MODELO INTELIGENTE DE ESPECIFICACIÓN DE LA GRANULARIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS.

FREDY HUMBERTO VERA RIVERA. MSc.
UNIVERSIDAD FRANCISCO DE PAULA SANTANDER
Código: 201700019
fredyhumbertovera@ufps.edu.co, fredy.vera@correounivalle.edu.co

DOCTORADO EN INGENIERIA CON ENFASIS EN CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
FACULTAD DE INGENIERÍA
UNIVERSIDAD DEL VALLE
CALI - COLOMBIA
2021

MODELO INTELIGENTE DE ESPECIFICACIÓN DE LA GRANULARIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS.

TESIS PARA OPTAR EL TÍTULO DE DOCTOR EN INGENIERÍA CON ENFASIS EN CIENCIAS DE LA COMPUTACIÓN

AUTOR
FREDY HUMBERTO VERA RIVERA. MSc.
GRUPO DE INVESTIGACIÓN EN INTELIGENCIA ARTIFICIAL - GIA
UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

DIRECTOR
CARLOS MAURICIO GAONA CUEVAS, Ph.D.
GRUPO DE ESTUDIOS DOCTORALES EN INFORMÁTICA - GEDI.
UNIVERSIDAD DEL VALLE

DOCTORADO EN INGENIERIA CON ENFASIS EN CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
FACULTAD DE INGENIERÍA
UNIVERSIDAD DEL VALLE
CALI - COLOMBIA
2021

A la memoria de mi madre.

Gladyz Maria Rivera Vera (1953 - 2019)
Q.E.P.D

Gracias a su entrega a nosotros y sus sacrificios pudimos llegar lejos, seguiremos siempre sus enseñanzas y legado.

DEDICATORIA

A Dios mi guía, por darme la oportunidad de cumplir este sueño, a quien ofrezco este trabajo, todas las dificultades y logros obtenidos a lo largo de mi formación doctoral.

A mi Esposa Maribel, mis hijos Diego Alejandro y Laura Fernanda fuentes de mi inspiración, por su gran apoyo, sacrificio y esfuerzos realizados para ayudarme a cumplir esta meta.

A mis Padres Humberto y Gladys, gracias a su sacrificio, a su trabajo, a su esfuerzo, a sus enseñanzas y apoyo incondicional.

A mi hermano José Luis, mis hermanas Diana Andrea y Luz Adriana, por estar a mi lado y de alguna manera siempre aportaron algo a lo largo de este proceso.

A mi director de tesis Doctoral, Mauricio Gaona, sus aportes, conocimiento y enseñanzas en el proceso de formación doctoral fueron fundamentales para alcanzar este logro.

A todos mis amigos, compañeros docentes de la Universidad, quienes siempre están ahí para ayudar y apoyar.

Fredy Humberto Vera Rivera

AGRADECIMIENTOS

Al concluir esta etapa de formación doctoral expreso mis mas sinceros agradecimientos primero a la Universidad Francisco de Paula Santander por darme la oportunidad de realizar mi comisión de estudios de doctorado. También a la Gobernación del Norte de Santander, al Ministerio de Ciencia y Tecnología (MINCIENCIAS) y su programa de formación de capital humano de alto nivel para el Norte de Santander, que permitió apoyo económico para realizar los estudios de doctorado y la pasantía de investigación internacional.

Mi gratitud también a la escuela de Ingeniería de Sistemas y Computación de la Universidad del Valle, en especial a mi director, Ph. D. Carlos Mauricio Gaona Cuevas y a todos los docentes del Doctorado en ingeniería con énfasis en ciencias de la computación, sus enseñanzas y aportes permitieron la realización de esta tesis.

También agradezco los asesores Hernán Astudillo, profesor de la Universidad Técnica Federico Santa María de Chile, por permitirme realizar con él la pasantía de investigación internacional, y al profesor Eduard Puerto de la Universidad Francisco de Paula Santander, gracias a sus aportes, recomendaciones y sugerencias para mejorar este trabajo.

Además, a los miembros y socios de la Fundación Foristom, por su apoyo, por permitirme realizar las pruebas y el caso de estudio para validar los resultados de esta tesis doctoral.

Por último, a cada una de las personas que de una u otra forma influyeron en la realización de esta tesis doctoral.

RESUMEN

Los microservicios son un enfoque arquitectónico y organizativo del desarrollo de software en el que las aplicaciones están compuestas por pequeños servicios independientes que se comunican a través de una interfaz de programación de aplicaciones (API) bien definida, muchas empresas utilizan los microservicios para estructurar sus sistemas, también la arquitectura de microservicios ha sido utilizada en otras áreas como la internet de las cosas (IoT), computación en el borde (edge computing), computación en la nube, desarrollo de vehículos autónomos, telecomunicaciones, sistemas de E-Salud, E-Learning, entre otros. Un gran desafío al diseñar este tipo de aplicaciones es encontrar una partición o granularidad adecuada de los microservicios, proceso que a la fecha se realiza y diseña de forma intuitiva, según la experiencia del arquitecto o del equipo de desarrollo. La definición del tamaño o granularidad de los microservicios es un tema de investigación abierto y de interés, no se han estandarizado patrones, métodos o modelos que permitan definir qué tan pequeño debe ser un microservicio. Las estrategias más utilizadas para estimar la granularidad de los microservicios son: el aprendizaje automático, la similitud semántica, la programación genética y la ingeniería de dominio. En este trabajo de investigación doctoral se propone un modelo inteligente para especificar y evaluar la granularidad de los microservicios que hacen parte de una aplicación; teniendo en cuenta algunas características como la complejidad cognitiva, el tiempo de desarrollo, el acoplamiento, la cohesión y su comunicación.

En el capitulo uno se presentan el marco teórico, se plantea el problema de investigación resuelto, junto con las preguntas de investigación que ayudan a resolverlo, también se presentan los objetivos y la metodologia de investigación, por medio de la cual se propone una nueva práctica, un modelo inteligente de especificación de la granularidad de los microservicios llamada “Microsevices Backlog”, también se presentan las fases y métodos de investigación que permitieron resolver las preguntas de investigación planteadas. El captiulo dos presenta el esatado del arte y los trabajos relacionados con el presente trabajo de investigación doctoral; también se identifican las métricas que se han utilizado para definir y evaluar la granularidad de los microservicios. En el capitulo 3 se caracteriza el proceso de desarrollo de aplicaciones basadas en microservicios, explicando su uso en un caso de estudio llamado “Sinplafut”. En el capitulo 4 se plantea la descripción del “Microservice Backlog”, se presenta la definición de cada uno de sus componentes, entre los cuales se encuentran: el componente parametrizador, el componente agrupador (un algoritmo genético y un algoritmo de agrupamiento semántico basado en aprendizaje automático no supervisado), el componente evaluador de métricas y el componente comparador de descomposiciones y de microservicios candidatos, también se presenta la formulación matemática de la granularidad de aplicaciones basadas en microservicios. El capitulo 5 presenta la evaluación de la práctica propuesta, se realizó de forma iterativa usando cuatro casos de estudio, dos ejemplos planteados en el estado del arte (Cargo Tracking and JPet-Store) y dos proyectos reales (Foristom Conferences y Sinplafut), se utilizó el Microservices Backlog para obtener y evaluar los microservicios candidatos de las cuatro aplicaciones. Se realizó un analisis comparativo contra métodos propuestos en el estado del arte y con el diseño basado en el dominio (DDD), el cual es le método más utilizado para definir los microservicios que van a ser parte de una aplicación. El Microservices Backlog obtuvo un bajo acoplamiento, alta cohesión, baja complejidad y reduce la comunicación entre los microservicios, esto comparado con las propuestas del estado del arte y con DDD. Finalmente en el capitulo 6 se presentan las conclusiones, contribuciones, limitaciones y productos obtenidos como resultado de esta tesis.

PALABRAS CLAVE: Sistemas orientados a servicios, arquitectura de software, granularidad de microservicios, descomposición de microservicios, algoritmos genéticos, métricas de software.

ABSTRACT

Microservices are an architectural and organizational approach to software development in which applications are composed of small independent services that communicate through a well-defined application programming interface (API). Many companies use microservices to structure their systems, and microservices architecture has also been used in other areas such as the Internet of Things (IoT), edge computing, cloud computing, autonomous vehicle development, telecommunications, E-Health systems, E-Learning, among others. A great challenge when designing this type of applications is to find an appropriate partition or granularity of the microservices, a process that to date is performed and designed in an intuitive way, according to the experience of the architect or the development team. The definition of the granularity of microservices is an open research topic. There are no standardized patterns, methods or models that allow defining how small a microservice should be. The most used strategies to estimate the granularity of microservices are machine learning, semantic similarity, genetic programming, and domain engineering. In this doctoral research work an intelligent model was proposed to specify and evaluate the granularity of the microservices that are part of an application, considering some characteristics such as cognitive complexity, development time, coupling, cohesion, and its communication.

In chapter one the theoretical foundations are presented, the research problem is solved: How to model and define the adequate size of a microservice, considering its properties, dependencies, cognitive complexity, coupling and cohesion? together with the research questions that help to solve it, the objectives and research methodology are also presented. Based on "Design Science Research", a new practice was proposed, an intelligent model of specification of the granularity of the microservices called "Microsevices Backlog", the phases and research methods that allowed to solve the research questions posed are also presented. Chapter two presents the state of the art and the related works to the present doctoral research thesis; the metrics that have been used to define and evaluate the granularity of microservices are also identified. In Chapter three, the process of developing of microservice-based applications is characterized, explaining its use in a case study called "Sinplafut". Chapter four describes the "Microservice Backlog", and presents the definition of each of its components, among which are: the parameterizing component, the grouping component (a genetic algorithm and a semantic grouping algorithm based on unsupervised machine learning), the metrics evaluation component and the decomposition and candidate microservices comparison component. It also presents the mathematical formulation of the granularity of applications based on microservices. The fifth chapter presents the evaluation of the proposed practice. It was done in an iterative way using four case studies, two examples raised in the state of the art (Cargo Tracking and JPet-Store) and two real projects (Foristom Conferences and Sinplafut), the Microservices Backlog was used to obtain and evaluate the candidate microservices of the four applications. A comparative analysis was made against methods proposed in the state of the art and with the domain-driven design (DDD), which is the most used method to define the microservices of an application. The Microservices Backlog obtained a low coupling, high cohesion, low complexity and reduces the communication among the microservices, this compared with the proposals of the state of the art and with DDD. Finally, in chapter six we present the conclusions, contributions, limitations, and products obtained as result of this thesis.

Keywords: Service-oriented systems engineering, Service computing, Software design, Software architecture, Micro-services granularity, Microservices decompositions, Genetic algorithms, Software metrics.

TABLA DE CONTENIDO

LISTADO DE FIGURAS

LISTADO DE TABLAS

INTRODUCCIÓN

El enfoque de las arquitecturas basadas en microservicios es relativamente nuevo, se basa en el desarrollo de aplicaciones como un conjunto de pequeños servicios independientes. Los microservicios son servicios pequeños y autónomos que funcionan en conjunto. Siguen el principio de la responsabilidad simple que dice "Reuna las cosas que cambian por la misma razón, y separe las cosas que cambian por diferentes razones" [1]. La clave para comprender los microservicios es su independencia, cada uno se desarrolla, prueba e implementa por separado, cada servicio se ejecuta en su propio proceso independiente de los demás. La única relación entre diferentes microservicios es el intercambio de datos a través de las interfaces que están exponiendo [2].

Los microservicios son una nueva tendencia en la arquitectura de software que está fuertemente influenciada por la computación distribuida; la idea principal es dividir una aplicación en pequeños servicios; cada servicio debe ser lo más independiente posible de los demás. Cada uno de ellos se ejecuta en procesos separados, en un contenedor independiente y aislado. Es posible asignar recursos computacionales a cada microservicio individualmente de acuerdo con sus requerimientos, al contrario de las aplicaciones monolíticas que asignan recursos a toda la aplicación [3].

Los microservicios se presentan como un enfoque de implementación de la Arquitectura Orientada a Servicios (SOA), tienen como objetivo mejorar las desventajas y los problemas presentados en SOA. La resiliencia, la escalabilidad, la entrega rápida de software y el uso de menos recursos son características esenciales que las aplicaciones actuales deben considerar. La arquitectura de los microservicios vino a cumplir esas expectativas [4], sin embargo, aún existen muchos desafíos, como la complejidad de tener que administrar pequeños sistemas distribuidos, la latencia de red y la falta de confiabilidad, tolerancia a fallas, coherencia e integración de datos, administración de transacciones distribuidas, capas de comunicación, equilibrio de carga, orquestación, monitoreo y seguridad [5].

En este trabajo de investigación doctoral se propone un modelo inteligente que permite especificar la arquitectura de aplicaciones basadas en microservicios, de tal forma que a partir del modelo se pueda definir la granularidad o tamaño adecuado de cada microservicio teniendo en cuenta algunas características como la complejidad cognitiva, el tiempo de desarrollo, el acoplamiento, la cohesión, la comunicación y dependencias entre ellos, todo en tiempo de diseño; de tal manera que el arquitecto o desarrollador pueda encontrar una estrategia optimizada para su implementación.

La definición de la granularidad de los microservicios es un tema abierto de investigación y de mucho interes tanto en la academia como en la industria. La granularidad de los microservicios se define principalmente, primero por su tamaño o dimensiones, es decir, el número de operaciones expuestas por el microservicio, junto con el número de microservicios que forman parte de toda la aplicación, y segundo por su complejidad y dependencias. El objetivo es tener un bajo acoplamiento, una baja complejidad y una alta cohesión entre los microservicios. Hassan y otros [6] afirmaron que un nivel de granularidad determina "el tamaño del servicio y el alcance de la funcionalidad que expone [7]". La adaptación de la granularidad implica la fusión o

descomposición de los microservicios, por lo que se pasa a un nivel de granularidad de grano más fino o más grueso. Homay et al. [8] afirmaron que "el problema para encontrar la granularidad de los servicios es identificar un límite correcto (tamaño) para cada servicio en el sistema. En otras palabras, cada servicio del sistema debe tener un propósito concreto, lo más desacoplado posible, y añadir valor al sistema. Un servicio tiene una granularidad correcta o buena si maximiza la modularidad del sistema y al mismo tiempo minimiza la complejidad. Modularidad en el sentido de flexibilidad, escalabilidad, mantenibilidad y rastreabilidad, mientras que complejidad en términos de dependencia, comunicación y procesamiento de datos".

En este trabajo se abordó el problema de la definición de la granularidad en el contexto del desarrollo de software ágil; partiendo de un conjunto de requisitos funcionales expresados como historias de usuario dentro de un "product backlog" (listado priorizado y estimado de las funcionalidades que una aplicación debe contener) Ver figura 1; se pretende definir un modelo que usando tecnicas de inteligencia artificial permita definir y evaluar el tamaño de cada microservicio y luego a partir del modelo el equipo de desarrollo pueda implementar, desplegar y monitorear la aplicación que se está desarrollando. Para que esto sea posible, es necesario identificar, descubrir y catalogar las funcionalidades de esos microservicios, verificando que cubran con los requisitos de la aplicación que se va a desarrollar. A partir de esta identificación de microservicios, se pretende representar los requerimientos funcionales de la aplicación en el "Microservices backlog", ver figura 2, un modelo inteligente donde se puede observar y evaluar la granularidad de los microservicios y también analizar cómo será implementada y estructurada la aplicación en términos de microservicios.

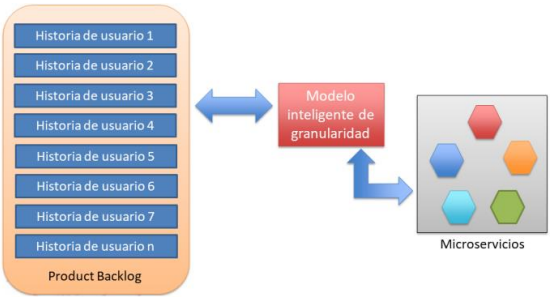


Figura 1. Desarrollo de aplicaciones basadas en microservicios – Modelo inteligente de especificación de la granularidad.

El modelo de especificación o “Microservices backlog”, figura 2, debe identificar y catalogar las funcionalidades de los microservicios preseleccionados, permitiendo así definir y evaluar la granularidad o tamaño de cada uno de ellos, teniendo en cuenta algunas características como son: complejidad cognitiva, acoplamiento, cohesión, comunicación y tiempo de desarrollo. Con este modelo, el arquitecto o desarrollador pueda ver cómo se construye su aplicación, qué dependencias tiene y evaluar el nivel de granularidad de cada microservicio.

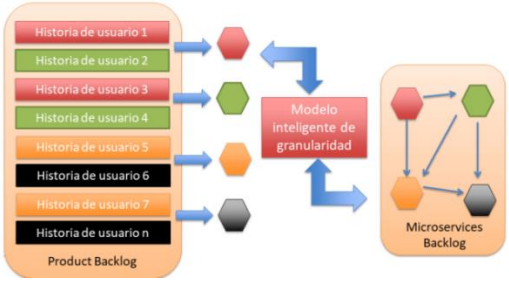


Figura 2. Microservices Backlog.

1. PROBLEMA

En el capítulo 1 se plantea el problema de investigación abordado en esta tesis doctoral, comenzando por los fundamentos teóricos, luego la definición del problema, las preguntas de investigación, los objetivos y la metodología llevada a cabo para culminar y proponer la nueva practica ágil denominada “Microservice Backlog” que representa y evalúa la granularidad de los microservicios de una aplicación.

1.1. MARCO TEÓRICO

A continuación, se detallan las áreas y fundamentos teóricos que sirvieron de base para el desarrollo de la presente investigación. El presenta trabajo de investigación se centró en las siguientes áreas de conocimiento:

1. La ingeniería del software: metodologías ágiles, prácticas ágiles, arquitecturas ágiles.
2. Los sistemas distribuidos: arquitectura orientada a servicios - SOA, microservicios,
3. La computación en la nube: infraestructuras, plataformas y software como servicio.
4. Diseño impulsado por el dominio (Domain Driven Design) – DDD
5. Inteligencia artificial: algoritmos Inteligentes, aprendizaje automático, aprendizaje profundo, clasificación.
6. Complejidad cognitiva.

En la figura 3 se presenta un cuadro sinóptico de los fundamentos que aportan las bases conceptuales, teóricas y metodológicas para resolver el problema de investigación planteado en la sección 1.2.

