S. Ullah, and S. Ahmed, "Matrix clustering based migration of system application to microservices architecture," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 1, pp. 284–296, 2018, doi: 10.14569/IJACSA.2018.090139.

[58] I. Ranawaka *et al.*, "Custos: Security Middleware for Science Gateways," in *Practice and Experience in Advanced Research Computing*, in PEARC '20. New

York, NY, USA: Association for Computing Machinery, 2020, pp. 278–284. doi: 10.1145/3311790.3396635.

[59] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019, doi: https://doi.org/10.1016/j.jss.2019.01.001.

[60] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanović, and A. van Hoorn, "Microservices: A Performance Tester's Dream or Nightmare?," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, in ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 138–149. doi: 10.1145/3358960.3379124.

[61] F. Klinaku, D. Bilgery, and S. Becker, "The Applicability of Palladio for Assessing the Quality of Cloud-Based Microservice Architectures," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, in ECSA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 34–37. doi: 10.1145/3344948.3344961.

[62] M. Cardarelli, A. Di Salle, L. Iovino, I. Malavolta, P. Di Francesco, and P. Lago, "An extensible data-driven approach for evaluating the quality of microservice architectures," in *34th Annual ACM Symposium on Applied Computing, SAC 2019*, University of L'Aquila, L'Aquila, Italy: Association for Computing Machinery, 2019, pp. 1225–1234. doi: 10.1145/3297280.3297400.

[63] F. H. Vera-Rivera, C. M. Gaona Cuevas, and H. Astudillo, "Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de

investigación," *Revista Ibérica de Sistemas e Tecnologias de Informação*, vol. E23, no. January 2019, pp. 107–120, 2019.

[64] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, pp. 187–195. doi: 10.1109/ICSA-C.2019.00041.

[65] M. Stocker, O. Zimmermann, U. Zdun, D. Lübke, and C. Pautasso, "Interface quality patterns - Communicating and improving the quality of microservices APIs," in *23rd European Conference on Pattern Languages of Programs, EuroPLoP 2018*, University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland: Association for Computing Machinery, 2018. doi: 10.1145/3282308.3282319.

[66] O. Zimmermann, M. Stocker, D. Lübke, and U. Zdun, "Interface representation patterns - Crafting and consuming message-based remote APIs," in *22nd European Conference on Pattern Languages of Programs, EuroPLoP 2017*, University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland: Association for Computing Machinery, 2017. doi: 10.1145/3147704.3147734.

[67] G. Márquez and H. Astudillo, "Identifying availability tactics to support security architectural design of microservice-based systems," in *13th European Conference on Software Architecture, ECSA 2019*, D. L., K. A., M. R., M. E.M.N., Q. C., S. R., S. P., T. C., and W. D., Eds., Universidad Técnica Federico Santa María, Valparaíso, Chile: Association for Computing Machinery, 2019, pp. 123–131. doi: 10.1145/3344948.3344996.

[68] M. Gao, M. Chen, A. Liu, W. H. Ip, and K. L. Yung, "Optimization of Microservice Composition Based on Artificial Immune Algorithm Considering Fuzziness and User Preference," *IEEE Access*, vol. 8, pp. 26385–26404, 2020, doi: 10.1109/ACCESS.2020.2971379.

[69] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Limiting Technical Debt with Maintainability Assurance: An Industry Survey on Used Techniques and Differences with Service- and Microservice-Based Systems," in *Proceedings of the 2018 International Conference on Technical Debt*, in TechDebt '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 125–133. doi: 10.1145/3194164.3194166.

[70] J. Bogner, A. Zimmermann, and S. Wagner, "Towards an Evolvability Assurance Method for Service-Based Systems," 2020, *Springer Science and Business Media Deutschland GmbH, University of Applied Sciences Reutlingen, Reutlingen, Germany*. doi: 10.1007/978-3-030-63161-1_10.

[71] J. Bogner, S. Wagner, and A. Zimmermann, "Automatically measuring the maintainability of service- and microservice-based systems - a literature review," in *27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, IWSM Mensura 2017*, Reutlingen University of Applied Sciences, Germany: Association for Computing Machinery, 2017, pp. 107–115. doi: 10.1145/3143434.3143443.

[72] M. Cojocaru, A. Uta, and A.-M. A. M. Oprescu, "MicroValid: A validation framework for automatically decomposed microservices," *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2019-Decem, pp. 78–86, 2019, doi: 10.1109/CloudCom.2019.00023.