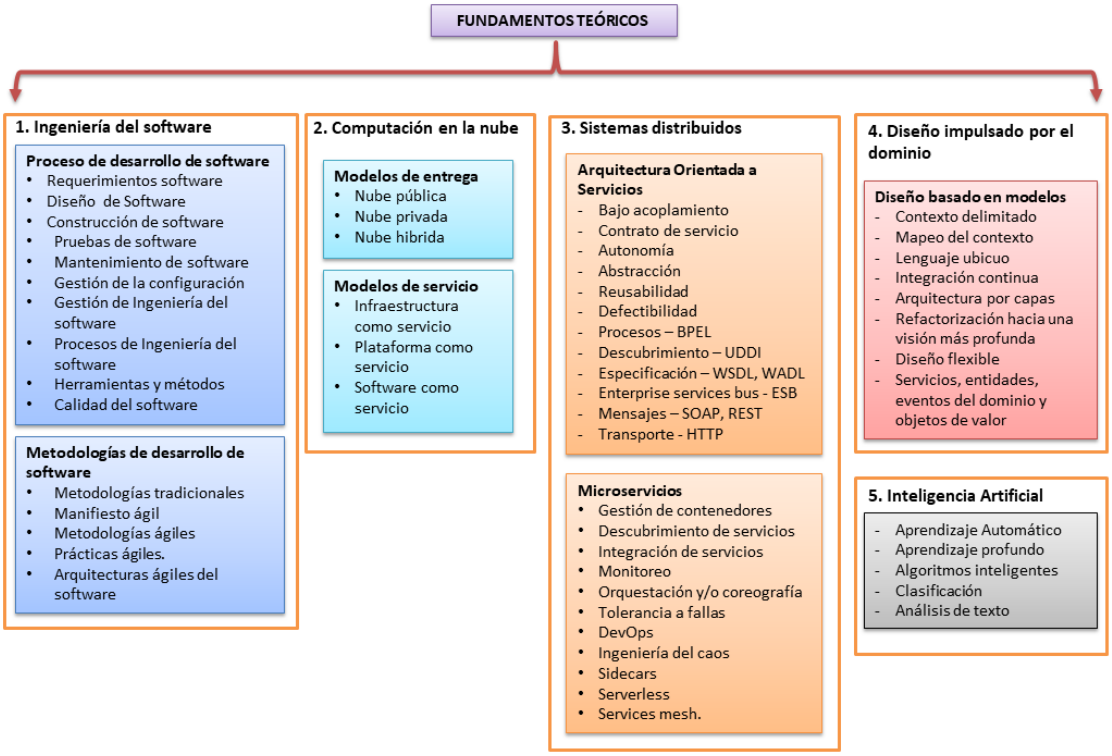


Figura 3. Fundamentos teóricos del trabajo de investigación doctoral.

1.1.1. INGENIERIA DE SOFTWARE

Es un área de las ciencias de la computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad. La ingeniería de software comprende los procesos técnicos del desarrollo del software, la gestión de proyectos de software y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de software. La ingeniería de software incluye procesos, métodos y herramientas que permiten elaborar a tiempo y con calidad sistemas complejos basados en computadoras. El proceso de software incorpora cinco actividades estructurales: comunicación, planeación, modelado, construcción y despliegue que son aplicables a todos los proyectos de software. La práctica de la ingeniería de software es una actividad para resolver problemas, que sigue un conjunto de principios fundamentales [9].

Un proceso de desarrollo de software es un conjunto de actividades y resultados asociados que producen un producto software. Estas actividades son llevadas a cabo por los ingenieros de software. Existen cuatro actividades fundamentales comunes a todos los procesos de desarrollo: (1) Especificación del software, (2) Desarrollo de software, (3) Validación del software y (4) Evolución del software [10].

El proceso de desarrollo de software es guiado por las siguientes áreas de conocimiento propuestas en el cuerpo del conocimiento de la Ingeniería del Software [11], estas áreas son:

- Requerimientos software
- Diseño de software
- Construcción de software
- Pruebas de software
- Mantenimiento de software
- Gestión de la configuración
- Gestión de Ingeniería del software
- Procesos de Ingeniería del software
- Herramientas y métodos
- Calidad del software

Practicas ágiles

La complejidad que implica el desarrollo de software ha sido abordada con el uso de las metodologías ágiles y las prácticas ágiles que nacieron a partir del manifiesto ágil y sus principios en contraste a las formas tradicionales [12]. Los beneficios más importantes que se obtienen con el uso y adopción de las prácticas ágiles son: el incremento de la productividad del equipo de desarrollo, se mejora la visibilidad del producto software, se adquiere la habilidad para manejar y controlar el cambio en los requerimientos y en las prioridades del proyecto y se hacen entregas más rápidas del proyecto [13]. Las prácticas o técnicas ágiles más utilizadas según el 14th estudio anual del estado del desarrollo ágil [13] son:

1. Reunión diaria de pie (Daily Standup),
2. Retrospectivas
3. Planificación de la iteración o del sprint (Iteration planning),
4. Revisión de la iteración o del sprint.
5. Iteraciones cortas

La planificación del sprint o iteración normalmente se realiza en un “product backlog”, el cual lista los requerimientos funcionales de la aplicación en forma de historias de usuario, junto con sus prioridades y estimados, el “producto backlog” lista las funcionalidades que debe desarrollar el equipo de desarrollo [14]; El equipo planifica la iteración, elabora la táctica que le permitirá conseguir el mejor resultado posible con el mínimo esfuerzo [15].

Una historia de usuario describe la funcionalidad que será de valor para un usuario o cliente del sistema software [16], [17]. La información que puede contener una historia de usuario según Kent Beck es: la fecha, el tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica (puntos y horas), una descripción, las notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios [18], ver tabla 1.

Tabla 1. Ejemplo de ficha para definir una historia de usuario.

Historia de Usuario	
Código o identificador:	Fecha:
Sprint o iteración:	Prioridad: (Alta, media, baja)
Estimación puntos: (estima el esfuerzo necesario para desarrollar la historia de usuario)	Estimación tiempo: (Tiempo en horas que implica desarrollar la historia de usuario)
Actores:	Fecha finalización:
Nombre historia:	
Descripción:	
Programador Responsable:	
Detalles: (Información requerida para entender la historia de usuario: escenario, entradas, procesos, salida, fotos, videos, audio, BPM, diseño interfaz de usuario, etc)	
Tareas: (Actividades necesarias para cumplir la historia de usuario)	
Restricciones:	
Criterios de aceptación:	
Definiciones de hecho (Definition of done - DoD):	
Dependencias:	

1.1.2. SISTEMAS DISTRIBUIDOS

Al dividir una aplicación monolítica en microservicios, cada uno ejecutándose en su propio proceso, desplegado en su propio contenedor, asignándole sus propios recursos computacionales como CPU, memoria y almacenamiento; integrándose y sincronizándose unos con otros para cumplir con las funcionalidades de la aplicación; de esta forma la aplicación monolítica se convierte en un sistema distribuido.

Según Tanenbaum y Steen un sistema distribuido se define como “una colección de computadoras independientes que aparece a sus usuarios como un único sistema coherente”, esta definición tiene varios aspectos importantes. El primero es que un sistema distribuido consta de componentes (es decir, computadoras) que son autónomos. Un segundo aspecto es que los usuarios (ya sean personas o programas) piensan que están tratando con un solo sistema. Esto significa que de una forma u otra los componentes autónomos necesitan colaborar. Cómo establecer esta colaboración se encuentra en el corazón del desarrollo de sistemas distribuidos [19].

Los microservicios son sistemas distribuidos altamente modulares, reutilizables a través de una API expuesta a través de la red. Esto implica que los microservicios heredan las ventajas y desventajas de los sistemas distribuidos y de los servicios web. Yarygina y Bagge platean que los microservicios son un enfoque de implementación de la Arquitectura orientada a servicios (SOA) y SOA es una subclase de sistema distribuido, como se puede ver en la figura 4 [20].

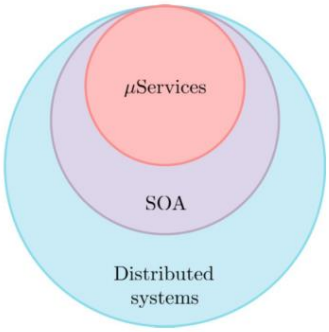


Figura 4. Microservicios en perspectiva, Sistemas distribuidos – SOA – Microservicios. Fuente: Yarygina y Bagge [20]

También en Zimmerman y otros [21], Pautasso y otros [22], indican que los microservicios comprenden un enfoque de implementación para SOA (al igual que Scrum es una, pero no la única forma de practicar el desarrollo ágil). Las características comunes incluyen la orientación comercial, la programación políglota en múltiples paradigmas e idiomas, el diseño tolerante a fallas; la descentralización y la automatización se enfatizan específicamente en el enfoque de implementación de microservicios.

Los microservicios tienen todas las complejidades asociadas de los sistemas distribuidos, y aunque se ha aprendido mucho sobre cómo administrar bien los sistemas distribuidos, todavía es difícil. También se tendrá que pensar diferente sobre cómo escalar sus sistemas y asegurarse de que sean resistentes. Temas como las transacciones distribuidas o el teorema CAP (Consistencia, Disponibilidad y partición de datos) son complejos de manejar [1]. Para Viktor Farcic [2], los aspectos clave de microservicios son los siguientes:

- Hacen una cosa o son responsables de una funcionalidad.
- Cada microservicio puede ser construido por cualquier conjunto de herramientas o lenguajes, ya que cada uno es independiente de otros.
- Son realmente ligeramente acoplados ya que cada microservicio está físicamente separado de los demás.
- Independencia relativa entre diferentes equipos desarrollando diferentes microservicios (suponiendo que las API que exponen se definen de antemano).
- Pruebas más sencillas automatizadas, entrega y despliegue continuo.

Según James Lewis y Martin Fowler, el término "microservicios" se discutió por primera vez en un taller de arquitectura de software en mayo de 2011, para denotar un enfoque arquitectónico común que los participantes del taller habían estado explorando. Anteriormente, destacados expertos de la industria ya habían estado explorando algunas de las mismas ideas, aunque bajo diferentes formas. Por ejemplo, Werner Vogels en Amazon describió su enfoque arquitectónico como "encapsulando los datos con la lógica de negocios que opera sobre los datos, con el único acceso a través de una interfaz de servicio pública", mientras que Adrian Cockcroft, luego en

Netflix, refirió a "una arquitectura débilmente acoplada orientada a servicios con contextos delimitados". Otros términos utilizados en la industria en ese momento para transmitir conceptos similares eran "SOA de grano fino" y "SOA bien hecho" [23].

Lewis y otros [23] presentan la evolución de los microservicios desde una perspectiva tecnológica (Ver figura 5) y arquitectural (Ver figura 6).

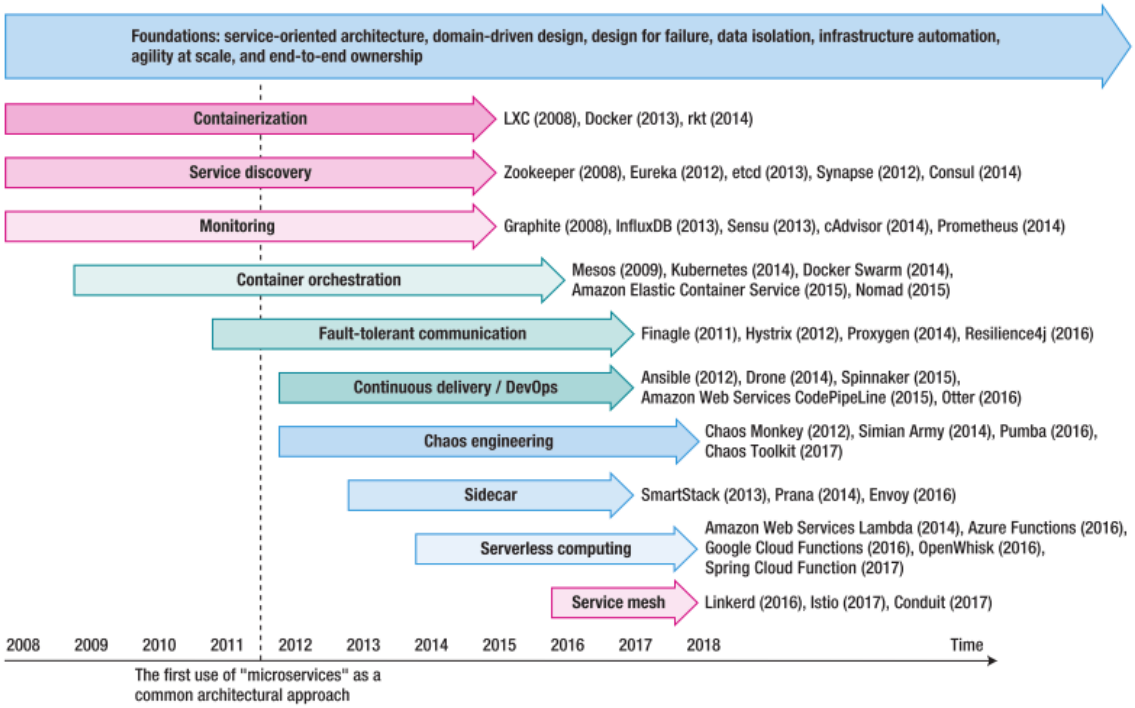


Figura 5. Línea del tiempo de evolución tecnológica de los microservicios. Fuente Lewis y otros [23].

En la figura 6 se presentan las generaciones de las arquitecturas de los microservicios y la forma como han venido evolucionando.

1.1.3. DISEÑO DIRIGIDO POR EL DOMINIO (DDD)

Los modelos representan algunos aspectos de la realidad o una idea de interés. Un modelo es una simplificación. Es una representación de la realidad que abstrae los aspectos más relevantes para resolver el problema en cuestión e ignora detalles extraños. Cada programa software es relacionado a una actividad de interés de su usuario. Esa área de interés a la cual el usuario aplica el programa es el dominio del software. Un modelo de dominio no es un diagrama particular; es la idea que el diagrama está destinado a transmitir. No es solo el conocimiento en la cabeza de un experto de dominio; es una abstracción rigurosamente organizada y selectiva del conocimiento [24].

“Domain-Driven Design” - DDD es un enfoque para el desarrollo de software complejo en el que:

- Se centra en el dominio principal.
- Se exploran modelos en una colaboración creativa de profesionales de dominio y profesionales del software.
- Se habla un lenguaje ubicuo dentro de un contexto explícitamente limitado.

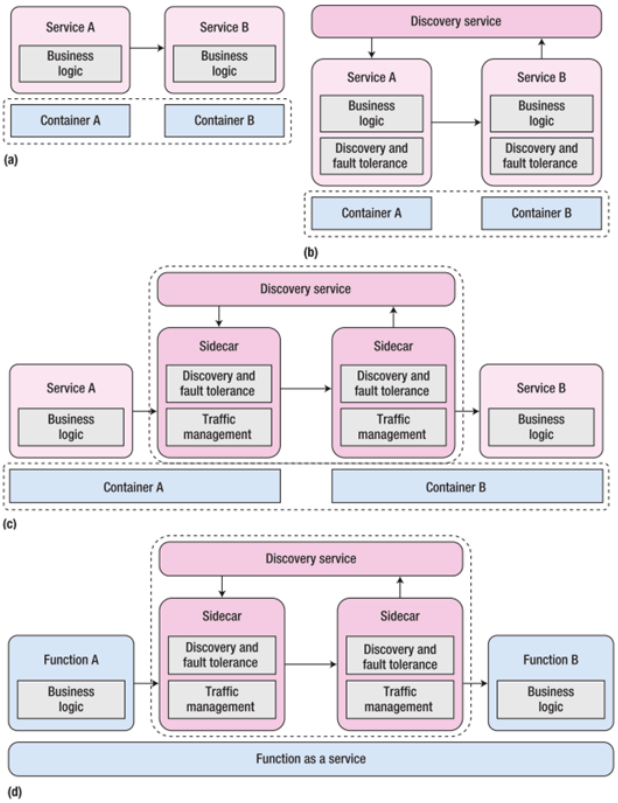


Figura 6. Generaciones de la arquitectura de los microservicios. (a) Orquestación de contenedores, (b) Descubrimiento de servicios y tolerancia a fallas, (c) Sidecar y service mesh, (d) Serverless y funciones como servicio. Fuente: Lewis y otros [23]

El enfoque de desarrollo de software llamado DDD existe para ayudar a lograr más fácilmente el diseño de un modelo de software de alta calidad. Cuando se implementa correctamente, DDD ayuda a llegar al punto en que el diseño es exactamente cómo funciona el software. DDD ofrece herramientas de modelaje estratégicas y tácticas para diseñar software de alta calidad que cubren los objetivos esenciales del negocio. DDD se trata de discusión, escucha, comprensión, descubrimiento y valor comercial, todo en un esfuerzo por centralizar el conocimiento. Si es capaz de comprender el negocio en el que trabaja su empresa, puede como mínimo participar en el proceso de descubrimiento del modelo de software para producir un lenguaje ubicuo [25].

DDD ha sido recomendado y utilizado para definir los límites de los microservicios, determinando las funcionalidades que cada microservicio debe manejar.

1.1.4. INTELIGENCIA ARTIFICIAL

La inteligencia artificial (IA) es una disciplina académica relacionada con la teoría de la computación cuyo objetivo es emular algunas de las facultades intelectuales humanas en sistemas artificiales. Con inteligencia humana nos referimos típicamente a procesos de percepción sensorial (visión, audición, etc) y a sus consiguientes procesos de reconocimiento de patrones, por lo que las aplicaciones más habituales de IA son el tratamiento de datos y la identificación de sistemas. El diseño de un sistema de IA normalmente requiere la utilización de herramientas de disciplinas muy diferentes como el cálculo numérico, la estadística, la informática, el procesado de señales, el control automático, la robótica o la neurociencia [26].

Se puede definir la IA como una ciencia que tiene por objetivo el diseño y construcción de máquinas capaces de imitar el comportamiento inteligente de las personas. Una rama especializada de la informática que investiga y produce razonamiento por medio de máquinas automáticas y que pretende fabricar artefactos dotados de la capacidad de pensar [27].

La IA ha sido utilizada en muchas áreas del conocimiento para resolver diversos tipos de problemas, desde la detección y prevención de enfermedades, en la clasificación de alimentos y la predicción de comportamientos económicos, clasificación de clientes e identificación de gustos y tendencias del mercado, el uso de algoritmos inteligentes ha permitido estos avances, el aprendizaje automático (Machine learning) ha otorgado a estos sistemas informáticos características completamente nuevas. Los algoritmos inteligentes son los métodos utilizados para extraer patrones de datos con el fin de otorgar a las computadoras los poderes para predecir y hacer inferencias [28]. En la figura 7 se puede apreciar los principales algoritmos y sus usos más comunes del aprendizaje automático. Estos se dividen en tres categorías: Aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado.