[73] J. Bogner, S. Wagner, and A. Zimmermann, "On the impact of service-oriented patterns on software evolvability: A controlled experiment and metric-based analysis," *PeerJ Comput Sci*, vol. 2019, no. 8, 2019, doi: 10.7717/peerj-cs.213.

[74] J. Bogner, S. Wagner, and A. Zimmermann, "Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems," *SICS SOFTWARE-INTENSIVE CYBER-PHYSICAL SYSTEMS*, vol. 34, no. 2–3, SI, pp. 141–149, Jun. 2019, doi: 10.1007/s00450-019-00402-z.

[75] C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, "Exploiting load testing and profiling for Performance Antipattern Detection," *Inf Softw Technol*, vol. 95, pp. 329–345, 2018, doi: 10.1016/j.infsof.2017.11.016.

[76] S. Li *et al.*, "A dataflow-driven approach to identifying microservices from monolithic applications," *Journal of Systems and Software*, vol. 157, p. 110380, 2019, doi: https://doi.org/10.1016/j.jss.2019.07.008.

[77] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated microservice identification in legacy systems with functional and non-functional metrics," in *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 135–145. doi: 10.1109/ICSA47634.2020.00021.

[78] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to microservice one based on a semi-automatic approach," in *17th IEEE International Conference on Software Architecture, ICSA 2020*, LIRMM,

University of Montpellier, CNRS, Montpellier, France: Institute of Electrical and Electronics Engineers Inc., 2020, pp. 157–168. doi: 10.1109/ICSA47634.2020.00023.

[79] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering," in *25th IEEE International Conference on Web Services, ICWS 2018*, Xi'An Jiaotong University, China: Institute of Electrical and Electronics Engineers Inc., 2018, pp. 211–218. doi: 10.1109/ICWS.2018.00034.

[80] M. Abdellatif *et al.*, *State of the practice in service identification for SOA migration in industry*, vol. 11236 LNCS. Département d'informatique, Université du Québec à Montréal, Montréal, Canada: Springer Verlag, 2018, pp. 634–650. doi: 10.1007/978-3-030-03596-9_46.

[81] C. Schröer, F. Kruse, and J. Marx Gómez, "A Qualitative Literature Review on Microservices Identification Approaches," *Communications in Computer and Information Science*, vol. 1310, pp. 151–168, 2020, doi: 10.1007/978-3-030-64846-6_9.

[82] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, and R. O. Mahamane, "Re-architecting OO software into microservices: A quality-centred approach," 2018, *Springer Verlag, LIRMM, CNRS, University of Montpellier, Montpellier, France*. doi: 10.1007/978-3-319-99819-0_5.

[83] O. Al-Debagy and P. Martinek, "A metrics framework for evaluating microservices architecture designs," *Journal of Web Engineering*, vol. 19, no. 3–4, pp. 341–369, 2020, doi: 10.13052/jwe1540-9589.19341.

[84] MARTÍNEZ CRISTIAN, "EVALUACIÓN DE UNA HERRAMIENTA DE PRUEBAS END-TOEND PARA MICROSERVICIOS IMPLEMENTADOS

EN JAVA Y NODE.JS," UNIVERSIDAD DE COSTA RICA, 2020. [Online]. Available: http://www.kerwa.ucr.ac.cr/bitstream/handle/10669/82800/TFIA_CristianMartinezHernandez SEP.pdf?sequence=3&isAllowed=y

[85] S. S. Scrocca M., Tommasini R., Margara A., Valle E.D., "The Kaiju project: Enabling event-driven observability," in *DEBS '20: The 14th ACM International Conference on Distributed and Event-based Systems*, Canada: ACM, 2020. doi: 10.1145/3401025.3401740.

[86] Y. Gan *et al.*, "Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2019, pp. 19–33. doi: 10.1145/3297858.3304004.

[87] F. Pina, J. Correia, R. Filipe, F. Araujo, J. Cardroom, and J. Pina, F., Correia, J., Filipe, R., Araujo, F., Cardroom, "Nonintrusive monitoring of microservice-based systems," in *NCA 2018 - 2018 IEEE 17th International Symposium on Network Computing and Applications*, Cambridge: IEEE, 2018. doi: 10.1109/NCA.2018.8548311.