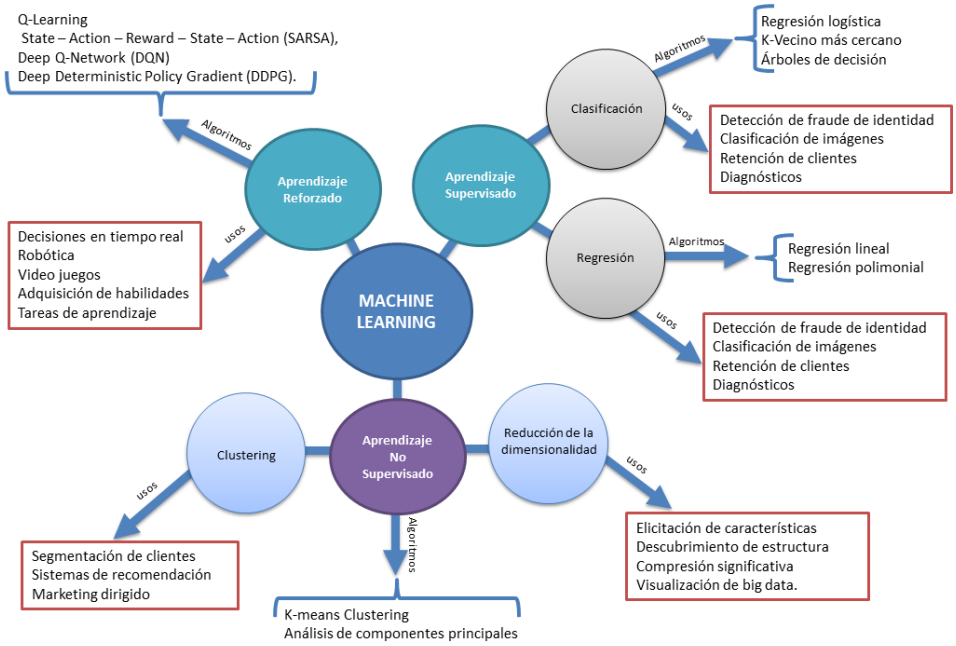


Figura 7. Principales algoritmos del aprendizaje automático (Machine learning). Fuente: Chugh [28]

El Aprendizaje profundo (Deep Learning) es un reconocimiento de patrones y modelos estadísticos muy elaborados que utilizan los modelos anteriores ajustados a enormes cantidades de datos. Las técnicas / algoritmos más comunes son las redes neuronales convolucionales (CNN), las redes neuronales recurrentes (RNN) y el aprendizaje por refuerzo (RL) [29].

La inteligencia artificial en especial el aprendizaje automático ha sido utilizado para el reconocimiento y clasificación de texto en categorías definidas, Fabrizio (2002), quien analiza los principales enfoques para la categorización de texto que se encuentran dentro del paradigma de aprendizaje automático. Discute en detalle los problemas relacionados con tres problemas diferentes, a saber, la representación de documentos, la construcción de clasificadores y la evaluación de clasificadores [30]. Por lo tanto, en este trabajo de investigación doctoral se pretende por medio de estas técnicas clasificar la información contenida en las historias de usuario para determinar cuáles de ellas formaran parte de los diferentes microservicios.

1.1.5. COMPLEJIDAD COGNITIVA

Las métricas de software nos permiten medir y monitorear diferentes aspectos y características del producto software, existen métricas a nivel de diseño, implementación, pruebas, mantenimiento y despliegue. Por lo tanto, estas métricas permiten comprender, controlar y mejorar lo que sucede durante el desarrollo y mantenimiento del software y tomar acciones correctivas y preventivas.

Una de las métricas más importantes es la complejidad ciclomática, que puede ser usada en las fases de desarrollo o mantenimiento entre otras. La complejidad ciclomática es la métrica que nos aporta que tan compleja es la lógica de un programa, se basa en un grafo que representa el diagrama de flujo que viene determinado por la representación de las estructuras de control de un determinado programa [31], es una métrica de software que proporciona una medida cuantitativa de la complejidad lógica de un programa [32].

La introducción de la informática cognitiva en el dominio de la ingeniería de software a través del trabajo de Wang [33] ha llevado a la aparición de un nuevo conjunto de métricas de complejidad denominadas métricas de complejidad cognitiva. Estas métricas introducen pesos cognitivos, que definen el esfuerzo requerido, el tiempo relativo o el grado de dificultad para comprender el software [32]. La Complejidad Cognitiva es una medida del grado de dificultad que implica entender intuitivamente un bloque de código; a diferencia de la complejidad ciclomática, que determina qué dificultad tiene probar el código. Para establecer el valor de la complejidad cognitiva, se establecen puntos en los cuales se deben fijar dentro de un algoritmo, de la siguiente forma: (1) Se incrementa cuando hay salto en el flujo del código (arriba-abajo, izquierda-derecha); algunos elementos que incrementan la complejidad cognitiva son: ciclos, condicionales, excepciones (catching/rescuing), instrucciones switch or case, secuencias de operadores lógicos (a || b && c || d), recursión, salto a etiquetas (go to label), ciclo for. (2) Se incrementa cuando las estructuras de control están anidadas. (3) El código no es más complejo por usar estructuras del lenguaje que nos permitan incluir en una sola línea varias sentencias. Una de las finalidades que busca la complejidad cognitiva es la de incentivar las buenas prácticas a la hora de codificar, para que de esta manera se tenga un producto más entendible y por lo tanto mantenible [31].

1.2. DEFINICIÓN DEL PROBLEMA

Actualmente las empresas de desarrollo de software deben desarrollar, probar y desplegar las aplicaciones realizando lanzamiento de nuevas versiones en periodos de tiempo cada vez más cortos. Anteriormente las versiones de software ocurrirían una o dos veces al año, ahora, dadas las oportunidades competitivas del mercado, se requieren entregas en períodos de tiempo más cortos. Por lo tanto, estas organizaciones necesitan innovar y mejorar su proceso de desarrollo, adoptando prácticas y métodos novedosos en la industria como practicas ágiles, DevOps y microservicios [34].

El desarrollo de aplicaciones es complejo, muchas empresas desarrolladoras presentan baja calidad en el software generado, el código fuente crece tanto que se vuelve difícil de manejar y retrasa la evolución de los productos; los desarrolladores repiten los métodos y procedimientos una y otra vez, dificultando el mantenimiento. El impacto de los cambios es incierto, al realizar un cambio, dada la magnitud y el alto acoplamiento de las aplicaciones, se pueden afectar otros procesos de negocio y se pueden producir bugs de forma incontrolada, causando una baja calidad de la aplicación. Estas fallas se presentan en ocasiones, debido al tamaño de las aplicaciones, a

la gran cantidad de líneas de código y sus dependencias. Además, los atributos de calidad son fundamentales y plantean desafíos para las aplicaciones de hoy en día. La disponibilidad, el rendimiento, el escalado automático, las pruebas automatizadas, la integración y el despliegue continuo, la seguridad y la tolerancia a fallas son características esenciales que toda aplicación debe manejar.

Por otro lado, la gestión de la información de las aplicaciones ha crecido considerablemente, cientos o miles de usuarios se conectan simultáneamente y ejecutan una gran cantidad de transacciones a la vez; lo cual puede afectar el consumo de recursos computacionales (memoria, capacidad de cómputo, latencia de red, ancho de banda, etc), los cuales deben ser administrados eficientemente de tal manera que la aplicación cumpla con sus objetivos y necesidades. Actualmente estos recursos son gestionados y desplegados principalmente en la nube. Administrar, asignar, monitorear, costear y probar esos recursos computacionales en la nube, son tareas tediosas, de mucho cuidado y difíciles de gestionar por los encargados de la infraestructura. Las aplicaciones tradicionalmente suelen ser grandes y monolíticas, por lo que es difícil migrar a la nube y desacoplar este tipo de aplicaciones.

Las prácticas ágiles han ayudado a mitigar estos problemas y han sido acogidas por la industria como esenciales en el proceso de desarrollo de software; también la arquitectura de microservicios ayuda a reducir la complejidad de gestionar y manejar estas características: escalado automático y granular, pruebas automatizadas, integración y despliegue continuo, seguridad y tolerancia a fallas. El desarrollo de aplicaciones basadas en microservicios facilita actualizaciones permanentes, más rápidas y automatizadas usando las prácticas de DevOps, logrando entregas más cortas, automatizadas y probadas ampliamente. Los sistemas de microservicios típicamente requieren la construcción de una tubería “pipeline” de implementación y despliegue continuos que garanticen la calidad de los microservicios y la puesta en producción más rápida. El uso de las tecnologías de la nube, en especial los contenedores pueden facilitar la construcción de dichas tuberías.

El área de investigación en microservicios se encuentra en una etapa formativa, hay inmadurez en este campo y una evolución continua. Los microservicios actualmente han sido adoptados por algunas empresas para desplegar sus negocios, por ejemplo: Amazon, Google, Netflix, Spotify y Twiter. Los microservicios están caracterizados por una rápida adopción industrial, los principales avances se vienen dando en la industria y se aprecia una tendencia creciente en publicaciones y avances científicos desde la academia.

Por otro lado, el estilo arquitectónico de microservicios cambia la forma en que se crean, prueban, implementan y mantienen las aplicaciones. Los microservicios facilitan la migración de aplicaciones a la infraestructura en la nube, escalamiento automático, balanceo de carga y tolerancia a fallas. Al usar microservicios, se puede implementar una aplicación grande como un conjunto de pequeñas aplicaciones (microservicios) que pueden desarrollarse, implementarse, expandirse, administrarse y monitorearse de manera independiente. La agilidad, la reducción de costos y la escalabilidad granular conllevan algunos desafíos como la complejidad de administrar y gestionar sistemas distribuidos [35]. La definición del tamaño óptimo (granularidad) del microservicio es fundamental, ya que afecta directamente el rendimiento de la aplicación, los tiempos de desarrollo y de pruebas, la mantenibilidad, el almacenamiento (transacciones y consultas distribuidas), el uso y consumo de recursos computacionales. El uso y consumo de recursos computacionales, usados principalmente en la nube, dado que la nube es la plataforma más común donde se ejecutan y despliegan los microservicios [36].

Los desafíos de la investigación en el área de microservicios, se centran principalmente en la definición del nivel de granularidad del microservicio, la modularización y refactorización de servicios, la integración con la interfaz de usuario (front-end), en la seguridad, en la orquestación, en el monitoreo, en la gestión y supervisión de microservicios, en la tolerancia a fallas, en la recuperación y auto reparación, en la definición de técnicas, procesos, modelos, herramientas y buenas prácticas a nivel de diseño, implementación y mantenimiento de microservicios [37].

Así mismo, existe una brecha de investigación en el área de diseño y modelaje de la arquitectura de aplicaciones basadas en microservicios: N. Alshuqayran y sus colaboradores (2016) en su revisión evidencian la necesidad de proponer una vista / lenguaje de modelado integral que cubra y describa mejor una arquitectura basada en microservicios, no existe un estándar que permita representar los microservicios y sus relaciones [38]. Esta afirmación se confirma por Di Francesco y otros (2017), que evidenciaron poca atención a los patrones de diseño y lenguajes arquitectónicos para microservicios, concluyen que el uso de lenguajes arquitectónicos informales y la falta de un lenguaje arquitectónico predominante pueden indicar dificultades en la descripción y el modelado de arquitecturas de microservicio [39], finalmente en la perspectiva tecnológica presentada por J. Lewis y otros (2018) afirman que no se presenta una herramienta y/o método que permita modelar y representar las relaciones existentes entre los microservicios que hacen parte de una aplicación [23], evidenciando un vacío de investigación en esta temática.

Adicionalmente, el tamaño del microservicio o granularidad optima es una de las propiedades más discutidas, se han propuesto pocos patrones, métodos o modelos que permitan determinar qué tan pequeño debe ser un microservicio. Soldani y otros (2018) afirman que se presenta dificultad para identificar las capacidades del negocio y los contextos delimitados que pueden ser asignados a cada microservicio [40]. J. Bogner, S. Wagner, y A. Zimmermann (2018) enuncian que se requieren metodologías y técnicas que faciliten el dimensionamiento y el versionado de los microservicios [41]. O. Zimmerman (2017) propone las siguientes preguntas ¿Cómo encontrar un corte de servicio adecuado, que tan pequeño o fino es lo suficientemente pequeño? ¿Cómo se puede aplicar DDD (Domain driven design) u otros enfoques para definir el alcance de la aplicación y la partición funcional, para descomponer los monolitos en microservicios? Dice en su revisión qué: Los profesionales han informado que agradecerían una orientación más concreta que el consejo más frecuentemente declarado "definir un contexto limitado para cada concepto de dominio que se expondrá como servicio” [21]; Lewis y otros (2018), afirman que encontrar los módulos correctos, con el tamaño correcto, la asignación correcta de responsabilidades y las interfaces bien diseñadas, es un desafío; además enuncian que se evidencia una falta de acuerdo sobre el tamaño correcto de los microservicios. El hecho de que estén etiquetados como "microservicios" muestra que existe la posibilidad de establecer un conjunto de patrones para ayudar con las decisiones de diseño cuando se está dividiendo un dominio en microservicios y dimensionando cada microservicio [23].

Por otro lado, la inteligencia artificial, en especial las técnicas de aprendizaje automático y aprendizaje profundo están siendo utilizadas en muchos campos de la ciencia, la industria y aplicaciones comerciales. Además se está incorporando la inteligencia artificial en aplicaciones nativas de la nube a través de herramientas que les permiten incorporar estas características como microservicios basados en datos; aplicaciones innovadoras como los chatbots cognitivos, el reconocimiento facial automatizado y la búsqueda basada en imágenes dependen de la provisión de las tecnologías de inteligencia artificial dentro de las arquitecturas nativas de la nube usando microservicios [42]. Diferentes autores han propuesto el uso de la inteligencia artificial para el monitoreo de atributos de calidad de los microservicios para poder tomar acciones correctivas y preventivas mejorando su funcionamiento y calidad del servicio. En este trabajo se pretende usar la inteligencia artificial para determinar de forma automática o semiautomática el

conjunto de funcionalidades que debe contener cada microservicio, es decir, el modelo inteligente debe tener la capacidad de establecer la granularidad de cada microservicio, teniendo en cuenta las métricas de complejidad cognitiva, acoplamiento, cohesión y dependencias.

En conclusión, se pudieron evidenciar vacíos científicos y tecnológicos en el área de desarrollo de aplicaciones basadas en la arquitectura de microservicios, por lo tanto, se pretende con este trabajo de investigación doctoral abordar algunos de ellos, en especial los correspondientes al diseño y modelaje de la arquitectura de las aplicaciones basadas en microservicios y a la definición de su granularidad. En consecuencia, se pretende elaborar un modelo inteligente que permita modelar, componer y definir la granularidad adecuada de los microservicios que van a ser parte de la aplicación, teniendo en cuenta la complejidad cognitiva, el acoplamiento, la cohesión, sus dependencias y el tiempo de desarrollo; cubriendo así algunos de los vacíos de investigación planteados.

1.3. PREGUNTAS DE INVESTIGACIÓN

Una vez detallada la problemática e identificado los vacíos de investigación presentes en el estado del arte y de la práctica, se procede a plantear el siguiente problema de investigación:

¿Cómo modelar y definir el tamaño adecuado de un microservicio, teniendo en cuenta sus propiedades, dependencias, su complejidad cognitiva, su acoplamiento y cohesión?

Para resolver el problema de investigación, se han planteado las siguientes preguntas de investigación específicas (Research Question - RQ) que ayudan a resolverlo:

Pregunta de investigación 1 - RQ-1: ¿Qué desafíos, problemas y avances de investigación presentan el diseño, modelaje, la arquitectura y la definición del nivel de granularidad de aplicaciones basadas microservicios?

Con esta pregunta de investigación se identificaron los vacíos de investigación, los problemas y las tendencias actuales en microservicios; además se describieron los modelos de especificación existentes, lenguajes, herramientas y métodos arquitectónicos utilizados en aplicaciones basadas en microservicios. Se llevó a cabo una revisión sistemática de la literatura, siguiendo la guía propuesta por Kitchenham (2004) [43], consultando las bases de datos de las publicaciones científicas más importantes en el área de informática como IEEE, ACM y Scopus.

Pregunta de investigación 2 - RQ-2: ¿Qué aspectos y/o propiedades de la granularidad se están teniendo en cuenta en el desarrollo de aplicaciones basadas en microservicios?, ¿cómo se definen?,¿cómo se modelan y qué métodos se están usando?

Para resolver esta pregunta primero se caracterizó y estudió el proceso de desarrollo de aplicaciones basadas en microservicios, identificando las fases, los procesos, las herramientas (las conocidas y publicadas hasta el momento de la revisión) y las actividades que son fundamentales en la construcción de aplicaciones basadas en microservicios. Luego se identificaron las propiedades que afectan el modelado, la granularidad y la complejidad cognitiva de los microservicios. Para caracterizar completamente el proceso de desarrollo, se implementó una aplicación basada en microservicios utilizando las buenas prácticas propuestas en la literatura.

Pregunta de investigación 3 - RQ-3: ¿Qué relación existe entre la granularidad, la complejidad cognitiva, el acoplamiento, la cohesión y las dependencias de los microservicios? ¿cómo se expresan de forma matemática?

Basado en la revisión de la literatura llevada a cabo en RQ-1, se planteó formalmente una relación entre la granularidad, la complejidad cognitiva, el acoplamiento, la cohesión y las dependencias de los microservicios. Se identificaron, adaptaron y plantearon las métricas que permiten medir y evaluar estas propiedades en tiempo de diseño, y a partir de ellas definir la granularidad adecuada de los microservicios.

Pregunta de investigación 4 - RQ-4:

¿Cómo representar, clasificar y agrupar de forma inteligente el conjunto de requerimientos de una aplicación expresados en un "product backlog" como una nueva práctica ágil denominada “Microservices Backlog”, que permita modelar y evaluar el nivel de la granularidad de los microservicios que hacen parte de la aplicación?

Se puede definir el “Microservices Backlog” como el conjunto de funcionalidades distribuidas en microservicios, dependencias y métricas que cumplen con los requerimientos de la aplicación.

Tomando como punto de partida el listado de requerimientos expresados en el “product backlog”, se propuso el “Microservices Backlog” un modelo inteligente de especificación que detalla los microservicios que van a cumplir los requerimientos (expresados como historias de usuario) de la aplicación, junto con las relaciones existentes entre cada uno de ellos y el cálculo de las métricas de complejidad cognitiva, acoplamiento, cohesión y tiempo de desarrollo. Con base en el modelo el desarrollador de software o arquitecto pudo probar y evaluar cómo será implementada su aplicación, definiendo con ayuda del modelo propuesto la granularidad adecuada de cada microservicio.

Al abordar esta pregunta se identificó la forma de especificar las relaciones entre los microservicios que son parte de una aplicación y se describieron las características clave que esa especificación debe gestionar. Luego, con base a esa especificación, se agruparon las historias de usuario en los microservicios, mapeadas automáticamente usando algoritmos inteligentes (un algoritmo genético y un algoritmo de agrupamiento), generando el “Microservices Backlog”.

El modelo de especificación “Microservices Backlog” muestra gráficamente los microservicios que serán parte de la aplicación relacionándolos con los requerimientos o funcionalidades que va a contener. A partir de esta especificación, el desarrollador o arquitecto puede probar, evaluar y comparar las métricas calculadas de la aplicación que está construyendo, comparando con diferentes soluciones y distribuciones, pudiendo así obtener el nivel de la granularidad más adecuado de los microservicios, basados en métricas de complejidad cognitiva, cohesión y acoplamiento.

Pregunta de investigación 5 - RQ-5: ¿Cómo demostrar que el modelo inteligente de definición de la granularidad proporciona una mejora en el proceso de desarrollo de aplicaciones que utilizan microservicios?

Una vez definido el modelo de especificación de la granularidad de los microservicios, se desarrollaron 4 casos de estudio, entre los cuales 2 fueron casos típicos propuestos en el estado del arte y 2 fueron casos reales. Se implementó una aplicación basada en microservicios (un estudio de caso), se utilizó el modelo propuesto “Microservices backlog”, para probar, validar y

refinar esta propuesta, verificando que el “Microservices backlog” representa una mejora. Para los casos de estudio, se diseñó un conjunto de microservicios que son parte de la aplicación.

Luego se realizó un análisis comparativo de la solución propuesta con otros enfoques similares, de tal forma que se pueda proveer una evaluación sobre la aplicabilidad y uso del modelo, se comparó con los trabajos relacionados donde se defina la granularidad de los microservicios, resaltando las diferencias fundamentales y el aporte del modelo propuesto, este análisis se realizó siguiendo el proceso de experimentación propuesto por Wohlin y otros (2012) [44].

1.4. OBJETIVOS

1.4.1. OBJETIVO GENERAL

Elaborar un modelo inteligente llamado “Microservices backlog” con el fin de modelar y definir la granularidad de los microservicios que hacen parte de la aplicación, usando técnicas de inteligencia artificial, teniendo en cuenta las siguientes propiedades: cohesión, acoplamiento, complejidad cognitiva y tiempo de desarrollo.

1.4.2. OBJETIVOS ESPECÍFICOS

1. Identificar los desafíos y problemas de investigación presentados en el diseño, modelaje, la arquitectura y la definición del nivel de granularidad de aplicaciones basadas microservicios.
2. Caracterizar el proceso de desarrollo de aplicaciones basadas en microservicios, identificando fases, procesos, herramientas y actividades fundamentales dentro de ese proceso de desarrollo; con el fin de identificar los aspectos y/o propiedades que afectan la definición de la granularidad de los microservicios.
3. Proponer un modelo matemático que defina la relación entre la granularidad, la complejidad cognitiva, el acoplamiento, la cohesión y las dependencias de tal forma que se pueda definir la granularidad de cada microservicio que hace parte de una aplicación.
4. Definir el modelo inteligente de especificación de la arquitectura de las aplicaciones que usan microservicios llamado “Microservice Backlog” con el fin de modelar y evaluar la granularidad de los microservicios que hacen parte de una aplicación.
5. Demostrar que el modelo de especificación “Microservices Backlog” propuesto representa una mejora (en términos de definición de la granularidad, la complejidad cognitiva y el tiempo de desarrollo) al proceso de desarrollo de aplicaciones que utilizan microservicios.

1.5. METODOLOGÍA.

La metodología de este trabajo de investigación doctoral se basó en “Design Science Research” siguiendo los planteamientos hechos por Hevner y otros [45] para la investigación científica en el campo de los sistemas de información y el desarrollo de productos tecnológicos. El paradigma de la ciencia del diseño busca extender los límites de las capacidades humanas y organizacionales mediante la creación de artefactos nuevos e innovadores [45]. En la presente investigación el artefacto creado fue el modelo inteligente de especificación de la granularidad de los microservicios que hacen parte de una aplicación “Microsevices Backlog”.

Este trabajo de investigación se dividió en tres fases: (1) Fundamentación, (2) Desarrollo y evaluación y (3) Comunicación y divulgación. El modelo de la investigación se presenta en la figura 8. A continuación se detallan las actividades principales que se desarrollaron en cada una de las fases de la presente investigación.

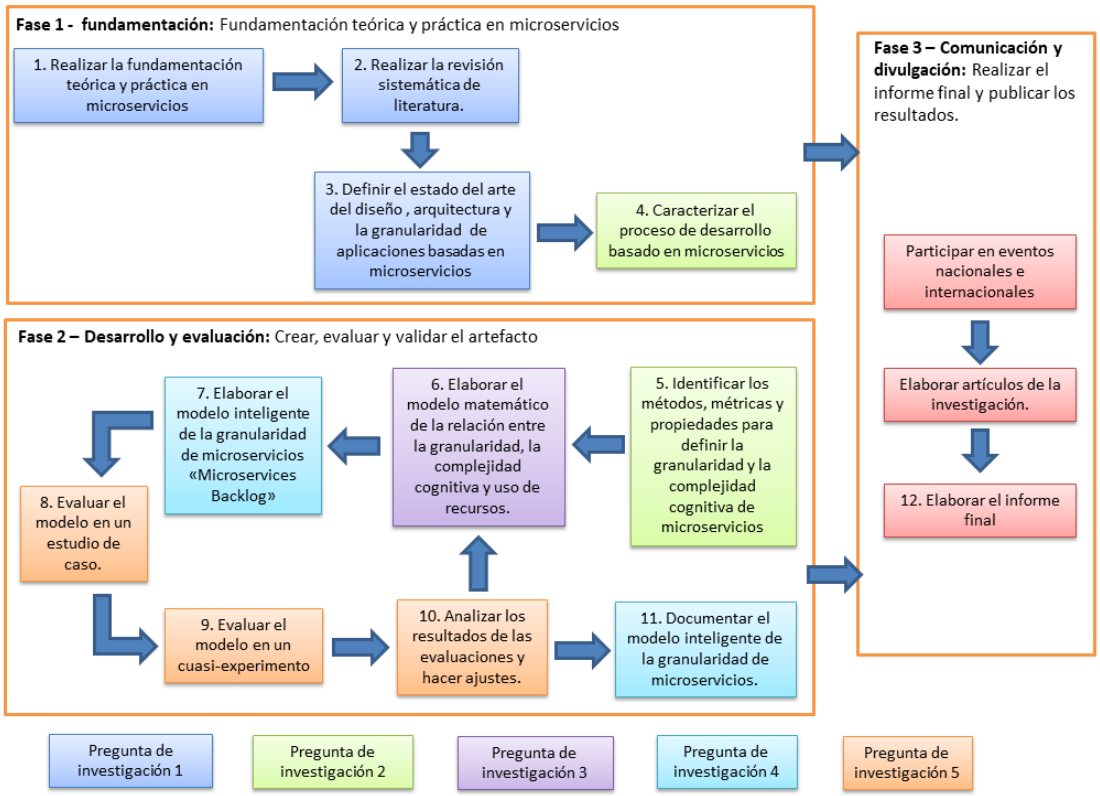


Figura 8. Modelo de la investigación

El trabajo de investigación comenzó con la fase 1 - fundamentación teórica y práctica en microservicios, se realizó la revisión sistemática de literatura con el fin de identificar los vacíos y retos de investigación en el desarrollo de aplicaciones que usan microservicios, posteriormente se realizó el estado del arte identificando los trabajos relacionados para su posterior evaluación y comparación con nuestra propuesta. También se caracterizó el proceso de desarrollo de aplicaciones que usan microservicios identificando las herramientas, métodos y procesos usados en la definición de la granularidad de los microservicios.

La fase 2 – Desarrollo y evaluación, inició con la identificación de los métodos, métricas y propiedades que afectan el modelado, la granularidad y la complejidad cognitiva de microservicios. Teniendo identificados esos métodos y procedimientos se procedió a plantear el modelo matemático para definir el nivel de granularidad de los microservicios que hacen parte de una aplicación. Luego se plantea un ciclo iterativo en el cual se elabora el modelo inteligente de especificación y se realiza su validación en dos partes primero en estudios de caso y luego en un cuasi-experimento. Este ciclo se realiza hasta resolver el problema de investigación de forma correcta y completa.

Por último, la fase 3 – Comunicación y divulgación de resultados en la cual se realizó la tesis doctoral y se publicaron los resultados del trabajo. Se obtuvieron publicaciones tanto a nivel local, regional, nacional e internacional. Se participó en conferencias nacionales e internacionales en el área de ingeniería del software, orientación a servicios y/o microservicios. Esta fase se realizó de forma paralela a las demás a medida que se iban resolviendo las preguntas de investigación y los objetivos. Prácticamente se logró la escritura de un artículo científico por cada objetivo

planteado, la mayoría de ellos ya están publicados y los otros se encuentran en proceso de evaluación.

En la tabla 2 se presentan las fases, actividades, métodos y resultados esperados del proceso de investigación planteado para este trabajo de investigación doctoral.

Al terminar las tres fases propuestas, se resolvieron las preguntas de investigación, se cumplieron los objetivos y se aplicaron los métodos de evaluación planteados, generando nuevo conocimiento para beneficio de la Universidad y de la comunidad desarrolladora de software.

Tabla 2. Fases, actividades, métodos y resultados esperados del proceso de investigación

Fase	Actividades	Métodos a usar	Resultados esperados
Fase 1: Fundamentación	Realizar la fundamentación teórica y práctica en microservicios. Realizar la revisión sistemática de literatura. Definir el estado del arte del desarrollo basado en microservicios. Caracterizar el proceso de desarrollo de aplicaciones basadas en microservicios.	Análisis y síntesis de bibliografía. Talleres prácticos de desarrollo y despliegue de microservicios y aplicaciones que los usan. Método de revisión sistemática basado en lo propuesto por Kitchenham [43].	Artículo científico de revisión sistemática de literatura enviado a evaluación en una revista internacional. Informe técnico que detalla la caracterización del proceso de desarrollo de aplicaciones basadas en microservicios junto con sus buenas prácticas. Artículo científico que detalla la caracterización del proceso de desarrollo de aplicaciones basadas en microservicios junto con sus buenas prácticas enviado a evaluación en una revista internacional. Una ponencia nacional y una internacional donde se socialicen los resultados de esta fase.
Fase 2: Desarrollo y evaluación	Identificar los métodos, métricas y propiedades que afectan el modelado, la granularidad y la complejidad cognitiva, el acoplamiento, la cohesión y el uso de recursos de los microservicios. Elaborar el modelo matemático de la relación entre la granularidad, la complejidad cognitiva, el acoplamiento, la cohesión y el uso de recursos. Elaborar el modelo inteligente de definición de la granularidad de los microservicios. Evaluar el modelo en estudios de caso. Evaluar el modelo en cuasi-experimento. Analizar los resultados de las evaluaciones. Documentar el modelo inteligente de definición de la granularidad de los microservicios “Microservice Backlog”.	Análisis y síntesis de bibliografía. Casos de estudio Cuasi – experimentos	Artículo científico donde se propone y evalúa el modelo matemático de la relación entre la granularidad, la complejidad cognitiva y el rendimiento. Artículo científico donde se propone y evalúa el modelo inteligente de definición de la granularidad de los microservicios. Informe técnico donde se detalla el modelo inteligente de definición de la granularidad de los microservicios. Aplicaciones basadas en microservicios desarrolladas en los casos de estudio (Por lo menos una). Una ponencia nacional y una internacional donde se socialicen los resultados de esta fase.
Fase 3: Comunicación y divulgación	Elaborar el informe final de la investigación. Elaborar artículos de investigación para su presentación en eventos internacionales.	Análisis y síntesis de información.	4 artículos de investigación que detalla los aportes obtenidos en el trabajo de investigación doctoral enviados a revistas internacionales, como resultado de las fases anteriores. 2 ponencias nacionales y 2 internacionales, resultado de las fases anteriores. Tesis doctoral.

1.5.1. MÉTODOS DE INVESTIGACIÓN

Los siguientes fueron los métodos y los instrumentos con los que se validaron y evaluaron los resultados de este trabajo de investigación. Se evaluó y analizó los siguientes puntos:

El modelo inteligente de definición de la granularidad de los microservicios “Microservices Backlog” permitió definir la granularidad adecuada de los microservicios y evaluar la complejidad cognitiva, el acoplamiento, la cohesión y las dependencias de la aplicación basada en microservicios.

MÉTODO DE EVALUACIÓN 1: CUASI - EXPERIMENTO: Se planteó una evaluación cuasi – experimental, donde se analizó y comparó el modelo propuesto contra otros métodos de definición de la granularidad identificados en el estado del arte. Se calcularon las métricas de evaluación sobre las descomposiciones propuesta por esos métodos a los mismos casos de estudio; las mediciones y comparaciones se realizaron sobre el tiempo desarrollo, el número de microservicios, la granularidad y la complejidad cognitiva de cada uno de ellos. Se identificaron las diferencias y aportes fundamentales.

MÉTODO DE EVALUACIÓN 2: ESTUDIO DE CASO: El estudio de caso se utilizó para validar en algún entorno de organización y proyecto real de aplicación, o utilizando un problema real y planteando la solución utilizando la propuesta. Se utilizó el modelo inteligente de definición de la granularidad de los microservicios en cuatro casos de estudio; se pudo revisar, evaluar y ajustar el modelo propuesto.

2. ESTADO DEL ARTE

Para definir el estado del arte del presente trabajo de investigación se realizaron dos revisiones de literatura, la primera para abordar el de forma general el desarrollo de aplicaciones basadas en microservicios, enfocados en las fases del proceso de desarrollo; y la segunda, una revisión sistemática de literatura, más específica en la definición de la granularidad de los microservicios. A partir de estas revisiones se definieron las tendencias y desafíos de investigación, se caracterizó el proceso de desarrollo de aplicaciones basadas en microservicios, se identificaron y detallaron los trabajos relacionados y se describieron las métricas usadas para definir y evaluar la granularidad de los microservicios.

2.1. APLICACIONES BASADAS EN MICROSERVICIOS: TENDENCIAS Y DESAFÍOS DE INVESTIGACIÓN

Se realizó una revisión de literatura para identificar y describir las tendencias de investigación actuales en el proceso de desarrollo de aplicaciones basadas en microservicios [37]. La búsqueda se realizó en las bases de datos de publicaciones científicas más importantes del mundo en el área de la computación, como son IEEE, Scopus y ACM. Se plantearon las preguntas de investigación, luego se definieron 4 cadenas o criterios de búsqueda (ver tabla 3) y se aplicaron en cada base de datos. Luego, se definen los criterios de inclusión y exclusión usados para seleccionar los artículos que fueron parte de este estudio.

Las preguntas de investigación cubiertas en esta revisión de literatura fueron:

1. ¿Cuáles son las tendencias actuales en los trabajos de investigación en el desarrollo de aplicaciones basadas en microservicios?
2. ¿Cuáles son los principales desafíos pendientes de resolver en el desarrollo de aplicaciones basadas en microservicios?
3. ¿Qué lenguajes, modelos o herramientas de especificación de la arquitectura de microservicios han sido publicados?

Tabla 3. Criterios de búsqueda

Criterio de busqueda	Descripción	Pregunta a resolver
1. Microservice or micro service	Se buscaron todas las publicaciones relacionadas con los microservicios.	Pregunta de investigación 1 y 2
2. "Microservices" and "Web Development"	Trabajos relacionados al desarrollo de aplicaciones web y microservicios	Pregunta de investigación 1 y 2
3. "Microservices" and "Agile"	Trabajos relacionados entre microservicios y metodologías ágiles.	Pregunta de investigación 1 y 2
4. ("micro service" OR microservice) AND (design OR model OR specification)	Trabajos relacionados entre microservicios, diseño, modelaje y especificación	Pregunta de investigación 3

Criterios de inclusión:

- Los trabajos de revisión de literatura, mapeo o revisión sistemática de literatura, los cuales identifican vacíos y tendencias de investigación en microservicios.
- Artículos de investigación primarios que se centran en el diseño, la arquitectura, el desarrollo, el despliegue, el monitoreo o pruebas de aplicaciones basadas en microservicios.

- Artículos de investigación primarios que proponen lenguajes, modelos o herramientas de especificación de la arquitectura de aplicaciones basadas en microservicios.
- Trabajos presentados en conferencias internacionales, revistas internacionales y capítulos de libros en el campo de la ingeniería del software.

Criterios de exclusión:

- Artículos de opinión, tutoriales, ejemplos o experiencias que no realicen un aporte al área de investigación de los microservicios.
- Artículos que utilizan a los microservicios como un tópico secundario.
- Literatura disponible solo en forma de resumen, blogs, videos o presentaciones.
- Artículos que no están escritos en inglés o español

Se detallaron y clasifican por fase del proceso de desarrollo los vacíos de investigación y posibles trabajos futuros. El método llevado a cabo para recopilar la información y realizar la revisión de literatura se puede apreciar en la figura 9. Primero se definieron los criterios de búsqueda en cada una de las bases de datos, luego se seleccionaron para cada criterio de búsqueda los artículos más relevantes y los artículos más citados, posteriormente se ordenaron y eliminaron los artículos repetidos, luego se aplicaron los criterios de inclusión y exclusión, luego se revisaron uno a uno los artículos seleccionados, detallando el trabajo y los aportes realizados.