[88] P. Bacchiega, I. Pigazzini, and F. A. Fontana, "Microservices smell detection through dynamic analysis," *Proceedings - 48th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2022*, pp. 290–293, 2022, doi: 10.1109/SEAA56994.2022.00052.

[89] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of Microservice Architectures: A Metric and Tool-Based Approach," in *International Conference on Advanced Information Systems Engineering*, 2018, pp. 74–89. doi: 10.1007/978-3-319-92901-9_8.

[90] S. Pulnil and T. Senivongse, "A Microservices Quality Model Based on Microservices Anti-patterns," in *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Thailand: IEEE, Jun. 2022, pp. 1–6. doi: 10.1109/JCSSE54890.2022.9836297.

[91] M. Schreiber, "Prevant (Preview servant): Composing microservices into reviewable and testable applications," *OpenAccess Series in Informatics*, vol. 78, no. 5, pp. 1–5, 2020, doi: 10.4230/OASIcs.Microservices.2017-2019.5.

[92] J. P. Sotomayor, S. C. Allala, P. Alt, J. Phillips, T. M. King, and P. J. Clarke, "Comparison of runtime testing tools for microservices," *Proceedings - International Computer Software and Applications Conference*, vol. 2, pp. 356–361, 2019, doi: 10.1109/COMPSAC.2019.10232.

[93] E. Tapia, Verónica; Gaona, Carlos; Marcalla, Alison; Aimacaña, "Evaluando Dependencia y Complejidad en el Diseño de Microservicios: Un Enfoque Integral," *Revista Ibérica de Sistemas e Tecnologias de Informação*, vol. E73, pp. 590–605, 2024.

[94] F. H. Vera Rivera, "Modelo inteligente de especificación de la granularidad de aplicaciones basadas en microservicios.," 2021. [Online]. Available: https://hdl.handle.net/10893/19567

[95] J. Luo, Shutian; Xu, Huanle; Lu, Chengzhi; Ye, Kejiang; Xu, Guoyao; Zhang, Liping; Ding, Yu; He and C. Xu, "Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis," in *SoCC '21: Proceedings of the ACM Symposium on Cloud Computing*, Association for Computing Machinery, 2021. doi: 10.1145/3472883.3487003.

[96] R. Capuano and H. Muccini, "A Systematic Literature Review on Migration to Microservices: A Quality Attributes perspective," *2022 IEEE*

19th International Conference on Software Architecture Companion, ICSA-C 2022, pp. 120–123, 2022, doi: 10.1109/ICSA-C54293.2022.00030.

[97] W. Hasselbring and G. Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 243–246. doi: 10.1109/ICSAW.2017.11.

[98] X. He, Z. Tu, M. Wagner, X. Xu, and Z. Wang, "Online Deployment Algorithms for Microservice Systems With Complex Dependencies," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1746–1763, Apr. 2023, doi: 10.1109/TCC.2022.3161684.

[99] R. Valverde, *Ontological Evaluation of the Requirements Model in IS Re-engineering: Shifting From a Traditional To a Component-Based Paradigm in Information Systems Re-Engineering*. 2010.

[100] K. J. P. G. Perera and I. Perera, "TheArchitect: A Serverless-Microservices Based High-level Architecture Generation Tool," *Proceedings - 17th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2018*, pp. 204–210, 2018, doi: 10.1109/ICIS.2018.8466390.

[101] I. U. P. Gamage and I. Perera, "Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach," *MERCon 2021 - 7th International Multidisciplinary Moratuwa Engineering Research Conference, Proceedings*, pp. 699–704, 2021, doi: 10.1109/MERCon52712.2021.9525743.

[102] T. Cerny, M. S. H. Chy, A. S. Abdelfattah, J. Soldani, and J. Bogner, "On Maintainability and Microservice Dependencies: How Do Changes Propagate?," *International Conference on Cloud Computing and Services*

Science, CLOSER - Proceedings, no. Closer, pp. 277–286, 2024, doi: 10.5220/0012725200003711.

[103] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, Dec. 2021, doi: 10.1016/j.jss.2021.111061.

[104] Omar Al-Debagy; Péter Martinek, "Dependencies-based microservices decomposition method," *INTERNATIONAL JOURNAL OF COMPUTERS AND APPLICATIONS*, 2021, doi: 10.1080/1206212X.2021.1915444.

[105] C. Zhong, H. Zhang, C. Li, H. Huang, and D. Feitosa, "On measuring coupling between microservices," *Journal of Systems and Software*, vol. 200, 2023, doi: 10.1016/j.jss.2023.111670.

[106] N. Santos and A. Rito Silva, "A complexity metric for microservices architecture migration," in *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 169–178. doi: 10.1109/ICSA47634.2020.00024.

[107] F. H. Vera-Rivera, E. Puerto, H. Astudillo, and C. M. Gaona, "Microservices Backlog—A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories," *IEEE Access*, vol. 9, no. September, pp. 117178–117203, 2021, doi: 10.1109/ACCESS.2021.3106342.

[108] K. Reyes Burgos and J. Aquino Trujillo, *Investigación en las Ciencias del Diseño Aplicación en los Contextos de Computación y Tecnología*. 2023.

[109] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[110] E. E. This, C. C. Attribution, and I. License, "Domain -- Driven Design Reference," 2015.

[111] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9846 LNCS, no. September 2016, pp. 185–200, 2016, doi: 10.1007/978-3-319-44482-6_12.

[112] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10465 LNCS, no. September, pp. 19–33, 2017, doi: 10.1007/978-3-319-67262-5_2.

[113] Mybatis, "mybatis/jpetstore-6." Accessed: Nov. 30, 2024. [Online]. Available: https://github.com/mybatis/jpetstore-6

ANEXOS

ANEXO 1. Encuesta a desarrolladores de aplicaciones basadas en microservicios

Encuesta dirigida a desarrolladores de aplicaciones basadas en microservicios.

* Indica que la pregunta es obligatoria

1. Correo electrónico *

2. NOMBRE DE EMPRESA *

3. Tiempo de experiencia en el desarrollo de microservicios. *

   Marca solo un óvalo.

   - ◯ Un año o menos
   - ◯ Más de un año
   - ◯ Más de dos años *Ir a la pregunta 8*

4. ¿Cómo determina los microservicios de una aplicación? *

5. ¿Qué técnicas utiliza para el diseño de aplicaciones basadas en microservicios? *

   Marca solo un óvalo.

   - ◯ Diseño Dirigido por el Dominio (DDD)
   - ◯ Diagrama de Flujo de Datos (DFD)
   - ◯ Modelado de Procesos de Negocio (BPMN)
   - ◯ Modelo C4
   - ◯ Ninguna

6. ¿Es importante considerar parámetros de calidad en el diseño de microservicios? *

Marca solo un óvalo.

- ◯ Sí
- ◯ No

7. ¿Cuáles son los atributos de calidad de microservicios que se ven afectados por arquitecturas dependientes y complejas? *

Selecciona todas las opciones que correspondan.

- ☐ Rendimiento
- ☐ Escalabilidad
- ☐ Seguridad
- ☐ Mantenimiento
- ☐ Acoplamiento
- ☐ Granularidad
- ☐ Cohesión
- ☐ Ninguna

Sección sin título

8. Teniendo en cuenta que la dependencia es la relación entre las operaciones de microservicios, de las cuales se invoca al menos una. ¿Ha identificado problemas de dependencia al desarrollar Software utilizando arquitecturas de microservicios? *

Marca solo un óvalo.

- ◯ Sí
- ◯ No

9. Si su respuesta es Sí, describa cuáles *

10. Teniendo en cuenta que la complejidad es el estado de eventos o cosas que tienen múltiples interconexiones y una estructura altamente complicada. ¿Ha identificado problemas de complejidad al desarrollar software utilizando arquitecturas de microservicios? *

Marca solo un óvalo.

- ◯ Sí
- ◯ No

11. Si su respuesta es Sí, describa cuáles *

_____

_____

_____

_____

_____

12. ¿Qué métricas de calidad utilizan en su empresa para evaluar microservicios? *

Selecciona todas las opciones que correspondan.

- ☐ Rendimiento
- ☐ Escalabilidad
- ☐ Seguridad
- ☐ Mantenimiento
- ☐ Acoplamiento
- ☐ Granularidad
- ☐ Cohesión
- ☐ Ninguna

13. Según su criterio ¿cuáles son los desafíos que se presentan en la evaluación de la calidad de los microservicios? *

Selecciona todas las opciones que correspondan.

- [ ] Seguridad
- [ ] Rendimiento
- [ ] Granularidad
- [ ] Diseño
- [ ] Escalabilidad
- [ ] Monitoreo
- [ ] Ninguna

14. ¿Qué herramientas de modelado utiliza para el diseño de aplicaciones basadas en microservicios? *