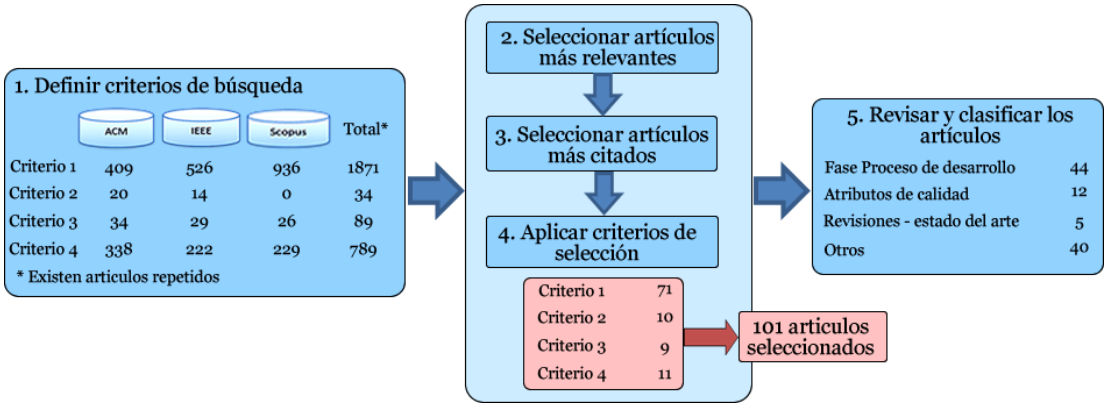


Figura 9. Resultados en cada etapa del método llevado a cabo en la revisión de literatura

2.1.1. Tendencias de investigación en desarrollo de aplicaciones basadas en microservicios.

Para identificar las tendencias y los vacíos de investigación en el desarrollo de aplicaciones basadas en microservicios, se clasificaron en las fases del proceso de desarrollo y en las áreas operacionales que se muestran en la tabla 4. Además, se clasificaron según los atributos de calidad estudiados y en otros factores diferentes al proceso de desarrollo. Los criterios de clasificación se basan en el trabajo propuesto por Wieringa y otros [46] y se usan ampliamente en otras revisiones de literatura [6], [40], [47]. Las áreas operacionales se identificaron también a partir esas revisiones de literatura.

La Figura 10.a muestra la distribución por fase del proceso de desarrollo de los artículos revisados. Se puede observar que el foco de la investigación se centra en el desarrollo o implementación de microservicios con el 22% de los artículos, seguido por la fase de diseño con el 10%, pruebas y despliegue también han sido abordadas con el 6%. No se encontraron artículos

específicos en la fase de monitoreo, siendo este un punto importante de los microservicios que requiere mayor investigación.

Tabla 4. Fases del proceso de desarrollo y áreas operacionales

Fase del proceso de desarrollo	Area Operacional
Diseño	1) Modelado de procesos de negocio. 2) Arquitectura
Desarrollo	1) Lenguaje de programación, 2) IDE de desarrollo, 3) Funcionalidad, 4) Descubrimiento de servicios, 5) Granularidad, 6) Composición – integración, 7) Buenas prácticas, 8) Base de datos, 9) Multitenant
Pruebas	Pruebas (sin división)
Despliegue	1) Despliegue, 2) DevOps
Monitoreo	1) Monitoreo, 2) Analíticas

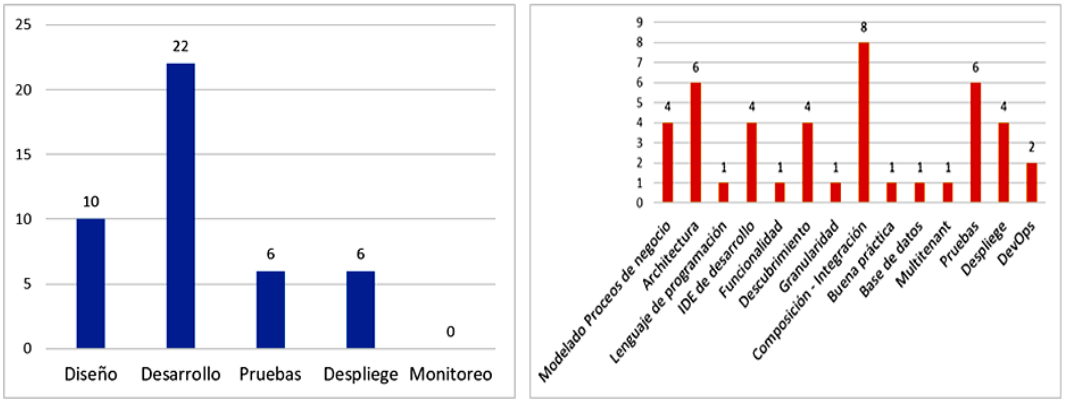
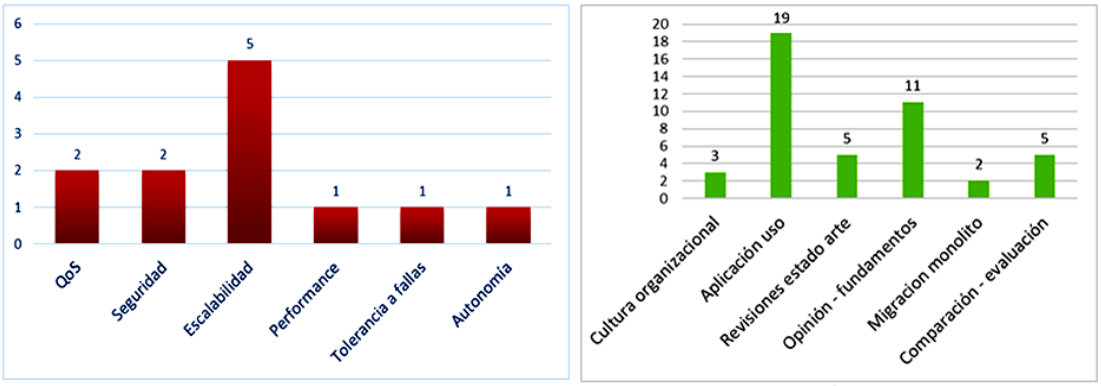


Figura 10. a. Distribución de los artículos por fase del proceso de desarrollo. **b.** Distribución de los artículos por área operacional.

La Figura 10.b muestra estos resultados detallados por área operacional, corresponden a 44 de los artículos revisados que representan el 43,6%. En esta división, el mayor número de trabajos corresponde a las técnicas o métodos de composición e integración de microservicios con un 18.2%, seguido por modelos o métodos para definir la arquitectura de microservicios y pruebas con 13.6%. En el siguiente grupo aparecen áreas como modelado de procesos de negocios, entornos de desarrollo integrados, descubrimiento de microservicios y despliegue con un 9.1%. La definición de la granularidad de microservicios, los lenguajes de programación específicos para microservicios, la funcionalidad, la definición de buenas prácticas, las técnicas para refactorizar la base de datos, el uso de arquitecturas “multitenan”, con el 2,3% de los artículos; evidenciando que hay vacíos de investigación en estos temas. Luego aparece DevOps con un 4,5%, siendo bajo el número de los artículos, DevOps es fundamental para el desarrollo de aplicaciones basada en microservicios. En otras áreas como la orquestación, la disponibilidad, el monitoreo y las analíticas no se encontraron trabajos, evidenciado que hacen falta investigaciones.

En la figura 11.a se presentan los artículos que se enfocan en atributos de calidad, correspondiendo a 12 artículos siendo el 11.9%. El atributo de calidad más estudiado es la escalabilidad con 41.7%, seguido por la calidad del servicio QoS y seguridad con 16.7%. Por último, aparecen atributos como el rendimiento, la tolerancia a fallas y la autonomía con 8.3%.



a. b.

Figura 11. a. Distribución de los artículos por atributos de calidad. **b.** Distribución de los artículos por otros factores diferentes al proceso de desarrollo.

En la figura 11.b, se clasifican los artículos por otros factores diferentes al proceso de desarrollo, encontrando 45 artículos en esta división (45.9%). Se puede apreciar que la aplicación y uso de la arquitectura de microservicio en otras áreas es la de mayor número de trabajos con el 42% de ellos, seguidos por los artículos de opinión y fundamentos con 24.4%, estos artículos discuten las propiedades y plantean los diferentes conceptos teóricos y prácticos de los microservicios. Luego siguen las revisiones de literatura o estados del arte junto con las comparación y evaluación de los microservicios 11.11%. Por último, aparecen artículos que discuten la cultura organizacional y las migraciones de monolitos a microservicios.

En resumen, la tabla 5 condensa las tendencias de investigación encontradas en esta revisión de literatura. Para finalizar, se analizaron otras áreas de aplicación de los microservicios, estas se pueden apreciar en la figura 12.

Tabla 5. Tendencias de investigación en microservicios

Area de clasificación	Porcentaje de artículos	Tendencia de investigación
Proceso de desarrollo	43.6%	1) Fase de desarrollo: Técnicas de composición e integración. 2) Fase de diseño: Arquitectura. 3) Pruebas.
Atributos de calidad	11,9%	1) Escalabilidad, 2) Calidad del servicio – QoS. 3) Seguridad
Otros factores	44.5%	1) Aplicación uso en otras áreas. 2) Opinión y fundamentos de los microservicios

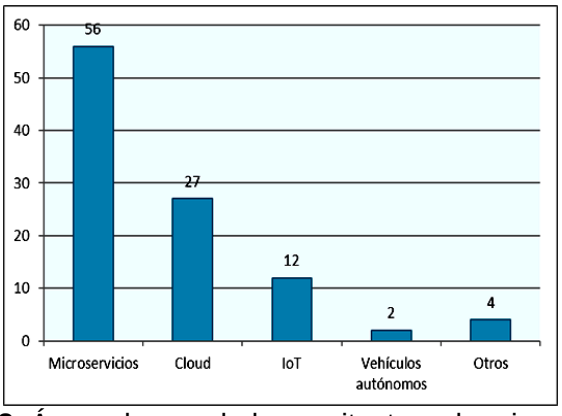


Figura 12. Áreas de uso de la arquitectura de microservicios.

En primer lugar, aparecen los trabajos que se enfocan en mejorar algún aspecto de los microservicios, estos incluyen los artículos por fase del proceso de desarrollo y los que estudian los atributos de calidad, correspondientes al 55.5% de los artículos revisados.

Luego aparecen los trabajos que se centran en estudiar temas específicos de la computación en la nube (Cloud) correspondientes al 26,7%, la relación de los microservicios y la nube es estrecha, sobre todo en la forma como se despliegan sobre infraestructuras como servicio: máquinas virtuales y contenedores principalmente; también temas como DevOps, gestión de contenedores, el despliegue en la nube, sistemas operativos heterogéneos, el ahorro de energía, la simulación de eventos distribuidos y paralelos y los centros de datos distribuidos.

La arquitectura de microservicios puede mejorar diferentes aspectos de las tecnologías del Internet de las Cosas (IoT), en esta revisión se encontraron artículos que abordan estas temáticas y corresponden al 11.9% de nuestra revisión. Otros usos incluyen redes de telecomunicaciones donde incluyen la arquitectura de microservicios, sistemas de salud E-health, vehículos autónomos y comercialización de energía eléctrica. Estos corresponden al 5,9% de los trabajos revisados.

2.1.2. Desafíos de investigación en el desarrollo de aplicaciones basadas en microservicios.

A partir de la revisión de literatura, se identificaron los vacios de investigación y se clasificaron por área del proceso de desarrollo y se resumen a continuación. La figura 13 presenta los vaciós de investigación identificados en la fase de diseño y en la fase de desarrollo.

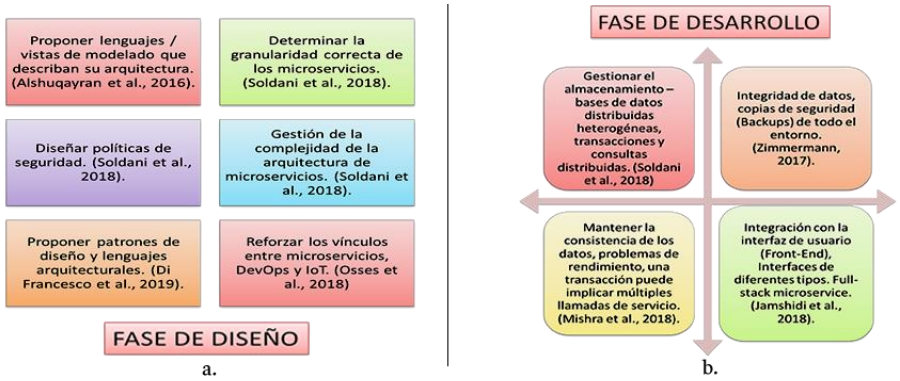


Figura 13. a. Desafíos de investigación fase de diseño. **b.** Desafíos de investigación fase de desarrollo.

Los retos en la fase de diseño se centran en proponer herramientas o lenguajes de modelado, determinar la granularidad correcta de los microservicios, determinar qué tan pequeño debe ser, cuántas funcionalidades debe incluir y manejar, también definir patrones de diseño y lenguajes arquitecturales para gestionar la complejidad de las aplicaciones basadas en microservicios. DevOps es fundamental, proponer trabajos que profundicen DevOps, microservicios y IoT [48].

En la fase de desarrollo los retos de investigación se presentan en la figura 13.b, la gestión de los datos es uno de los retos más importantes y muy discutidos. Gestionar bases de datos distribuidas, con transacciones y consultas distribuidas es un tema abierto que requiere más investigación. Mantener la consistencia de los datos en un sistema distribuido sin afectar el rendimiento es un tema de mucho interés [49]. Gestionar la interfaz de usuario con diferentes

dispositivos y de diferentes tipos, se requiere ayudar a las organizaciones a implementar un entorno de microservicio de pila completa (Full-Stack microservices) [23]

La figura 14.a, presenta los vacíos de investigación presentes en la fase de pruebas y en la fase de despliegue, las pruebas han sido estudiadas y se cuenta con estrategias de pruebas automáticas e integradas para los microservicios, estás pruebas normalmente hacen parte de una tubería (pipeline) de integración y despliegue continuo, aunque se siguen presentando dificultades al gestionarlas de forma distribuida entre diferentes contenedores y/o microservicios. La fase de despliegue presenta los desafíos que se resumen en la figura 14.b se requieren métodos de automatización del despliegue continuo, la gestión de recursos de red y computacionales principalmente desplegados en la nube.

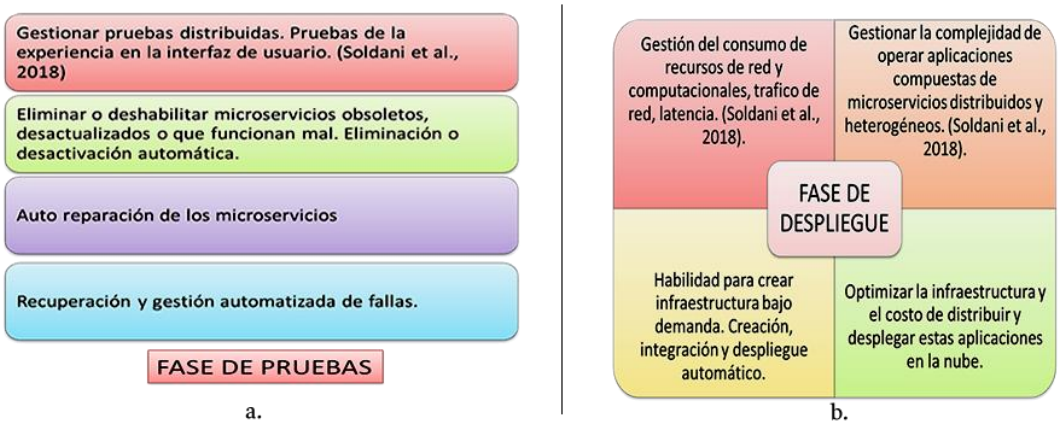


Figura 14. a. Desafíos de investigación fase de pruebas. **b.** Desafíos en la fase de despliegue.

Adicionalmente, la seguridad es un área de mucho interés, específicamente el control de acceso es un reto de investigación, el control consistente y descentralizado de la aplicación, se presenta proliferación de EndPoints generando mayores puntos vulnerables a ataques, actualmente se realiza un soporte centralizado para el control de la seguridad [40]. El monitoreo es otro punto importante, monitorear un conjunto heterogéneo de microservicios que evolucionan dinámicamente.

El manejo de logs, gestionar logs distribuidos, uno por cada microservicio, también la localización de problemas son otras preocupaciones importantes en la operación de aplicaciones basadas en microservicios. Definir umbrales y filtros de alerta apropiados, para notificar a los desarrolladores cuando algo sale mal sin sobrecargarlos con información redundante o irrelevante [23]. Brechas de investigación pueden existir en las áreas relacionadas con estos atributos de calidad: Seguridad, confiabilidad y portabilidad [47].

2.1.3. Lenguajes, modelos o herramientas de especificación de la arquitectura de microservicios.

El diseño de aplicaciones basadas en microservicios presenta algunos desafíos interesantes, se han propuesto trabajos que permiten especificar y modelar la arquitectura de las aplicaciones basadas en microservicios. La figura 15 presenta estos trabajos.

Se puede apreciar que se han propuesto y validado lenguajes de programación específicos para microservicios [50], ambientes integrados de desarrollo [51], lenguajes y herramientas de modelado que permiten especificar la arquitectura de aplicaciones basadas en microservicios y

luego generar el código de la aplicación [52], [53], [54], como también herramientas de modelado de procesos de negocio y flujos de trabajo [55], [56].

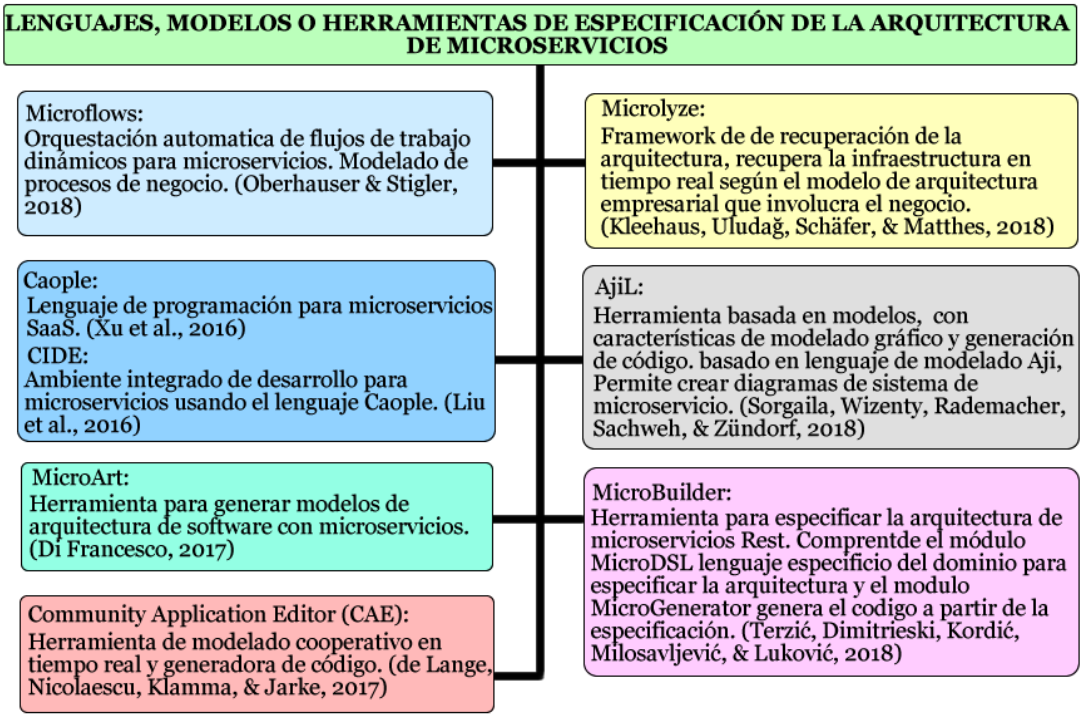


Figura 15. Lenguajes, modelos o herramientas de especificación de la arquitectura de microservicios.

En conclusión, las tendencias de investigación se encuentran en la fase de desarrollo, el cómo implementar y usar los microservicios es un tema muy estudiado, junto con las técnicas de composición, integración y pruebas; luego en la fase de diseño, la arquitectura ya ha sido muy investigada, se han propuesto herramientas, modelos y lenguajes de definición de la arquitectura propios para los microservicios. La aplicación y uso de los microservicios en otras áreas y los artículos de opinión y fundamentos de los microservicios son otras tendencias, estos artículos establecen las bases conceptuales y metodológicas para poder usar los microservicios. Los atributos de calidad más estudiados son la escalabilidad y la calidad del servicio – QoS, son las razones por las cuales los microservicios surgieron y son tan populares.

Los desafíos de la investigación se centran principalmente en la definición del nivel de granularidad de los microservicios, en la modularización y refactorización de servicios, en la integración con la interfaz de usuario (front-end), en la seguridad, en la orquestación, en el monitoreo, en la gestión y supervisión de microservicios, en la tolerancia a fallas, en la recuperación y auto reparación, en la definición de técnicas, procesos, modelos, herramientas y buenas prácticas a nivel de diseño, implementación y mantenimiento de microservicios. Esto representan temas interesantes para futuras investigaciones.

A partir de esta revisión esta tesis doctoral se enfocó en la definición del nivel de granularidad de los microservicios; luego de la revisión que muestra el panorama global de la arquitectura basada en microservicios, se realizó una revisión sistemática de literatura más específica, con el propósito de identificar los trabajos relacionados y las propuestas publicadas que definen la granularidad de los microservicios.

2.2. DEFINICIÓN DE GRANULARIDAD DE MICROSERVICIOS: ENFOQUES PROPUESTOS.

Granular significa "hecho de, o que parece, gránulos (un pequeño trozo como un grano de algo)" [57]. Un gránulo es único, atómico, indivisible y pequeño; estas son las principales propiedades de un servicio. Los microservicios son unidades de responsabilidad única (gránulos) que encapsulan los datos y la lógica de procesamiento, se despliegan a distancia; estas unidades remotas son servicios que pueden desplegarse, cambiarse, sustituirse y escalarse independientemente unos de otros [21]. La granularidad de los microservicios se define principalmente, primero por su tamaño o dimensiones, es decir, el número de operaciones expuestas por el microservicio, junto con el número de microservicios que forman parte de la aplicación, y segundo por su complejidad y dependencias. El objetivo es tener un bajo acoplamiento, una baja complejidad y una alta cohesión entre los microservicios que forman la aplicación.

La calidad de un sistema basado en microservicios está influenciada por la granularidad de sus microservicios, porque su tamaño y número afectan directamente a los atributos de calidad del sistema. El tamaño o granularidad óptima de un microservicio afectan directamente al rendimiento de la aplicación, la capacidad de mantenimiento, el almacenamiento (transacciones y consultas distribuidas) y el uso y consumo de los recursos computacionales (principalmente en la nube, la plataforma habitual para desplegar y ejecutar microservicios). Aunque el tamaño del microservicio o la granularidad óptima es un tema de discusión, existen pocos patrones, métodos o modelos para determinar cuán pequeño debe ser un microservicio.

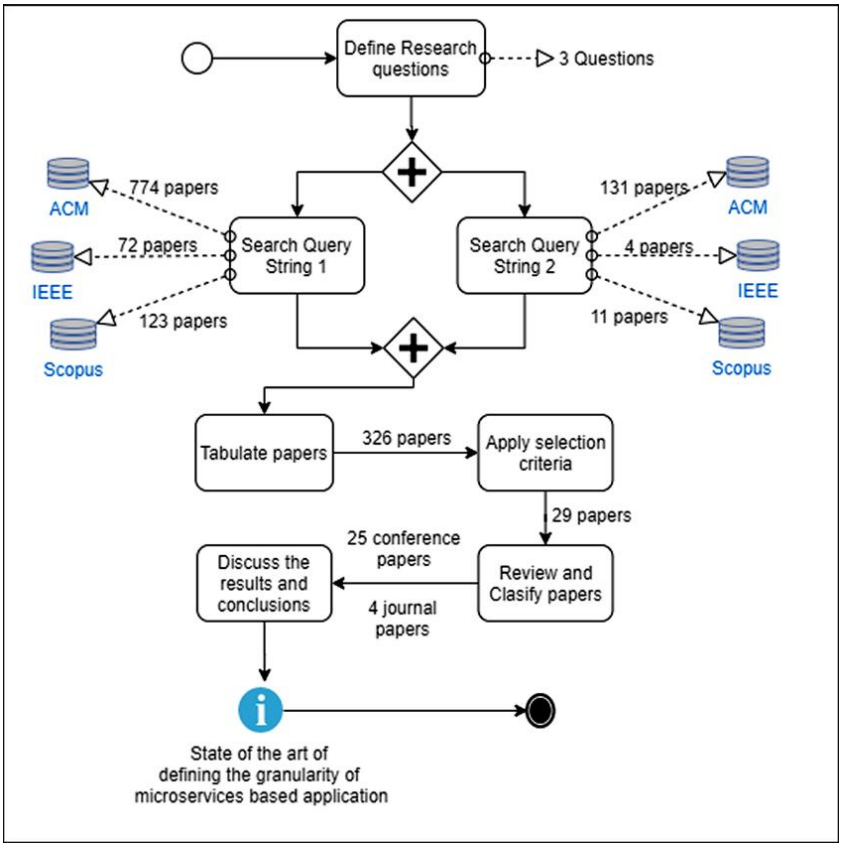


Figura 16. Proceso de revisión sistemática de literatura para definir el estado del arte de la definición de la granularidad de aplicaciones basadas en microservicios.

Se llevó a cabo una revisión sistemática de la literatura siguiendo el enfoque introducido por Kitchenham [43]. Se seleccionaron los artículos de investigación mediante dos cadenas de consulta utilizadas en el IEEE Xplore, la Biblioteca Digital de ACM y Scopus; luego los documentos se seleccionaron y revisaron, se aplicaron criterios de inclusión y exclusión, a continuación, se tabularon los artículos, se detalló su contribución, se clasificaron, se describieron las métricas y se especificaron los atributos de calidad usados en sus propuestas (ver figura 16).

Las preguntas de investigación (RQs) cubiertas en la revisión sistemática de la literatura fueron:

RQ1: ¿Qué enfoques se han propuesto para definir la granularidad de los microservicios y determinar su tamaño?
RQ2: ¿Qué métricas se están utilizando para definir y evaluar la granularidad de los microservicios?
RQ3: ¿Qué atributos de calidad se abordaron al investigar la granularidad de los microservicios?

Las cadenas de búsqueda (QS) usadas fueron:

QS1: ("micro service" OR microservice) AND (granularity OR size OR dimensioning OR decomposition).
QS2: ("micro service" OR microservice) AND "granularity" AND ("method" OR "technique" OR "methodology"), targeting only research papers.

Los criterios de inclusión y exclusión se pueden apreciar en la tabla 6.

Tabla 6. Criterios de inclusión y exclusión.

Criterios de inclusión	Criterios de exclusión
Artículos primarios de investigación que realicen una propuesta específica sobre el tamaño, granularidad o descomposición de aplicaciones a microservicios.	Artículos de tutoriales, ejemplos, experiencias o de opinión.
Artículos que proponen una metodología, método, modelo o técnica para definir la granularidad, el tamaño o dimensión de los microservicios.	Artículos de encuestas o revisiones de literatura.
Migraciones de monolito a microservicios que incluyan una metodología, método, modelo o técnica para definir la granularidad, el tamaño o dimensión de los microservicios.	Uso de microservicios en otras áreas.
Artículos publicados en revistas o conferencias en el campo de la arquitectura de software, ingeniería del software y/o ciencias de la computación.	Uso de microservicios como un tópico secundario.
Artículos presentados en conferencias, revistas o capítulos de libro.	Artículos que no proponen una metodología, método, modelo o técnica para definir la granularidad, el tamaño o dimensión de los microservicios.
	Artículos que proponen un método, metodología o técnica específica para soa, servicios web o móviles.
	Artículos publicados solo como resúmenes, blogs o presentaciones.
	Artículos que no sean escritos en inglés o español.

Para analizar los trabajos se definieron criterios de clasificación. Estos criterios se basaron en la clasificación realizada por [46], y han sido ampliamente utilizados en revisiones sistemáticas de literatura previas [47], [36], [6], [58]. Para responder a las preguntas de la investigación, se añadieron otros criterios de clasificación: métrica, etapa del proceso de desarrollo, técnica utilizada y atributos de calidad estudiados o analizados, estos se detallan a continuación:

- **Métrica utilizada:** ¿Qué métrica se utiliza para definir la granularidad de los microservicios?
- **Fases del proceso de desarrollo:** Fases del proceso de desarrollo en las que se centra el trabajo.

- **Estrategias de investigación:** Incluye la propuesta de solución, la investigación de validación, el documento de experiencia, el documento de opinión, el documento filosófico y la investigación de evaluación.
- **Enfoque:** Aspectos estructurales o de comportamiento propuestos en los trabajos para definir la granularidad de los microservicios [6].
- **Atributo de calidad estudiado:** Los atributos de calidad considerados en la propuesta, tales como el rendimiento, la disponibilidad, la fiabilidad, la escalabilidad, la mantenibilidad, la seguridad y la complejidad.
- **Contribución de la investigación:** Tipo de contribución realizada en el artículo; a saber, método, aplicación, formulación del problema, arquitectura de referencia, middleware, lenguaje arquitectónico, patrón de diseño, evaluación o comparación.
- **Tipo de experimentación:** Tipo de experimentación utilizado para validar la propuesta; a saber, experimento, estudio de caso, teórico u otro.
- **Técnica utilizada:** Este criterio describe la técnica, método o modelo utilizado para definir la granularidad de los microservicios.
- **Datos de entrada:** Tipo de datos de entrada utilizados para identificar los microservicios (es decir, casos de uso, registros, código fuente, trazas de ejecución, entre otros)
- **Tipo de caso de estudio utilizado:** Este criterio determina si el estudio de caso es un ejemplo académico (caso hipotético) o un estudio de caso de la vida real.
- **Nivel de automatización:** Este criterio determina el nivel de automatización de la técnica o método propuesto, si es manual, automático o semiautomático.

El proceso de búsqueda se realizó en julio de 2020. La búsqueda en las bases de datos de publicaciones científicas al aplicar las cadenas de búsqueda (QS1 y QS2) relacionadas con la granularidad de los microservicios arroja 969 y 146 trabajos, respectivamente. Tras aplicar los criterios de inclusión y exclusión, se seleccionaron 29 trabajos que abordan la definición de la granularidad de los microservicios. (Ver tabla 7 y tabla 8). Los resultados resumidos de esta revisión sistemática de la literatura se sintetizan en la infografía presentada en la figura 17.

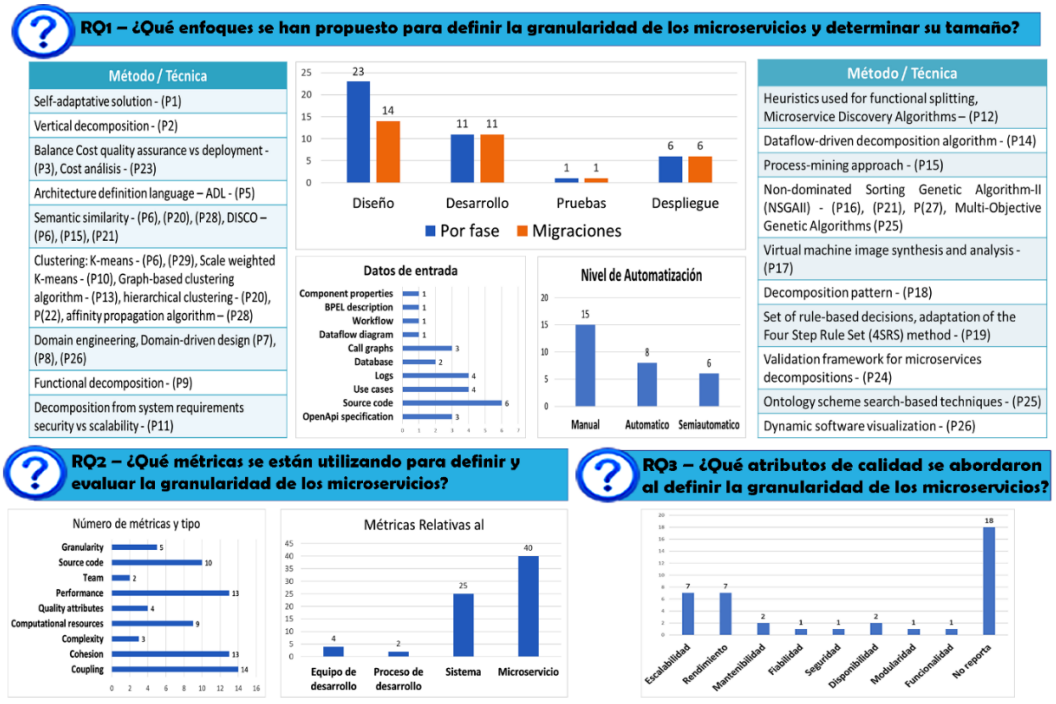


Figura 17. Infografía resumen de los enfoques para definir la granularidad de los microservicios

Tabla 7. Trabajos relacionados con el problema de la granularidad del microservicio.

Año	Artículos	Métricas	Atributos de calidad	Técnica o método usado.	Datos de entrada
2020	Microservice Backlog – Nuestra propuesta	Complejidad, acoplamiento, cohesión, granularidad, rendimiento	Modularidad. Mantenibilidad Funcionalidad. Rendimiento.	- Algoritmo genético, algoritmo de agrupamiento, similitud semántica, procesamiento de lenguaje natural (NPL), algoritmo calculador de métricas.	Historias de usuario.
2020	2 [59], [60].	Cohesión, granularidad.	Ninguno	- Diseño basado en el dominio, diseño arquitectónico vía visualización dinámica de software. - Clustering usando un algoritmo de propagación de afinidad. - Clustering con similitud semántica.	- Código fuente - Logs de ejecución.
2019	12 [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72].	Acoplamiento, cohesión, granularidad, uso de recursos computacionales, rendimiento, código fuente.	Escalabilidad. Rendimiento. Funcionalidad. Modularidad. Mantenibilidad.	- Machine learning, scale weighted k-means. - Algoritmo de descomposición basado en el flujo de datos. - El enfoque de minería de procesos, DISCO utilizado para identificar los procesos de negocio. - Algoritmo de agrupación de átomos funcional basado en búsquedas. - Algoritmo genético de clasificación no dominado II (NSGA II). - Conjunto de decisiones basadas en reglas, adaptación del método del conjunto de reglas de cuatro pasos (4SRS). - Incorporación de palabras y agrupación jerárquica de similitudes semánticas. - Algoritmos de descubrimiento de microservicios. - Algoritmo de agrupación aplicado a entidades de dominio agregadas. - Método basado en el análisis de costos de granularidad de servicios, función de análisis de costos. - Marco de validación para las descomposiciones de microservicio. - Técnicas de búsqueda de esquemas ontológicos, algoritmo genético multiobjetivo.	- Logs de acceso. - Diagrama de flujo de datos. - Casos de uso. - Trazas de ejecución en los logs. - Especificación OpenApi - La base de datos. - Grafo de llamadas de ejecución - Propiedades de los componentes y de los microservicios.
2018	6 [73], [74], [75], [76], [77], [78].	Acoplamiento, cohesión, complejidad, granularidad, recursos computacionales rendimiento.	Escalabilidad. Rendimiento. Disponibilidad.	- Ingeniería de dominio, diseño basado en el dominio. - Diseño dirigido por el dominio, puntos de función COSMIC. - Descomposición funcional. - Heurística usada para la división funcional. - Algoritmos de descubrimiento de microservicios. - Patrón de descomposición.	- Casos de uso - Código Fuente. Base de datos. – Grafos de llamadas de ejecución - Flujo de trabajo, descripción de BPEL.
2017	7 [79], [80], [81], [82], [83], [84], [85]	Rendimiento.	Escalabilidad. Rendimiento. Fiabilidad. Mantenibilidad	- Descomposición vertical en sistemas autónomos. - Equilibrar costos de aseguramiento de la calidad con los costos del despliegue. - Comparar los mismos microservicios en un solo contenedor y en dos contenedores. - Lenguaje de definición de arquitectura (ADL). - Similitud semántica, clustering k-means, DISCO. - Algoritmo de clustering basado en grafos. - Síntesis y análisis de imágenes de máquinas virtuales.	- Especificiación OpenApi - Código Fuente.
2016	2 [86], [87].	Acoplamiento, Impacto en la seguridad y en la escalabilidad.	Escalabilidad. Seguridad.	Solución auto adaptativa. Descomposición de los requisitos del sistema - seguridad vs. Escalabilidad	Ninguno

En RQ1, identificamos los trabajos que proponen un método, modelo o metodología para definir la granularidad de los microservicios; las métricas son fundamentales porque permiten medir, monitorear y evaluar cualquier aspecto de un microservicio, definiendo o determinando así la granularidad apropiada de un microservicio.

En RQ2, identificamos las métricas utilizadas para evaluar la granularidad del microservicio y su descomposición. La figura 17 muestra el tipo y número de métricas y si se aplicó en el

microservicio, sistema, proceso de desarrollo o equipo de desarrollo; se muestran el tipo de métricas utilizadas. Estas métricas se detallan en la sección 3.4.

Finalmente, para la RQ3 sintetizamos los trabajos que abordan los atributos de calidad para evaluar la granularidad de los microservicios.

2.2.1. Clasificación de los enfoques para definir la granularidad de los microservicios.

La mayoría de los trabajos se publicaron en conferencias (86%), y sólo cuatro (14%) se publicaron en revistas. Todos los trabajos seleccionados se publicaron entre 2016 y principios de 2020 (2 en 2016, 7 en 2017, 6 en 2018, 12 en 2019 y 2 en 2020). Ver tabla 8.

Tabla 8. Artículos seleccionados que abordan el problema de la granularidad de los microservicios.

ID.	Artículo	Año	Tipo
P1	Microservices and Their Design Trade-offs: A self-adaptive Roadmap [86] .	2016	Artículo de Conferencia
P2	Microservice Architectures for Scalability, Agility and Reliability in E-Commerce [79].	2017	Artículo de Conferencia
P3	From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture [80].	2017	Artículo de Conferencia
P4	Microservices: Granularity vs. Performance [81].	2017	Artículo de Conferencia
P5	Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity [82].	2017	Artículo de Conferencia
P6	Microservices Identification Through Interface Analysis [83].	2017	Artículo de Conferencia
P7	Partitioning Microservices: A domain engineering approach [73].	2018	Artículo de Conferencia
P8	A Case Study on Measuring the Size of Microservices [74]	2018	Artículo de Conferencia
P9	Identifying Microservices Using Functional Decomposition [75].	2018	Artículo de Conferencia
P10	Unsupervised Learning Approach for Web Application Auto-decomposition into Microservices [62].	2019	Artículo de revista
P11	Requirements Reconciliation for Scalable and Secure Microservice (De)composition [87].	2016	Artículo de Conferencia
P12	Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems [76].	2018	Artículo de Conferencia
P13	Extraction of Microservices from Monolithic Software Architectures [84].	2017	Artículo de Conferencia
P14	From Monolith to Microservices: A Dataflow-Driven Approach [88]. A Dataflow-driven Approach to Identifying Microservices from Monolithic Applications [63].	2017 2019	Artículo de Conferencia Artículo de revista
P15	From Monolithic Systems to Microservices: A Decomposition Framework Based on Process Mining [64].	2019	Artículo de Conferencia
P16	Service Candidate Identification from Monolithic Systems Based on Execution Traces [61].	2019	Artículo de revista
P17	The ENTICE Approach to Decompose Monolithic Services into Microservices [89]. Towards a Methodology to Form Microservices from Monolithic Ones [85].	2016 2017	Artículo de Conferencia
P18	Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern [77].	2018	Artículo de Conferencia
P19	A logical architecture design method for microservices architectures [65].	2019	Artículo de Conferencia
P20	A New Decomposition Method for Designing Microservices [66].	2019	Journal paper
P21	Business Object Centric Microservices Patterns [67].	2019	Artículo de Conferencia
P22	From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [68].	2019	Conference paper
P23	Granularity Cost Analysis for Function Block as a Service [69].	2019	Artículo de Conferencia
P24	MicroValid: A Validation Framework for Automatically Decomposed Microservices [70].	2019	Artículo de Conferencia
P25	Migration of Software Components to Microservices: Matching and Synthesis [71].	2019	Artículo de Conferencia
P26	Microservice Decomposition via Static and Dynamic Analysis of the Monolith [59].	2020	Artículo de Conferencia
P27	Towards Automated Microservices Extraction Using Multi-objective Evolutionary Search [72].	2019	Artículo de Conferencia
P28	Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach [60].	2020	Artículo de Conferencia
P29	Migrating Web Applications from Monolithic Structure to Microservices Architecture [78].	2018	Artículo de Conferencia

En la figura 17 se muestran las fases del proceso de desarrollo abordadas por cada propuesta. En varios documentos se hace énfasis en más de una fase (por ejemplo, [62] se centra en el desarrollo y el despliegue, corresponde a un método de migración del monolito a microservicios). La mayoría de los métodos propuestos se centran en la fase de diseño (79%) y desarrollo (38%), y sólo uno aborda las pruebas (3%). Las migraciones de arquitecturas monolíticas a microservicios fueron muy comunes e importantes 19 de 29 trabajos (66%). Los artículos que no abordan la migración se centran en la identificación de microservicios en la fase de diseño; por lo

tanto, la definición del tamaño y la granularidad de los microservicios a partir de la fase de diseño es fundamental, porque tiene consecuencias para el desarrollo, las pruebas y el despliegue.

La migración de los sistemas informáticos implica muchas decisiones arquitectónicas que deben ser evaluadas sistemáticamente para valorar las acciones tomadas y los riesgos concretos [90]. En estos casos, se parte de un sistema monolítico que debe descomponerse en microservicios, y ese sistema monolítico tiene importantes fuentes de datos que permiten identificar y evaluar los microservicios candidatos. Estas fuentes son principalmente: el código fuente, los casos de uso, la base de datos, los registros (logs) y las trazas de ejecución. Cabe señalar que el desarrollo de aplicaciones basadas en microservicios está estrechamente relacionado con las prácticas ágiles y DevOps, ninguno de los datos de entrada que se están considerando en los métodos propuestos, corresponden a prácticas ágiles como historias de usuarios, planificación del producto, planificación de iteraciones, entre otros. Por lo tanto, se requiere más investigación en estos tópicos.

Además, la mayoría de los trabajos (79%) se centran en la fase de diseño; implícita o explícitamente, sugieren que es fundamental definir la granularidad "correcta" de los microservicios a partir de la fase de diseño. Sin embargo, algunos autores afirman que razonar sobre el tamaño y el rendimiento de los microservicios no es posible en tiempo de diseño; de hecho, Hassan y otros [82] afirman que el comportamiento esperado del sistema no puede captarse plenamente en tiempo de diseño.

En cuanto a la estrategia de investigación (ver figura 18), la investigación de validación y las propuestas de solución son las más abordadas, representan casi todos los artículos (14 y 11, respectivamente); las propuestas que se han ensayado y validado en la práctica, en la industria, son muy pocas (Investigación de evaluación, sólo 2 artículos).

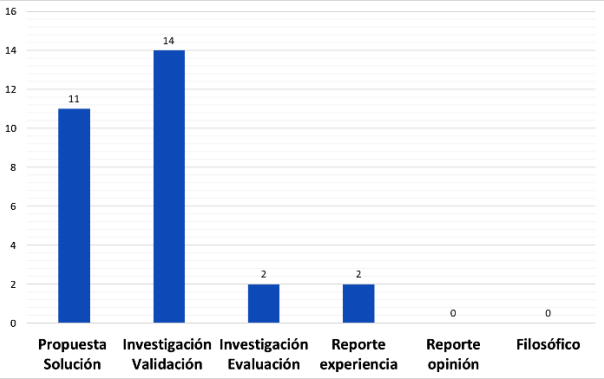


Figura 18. Número de trabajos por estrategia de investigación.

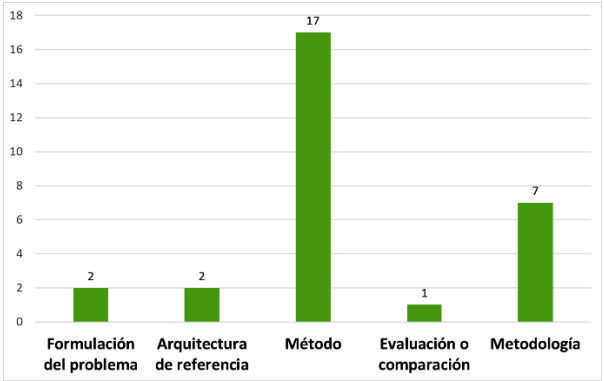


Figura 19. Número de documentos por tipo de contribución

En cuanto al tipo de contribución (véase figura 19), la gran mayoría de los artículos proponen métodos 17 artículos (59%), algunos proponen metodologías 7 artículos (24%), pocos proponen arquitecturas de referencia 2 artículos (7%) y formulación de problemas (7%), sólo uno propone una evaluación o comparación (3%). En cuanto al enfoque de validación (véase la figura 20), la mayoría de los documentos 20 (69%) utilizaron casos de estudio para la validación y la evaluación, otros documentos utilizaron experimentos 11 artículos (37%), la mayoría de los que usaron experimentos también utilizaron casos de estudio.

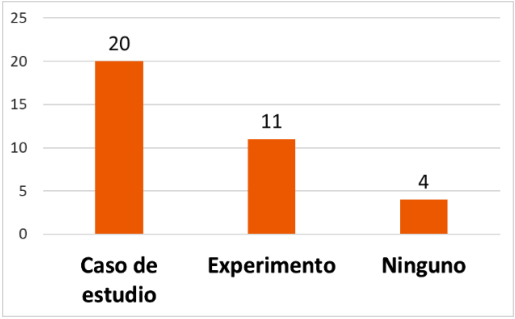


Figura 20. Número de documentos por enfoque de validación

Más de la mitad de los estudios (13 de 29) validaron sus propuestas utilizando casos de estudio hipotéticos o de uso académico, y la casi mitad restante (14 de 29) utilizaron casos de estudio de la vida real, logrando así una mejor validación. Aún mejor, algunos estudios (8 de 29) utilizaron proyectos reales de código abierto. Hay dificultad para encontrar datos de proyectos reales o ficticios tabulados y disponibles para hacer pruebas de los métodos y técnicas de inteligencia artificial o de cualquier otra para mejorar la arquitectura de microservicios. Algunos autores han hecho un esfuerzo para lograr este objetivo, por ejemplo, Rahman y otros [91] compartieron un conjunto de datos compuesto por 20 proyectos de código abierto que utilizaban patrones específicos de arquitectura de microservicios; Márquez y Astudillo [92] compartieron un conjunto de datos de proyectos basados en microservicios de código abierto cuando investigaron el uso real de patrones arquitectónicos. La mayoría de los proyectos no tienen disponibles las historias de usuario o especificación de requerimientos, ni el detalle de los microservicios identificados e implementados. En el artículo de revisión de literatura resultado de este trabajo se listan estos proyectos.

Los casos de estudio más utilizados para validar las propuestas fueron los “Kanban Board” y “Money Transfer”, que fueron utilizados por 3 trabajos, seguidos por “JPetsStore” y “Cargo Tracking” los cuales fueron utilizados por 2 trabajos.

2.2.2. Principales contribuciones y vacíos de investigación en la definición de la granularidad de los microservicios.

La granularidad de los microservicios implica definir su tamaño y el número que formará parte de la aplicación. A partir de la propuesta de S. Newman [1], los microservicios siguen el principio de la responsabilidad simple que dice "Reúne las cosas que cambian por la misma razón y separa las cosas que cambian por razones diferentes". El tamaño, la dimensión o la granularidad de los microservicios se han definido tradicionalmente de la siguiente manera:

1. Prueba y error, dependiendo de la experiencia del arquitecto o del desarrollador.
2. Según el número de líneas de código.
3. Por unidades de implementación
4. Por la capacidad de negocio.
5. Por las capacidades del equipo o equipos de desarrollo.
6. Usando el diseño dirigido por el dominio.

Richardson [93] propuso cuatro patrones de descomposición, que permiten dividir una aplicación en servicios: 1) Descomponer por capacidad empresarial: definir servicios correspondientes a las capacidades empresariales; 2) descomponer por subdominio: definir servicios correspondientes a los subdominios DDD; 3) servicio autónomo: diseñar servicios para gestionar peticiones sincrónicas sin esperar a que otros servicios respondan. (4) servicio por equipo: cada servicio es

propiedad de un equipo, que tiene la responsabilidad exclusiva de realizar los cambios, e idealmente cada equipo tiene un solo servicio. Zimmerman y otros [94] propusieron patrones para los microservicios (MAP) para el diseño y la evolución de los microservicios. Los patrones se dividen en cinco categorías: (1) fundamento, (2) responsabilidad, (3) estructura, (4) calidad, y (5) evolución. Estos patrones son una referencia importante para el desarrollo de aplicaciones basadas en microservicios. No hay ningún patrón específico que ayude a determinar el número de microservicios y su tamaño, es decir, el número de operaciones que debe contener.

Tabla 9. artículos por datos de entrada, tipo de contribución y nivel de automatización.

	Datos de entradas	**_Tipo de Contribución**					**Nivel de Automatización**		
		Método	**Metodología**	**Ref. Arquitec.**	**Problema**	**Evalua.**	**Manual**	**Automático**	**Semi Auto.**
MB	Historias de usuario	X							X
P1					X		X		
P2				X			X		
P3		X					X		
P4						X	X		
P5				X			X		
P6	Especificación OpenApi.	X						X	
P7		X					X		
P8		X					X		
P9	Casos de uso.	X					X		
P10	Logs de acceso.	X						X	
P11			X				X		
P12	Código Fuente Base de datos, grafo de llamadas de ejecución.	X						X	
P13	Código fuente.	X						X	
P14	Diagrama de flujo de datos, casos de uso.		X						X
P15	Logs de ejecución.	X							X
P16	Trazas de ejecución desde los logs.	X						X	
P17			X				X		
P18	Declaraciones de escenario, flujo de trabajo, descripción de BPEL.		X				X		
P19	Modelos de casos de uso UML.	X					X		
P20	Especificación OpenApi.	X						X	
P21	Código Fuente Base de datos, grafo de llamadas de ejecución.	X							X
P22	Código fuente, estilo arquitectural MVC, grafo de llamadas.		X						X
P23					X		X		
P24		X					X		
P25	Propiedades de componentes y microservicios		X						X
P26	Código fuente.		X				X		
P27	Especificación OpenAPI	X						X	
P28		X						X	
P29	Código fuente, logs de ejecución.	X							X

MB: Microservices Backlog. Ref. Arquitec: Arquitectura de referencia. Evalua: Evaluación. Semi auto: Semi automático.

El tamaño del microservicio o la granularidad óptima es una de las propiedades más discutidas y hay pocos patrones, métodos o modelos para determinar cuán pequeño debe ser un microservicio. A este respecto, algunos autores han abordado este problema y han propuesto las soluciones que se resumen en la tabla 9.

Las técnicas propuestas se clasificaron en manuales, semiautomáticas o automáticas; las técnicas manuales son métodos, procedimientos o metodologías realizadas por el arquitecto o desarrollador descomponiendo los sistemas siguiendo una serie de pasos. Las técnicas automáticas utilizan algún tipo de algoritmo para generar la descomposición, y el sistema genera la descomposición. Las semiautomáticas combinan una parte hecha manualmente y otra hecha automáticamente. La mayoría de los documentos proponían procedimientos manuales para identificar los microservicios (15 documentos); algunas propuestas eran automáticas (8 documentos) y pocas propuestas eran semiautomáticas (6 documentos). Los estudios de casos más utilizados para validar las propuestas fueron “Kanban Board” y “Money Transfer” (P6, P20, P28).

Los documentos de 2017 y 2018 son en su mayoría métodos o metodologías manuales que detallan la forma de descomponer o determinar los microservicios, utilizando diseño impulsado por el dominio (DDD), ingeniería de dominio o una metodología específica. Más adelante, los documentos de 2019 y 2020 proponen métodos semiautomáticos y automáticos que utilizan algoritmos inteligentes y aprendizaje automático, centrados principalmente en las migraciones del monolito a los microservicios. Podemos observar una evolución cronológica en las propuestas, el tipo de técnicas utilizadas para definir la granularidad de los microservicios que forman parte de una aplicación se presentan en la figura 21, la similitud semántica, el aprendizaje automático y la programación genética fueron las técnicas más importantes.

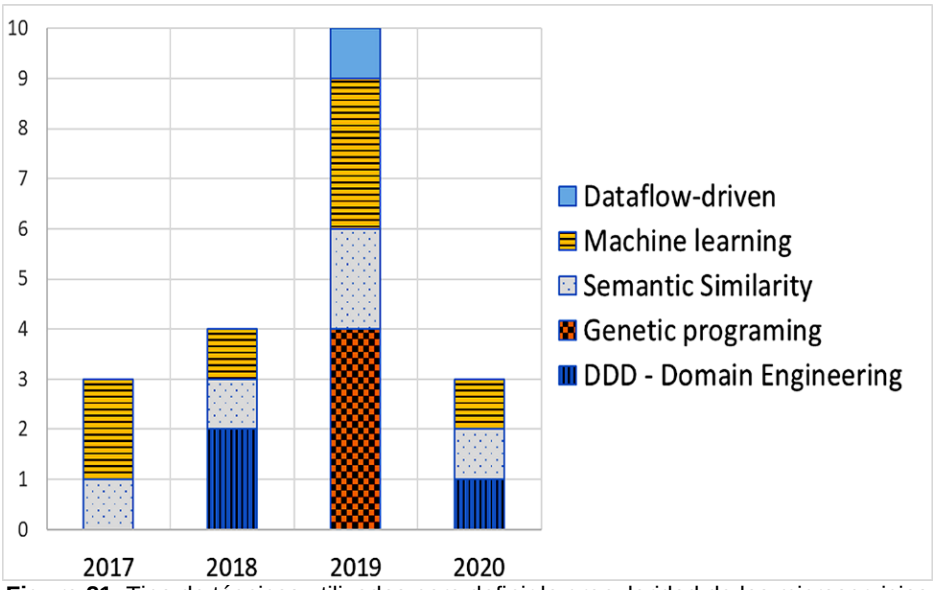


Figura 21. Tipo de técnicas utilizadas para definir la granularidad de los microservicios

El uso de técnicas de inteligencia artificial para determinar la granularidad apropiada o para identificar los microservicios que formarán parte de una aplicación es una tendencia creciente; esto es especialmente cierto en el caso de los algoritmos de aprendizaje automático no supervisado (clustering) y los algoritmos genéticos, haciendo énfasis en la similitud semántica para agrupar los microservicios que se refieren a la misma entidad. La ingeniería de dominio y el DDD siguen siendo algunas de las técnicas más utilizadas.

La migración de monolito a microservicios es un tema de gran interés y ampliamente estudiado. Por el contrario, se encontraron pocas propuestas para que aborden el problema de la granularidad en el diseño y desarrollo de aplicaciones basadas en microservicios desde cero. Los métodos propuestos hacen énfasis en artefactos disponibles en tiempo de ejecución, desarrollo, despliegue o producción, que apenas están disponibles cuando se inicia un proyecto o en tiempo de diseño.

El desarrollo de aplicaciones basadas en microservicios desde cero se asemeja al desarrollo basado en componentes [95], en el que los microservicios son componentes de software reutilizables. En [96] caracterizamos el proceso de desarrollo de aplicaciones basadas en microservicios, identificando dos partes fundamentales, primero el desarrollo de cada microservicio y luego el desarrollo de aplicaciones basadas en esos microservicios.

La definición de una granularidad adecuada es fundamental para el desarrollo de aplicaciones basadas en microservicios [96]. La granularidad de un monolito no es la óptima y la definición de una operación por microservicio tampoco es óptima. Por lo tanto, si una aplicación que ofrece 100 operaciones tuviera 100 microservicios, tampoco sería óptima, debido a la latencia, el rendimiento y la gestión de este gran sistema distribuido. La granularidad óptima se encuentra en algún punto entre la aplicación monolítica y el sistema de operación por microservicio, esta granularidad debería definirse de acuerdo con las características de la aplicación, el equipo de desarrollo, los requisitos no funcionales, los recursos disponibles y las compensaciones (trade-offs) de diseño, desarrollo y operación.

Los vacíos de la investigación se centran en proponer técnicas o métodos que permitan evaluar la granularidad y su impacto en las pruebas, considerando los controles de seguridad, los mecanismos de tolerancia a fallos y DevOps. Al gestionar más microservicios o microservicios más grandes, las pruebas pueden ser más lentas y tediosas. Además, las tuberías de integración y despliegue continuos serían más complejos. Determinar el número apropiado de microservicios y su impacto en el despliegue continuo es un tema de investigación interesante. Pocos trabajos abordan estas cuestiones.

Además, pocos o ninguno de los trabajos utilizan como datos de entrada o unidades de análisis las prácticas utilizadas en el desarrollo ágil, tales como historias de usuarios, planificación del producto, planificación del lanzamiento y sus datos, para proponer métodos ágiles o nuevas prácticas que permitan determinar o evaluar los microservicios que formarán parte de la aplicación. Ninguno de los trabajos propuestos se centra en el desarrollo ágil de software.

Se han propuesto varios trabajos interesantes, pero todavía hay pocas propuestas concretas y aplicables; se necesita más investigación para proponer patrones de diseño, buenas prácticas, modelos más completos, métodos o herramientas que puedan generalizarse para definir la granularidad de los microservicios teniendo en cuenta métricas, atributos de calidad y las compensaciones (trade-offs).

2.2.3. Atributos de calidad evaluados para definir la granularidad de los microservicios.

Los atributos de calidad son esenciales para las aplicaciones actuales. La disponibilidad, el rendimiento, la escalabilidad, seguridad, mantenibilidad, y la tolerancia a fallos son características esenciales que toda aplicación debe manejar. Una arquitectura basada en microservicios permite una gestión independiente de los atributos de calidad, según la necesidad específica de cada microservicio. Esta es una de las principales ventajas en comparación con las arquitecturas monolíticas.

El tamaño y el número de microservicios que componen una aplicación afectan directamente a sus atributos de calidad. La creación de más microservicios puede afectar a la mantenibilidad porque los costes de las pruebas aumentarán, más aún si no se dispone de pruebas automatizadas. Por otra parte, el rendimiento también puede verse afectado por tener que integrar y procesar datos de varias aplicaciones distribuidas. Es evidente que los atributos de calidad se ven afectados por la granularidad de los microservicios, por lo que deben tenerse en cuenta al definir un modelo, método o técnica para determinar la granularidad (véase la Figura 22).

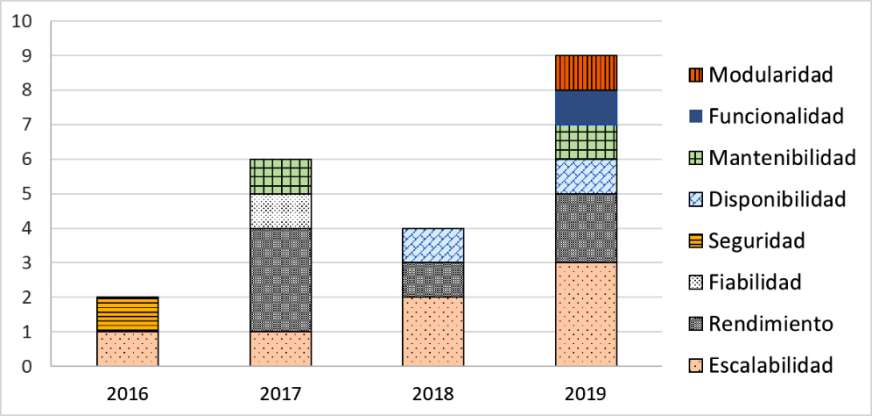


Figura 22. Número artículos y atributos de calidad usados para definir la granularidad de los microservicios

Sorprendentemente, el 62% de las propuestas identificadas no consideraron ni informaron de ningún atributo de calidad. De las que sí lo hicieron, la escalabilidad y el rendimiento fueron las más consideradas (7 documentos, 24%), seguidas de la mantenibilidad y la disponibilidad (2 documentos, 7%); por último, la fiabilidad (tolerancia a fallos), la seguridad, la funcionalidad y la modularidad con un solo documento cada una. Se necesita más investigación que considere los atributos de calidad para definir la granularidad de los microservicios que componen una aplicación. La seguridad y la tolerancia a los fallos son atributos clave que las aplicaciones basadas en microservicios deben manejar, pocos trabajos abordaron estas características (véase tabla 10).

Tabla 10. Artículos por atributos de calidad

Artículo	Características en tiempo de ejecución						Características del Software como artefacto	
	Escalab.	Rend.	Fiab.	Disp.	Seguridad	Funct.	Mant.	Modul.
MB		X				X	X	X
P2	X	X	X					
P3							X	
P4		X						
P10	X	X						
P11	X				X			
P12	X	X		X				
P13		X						
P16	X					X	X	X
P21	X	X		X				
P29	X	X						

Abreviaturas. MB: Microservices Backlog, Escalab: Escalabilidad, Rend: Rendimiento, Fiab: Fiabilidad, Disp: Disponibilidad, Funct: funcionalidad, Main: Mantenibilidad, Modul: Modularidad.

Se agruparon los atributos de calidad del software en las dos categorías: en primer lugar, según la característica en tiempo de ejecución, (escalabilidad, rendimiento, fiabilidad, disponibilidad y funcionalidad), que son observables durante la ejecución; y en segundo lugar, según el software como característica de artefacto (mantenibilidad, modularidad, reutilización), que no son

observables durante la ejecución [97] [98]; las características en tiempo de ejecución fueron las más utilizadas, 8 documentos (P2, P4, P10, P11, P12, P13, P21 y P29); sólo dos documentos abordaron las características del software como artefacto (P3 y P16); sólo un documento utilizó tanto las características del artefacto como las del tiempo de ejecución (P16). Por consiguiente, se necesitan más propuestas para definir la granularidad de los microservicios teniendo en cuenta tanto las características observables en tiempo de ejecución como las del software como artefacto.

Características en tiempo de ejecución.

En esta sección se detallan los atributos de calidad observables en tiempo de ejecución y la forma en que fueron abordados por los trabajos relacionados.

- La escalabilidad, el rendimiento y la fiabilidad (tolerancia a los fallos) fueron utilizados por un solo documento (P2). P2 propuso una reimplementación de otoo.de (un estudio de caso de la vida real), definieron la granularidad a través de la descomposición vertical, utilizaron DevOps incluyendo el despliegue continuo, para entregar las funcionalidades rápidamente a los clientes. La organización del equipo es crucial para el éxito, esta organización se basó en la Ley de Conway. La automatización completa de la garantía de calidad y el despliegue de software permite la detección temprana de fallos y errores, reduciendo así los tiempos de reparación tanto durante el desarrollo como durante las operaciones [79]. En este artículo no se propusieron métricas para la evaluación.
- La escalabilidad y el rendimiento se utilizaron en dos documentos (P10, P29); el P10 propuso un método de descomposición automática, que se basaba en un enfoque de caja negra que extrae los registros de acceso a la aplicación utilizando un método de agrupación para descubrir las particiones de URL que tienen un rendimiento y unos requisitos de recursos similares. Esas particiones se asignaron a microservicios [87]. Las métricas utilizadas en este documento fueron las métricas de rendimiento (violaciones de SLO en tiempo de respuesta, número de llamadas, número de solicitudes rechazadas y rendimiento) y las métricas de recursos computacionales (promedio de CPU, número de máquinas virtuales utilizadas y máquinas virtuales asignadas). P29 utilizó el código fuente y los registros de tiempo de ejecución en un método semiautomático, utilizó la granularidad, el rendimiento y la métrica del código fuente para evaluar las descomposiciones. Presentó un método basado en el análisis de programas para migrar las aplicaciones heredadas del monolito a la arquitectura de microservicios; este método utilizaba un gráfico de llamada de funciones, un modelo de cadena de Markov para representar las características de la migración y un algoritmo de agrupación jerárquica K-means [78].
- Dos trabajos (P4, P13) utilizaron únicamente el rendimiento. P4 examinó el problema de la granularidad del microservicio y exploró su efecto en la latencia de la aplicación. Se simularon dos enfoques para el despliegue de microservicios; el primero con microservicios en un solo contenedor, y el segundo con microservicios divididos en contenedores separados. Se examinaron las conclusiones en el contexto de las arquitecturas de aplicación de la Internet de las Cosas (IoT) [81]; este documento corresponde a una evaluación o comparación, no es un método para definir la granularidad del microservicio; se utilizaron métricas de rendimiento (tiempo de respuesta y número de llamadas). P13 presentó tres estrategias de acoplamiento formales e incorporó las mismas en un algoritmo de agrupación basado en grafos: 1) Acoplamiento lógico, 2) Acoplamiento semántico y 3) Acoplamiento de contribuyentes. Las estrategias de acoplamiento se basan en la metainformación de bases de código monolíticas para construir una representación gráfica de los monolitos que a su vez son procesados por el algoritmo de agrupamiento para generar recomendaciones para posibles candidatos a microservicios en un escenario de refactorización; P13 fue el único que propuso métricas basadas en el equipo de desarrollo; el acoplamiento lógico, la redundancia de dominio media,

el acoplamiento de contribuyente, el acoplamiento semántico, el recuento de cambios en el repositorio de código (commit count), el recuento de contribuyentes y las líneas de código fueron las métricas utilizadas en este documento.

- La escalabilidad y la seguridad fueron utilizadas por un documento (P11). P11 propuso una metodología, que consiste en una serie de pasos y actividades que deben llevarse a cabo para identificar los microservicios que formarán parte del sistema. Se basa en los casos de uso y en el análisis realizado por el arquitecto en cuanto a la escalabilidad y la seguridad de cada caso de uso, así como las dependencias con los demás casos de uso [87]; en este documento se utilizaron las siguientes métricas: peso de la dependencia, impacto de la seguridad e impacto de la escalabilidad (métrica cualitativa).
- La escalabilidad, el rendimiento y la disponibilidad se abordaron en dos documentos (P12, P21). En el P12 se presentaron técnicas de descubrimiento que ayudan a identificar las partes adecuadas de los sistemas comerciales orientados al consumidor que podrían rediseñarse como microservicios con características deseadas como alta cohesión, bajo acoplamiento, alta escalabilidad, alta disponibilidad y alta eficiencia de procesamiento [82]. Propusieron algoritmos de descubrimiento de microservicios y heurísticas. Se trataba de un método automático que utiliza el acoplamiento (acoplamiento estructural), la cohesión (falta de cohesión), los recursos computacionales (promedio de memoria, promedio de disco) y las métricas de rendimiento (número de solicitudes, tiempo de ejecución). El P21 corresponde a un método semiautomático, un algoritmo genético con similitudes semánticas basado en DISCO y el algoritmo genético de clasificación no dominado -II (NSGAII). En este documento se presentaron cuatro patrones de microservicios, a saber, asociación de objetos, contención exclusiva, contención inclusiva y subtipificación para el desarrollo de software desde cero (''greenfield''), al tiempo que se demostraba el valor de los patrones para los desarrollos en evolución (''brownfield'') mediante la identificación de posibles microservicios [67]. Las métricas utilizadas en este documento fueron: el acoplamiento estructural, la falta de cohesión, el promedio de la CPU, el promedio de la red, el número de ejecuciones y el número de paquetes enviados.

Características del Software como un artefacto

Sólo se utilizaron dos características de software como artefacto, que fueron la mantenibilidad y la modularidad. Sólo se utilizó la mantenibilidad en un documento (P3); mientras que la la mantenibilidad y la modularidad fueron utilizadas por (P16), P16 fue el método más completo (también uso escalabilidad y funcionalidad) en cuanto al uso de atributos de calidad.

P3 utilizó un equilibrio entre el costo de aseguramiento de calidad y el costo de despliegue para definir la granularidad de los microservicios, fue un método manual. La elección de la granularidad debe basarse en el equilibrio entre los costos de aseguramiento de calidad y el costo del despliegue [80].

El P16 presentó un marco que consiste en tres pasos principales: 1) la extracción de trazas de ejecución representativas, 2) la identificación de entidades mediante un algoritmo de agrupación de átomos funcionales basado en la búsqueda, y 3) la identificación de interfaces para servicios candidatos [61]. También presentaron un sistema de medición integral para evaluar cuantitativamente la calidad de los servicios candidatos en términos de funcionalidad, modularidad y capacidad de evolución. P16 fue un método automático, este método también utilizó un algoritmo genético de clasificación no dominado - II (NSGA II), las métricas de evaluación fueron el acoplamiento (número de interfaces de integración), la cohesión a nivel de mensaje, la cohesión a nivel de dominio, la calidad de la modularidad estructural, la calidad de la

modularidad conceptual, la frecuencia de intercambio interno (ICF), la frecuencia de intercambio externo (ECF), y la proporción de ECF a ICF.

Atributos de calidad e inteligencia artificial

En algunos casos, se están utilizando técnicas de inteligencia artificial para mejorar los atributos de calidad de los microservicios. Por ejemplo:

- Alipour y Liy, [99] propusieron dos algoritmos de aprendizaje automático y predijeron la demanda de recursos de los sistemas de fondo de microservicios, tal como lo emula una aplicación de referencia de carga de trabajo de Netflix, propusieron una arquitectura de microservicios que encapsula las funciones de vigilancia de las métricas y el aprendizaje de los patrones de carga de trabajo. Luego, esta arquitectura de servicios se utiliza para predecir la futura carga de trabajo para la toma de decisiones sobre el aprovisionamiento de recursos.
- Prachitmutita y otros [105] propusieron un nuevo marco de autoescalado basado en la carga de trabajo prevista, con una red neuronal artificial, una red neuronal recurrente y un algoritmo de optimización del escalado de recursos utilizado para crear un sistema automatizado para gestionar toda la aplicación con la Infraestructura como servicio (IaaS).
- Ma y otros [100] propusieron un enfoque, denominado recuperación de microservicios basada en escenarios (SMSR), para recomendar microservicios apropiados para los usuarios basados en los escenarios de prueba, Desarrollo Basado en el Comportamiento (BDD) escritos por el usuario. El algoritmo de recuperación de servicios propuesto se basa en word2vec, un método de aprendizaje automático ampliamente utilizado en el procesamiento del lenguaje natural (NPL), para realizar el filtrado de servicios y calcular la similitud de estos.
- Abdullah y otros [62] propusieron un sistema automatizado completo para descomponer una aplicación en microservicios, implementar microservicios utilizando los recursos apropiados y escalar automáticamente los microservicios para mantener el tiempo de respuesta deseado.

La inteligencia artificial puede ayudar a mejorar y controlar diferentes características de los microservicios, especialmente las relacionadas con la mejora de los atributos de calidad. Se han formulado algunas propuestas a este respecto, pero es necesario seguir investigando.

Por último, identificamos los métodos automáticos y semiautomáticos, que utilizaron métricas y abordaron algún atributo de calidad para definir la granularidad (ver tabla 11). Sólo seis artículos cumplen esas condiciones (P10, P12, P13, P16, P21 y P29), y fueron los métodos más adecuados para definir la granularidad de los microservicios.

Sólo dos artículos (P14 y P22) proponen metodologías semiautomáticas que usan métricas para definir la granularidad, no han propuesto metodologías automáticas, la mayoría corresponden a metodologías manuales (P11, P17, P18 y P26) y sólo una semiautomática pero no utilizaba métricas (P25).

En P14 proponen un algoritmo de descomposición basado en flujo de datos. En su metodología, en primer lugar, se analiza la especificación de los casos de uso y la lógica del negocio en función de los requisitos; en segundo lugar, se construyen los diagramas detallados de flujo de datos (DFD) en diferentes niveles; en tercer lugar, diseñaron un algoritmo para condensar automáticamente el PD DFD a un DFD descomponible, en el que se combinan las sentencias entre procesos y almacenes de datos; por último, pero no menos importante, los candidatos a

microservicios fueron identificados y extraídos automáticamente del DFD descomponible [63]. Las métricas utilizadas por P14 fueron métricas de acoplamiento (acoplamiento aferente, acoplamiento eferente, inestabilidad) y cohesión (cohesión relacional). P14 no abordó ningún atributo de calidad directamente.

Tabla 11. Métodos automáticos o Semiautomáticos, que utilizaban métricas y abordaban algunos atributos de calidad.

Propuesta	Métricas	Atributos de calidad
MB	Acoplamiento: Absoluta importancia del microservicio, absoluta dependencia del microservicio, interdependencia entre microservicio. Cohesión: Falta de cohesión, grado de cohesión. Similitud semántica. Complejidad cognitiva, tiempo estimado de desarrollo, llamadas y dependencias entre los microservicios. Número de microservicios, numero de historias asociadas al microservicio.	Rendimiento, funcionalidad, mantenibilidad y modularidad.
P10	Tiempo de respuesta, Número de llamadas, Número de solicitud rechazada.	Escalabilidad y rendimiento
P12	Acoplamiento estructural, Falta de cohesión, Memoria media, Disco medio, Número de solicitudes, Tiempo de ejecución.	Escalabilidad, rendimiento y disponibilidad
P13	Acoplamiento lógico, Redundancia media de dominio, Acoplamiento de colaborador, Acoplamiento semántico, Recuento de confirmaciones, Recuento de colaboradores, Líneas de código.	Rendimiento
P16	Número de interfaz de integración, Cohesión a nivel de mensaje, Cohesión a nivel de dominio, Calidad de modularidad estructural, Calidad de modularidad conceptual, Frecuencia de coincuerdo interno y extermal (ICF, ECF). Relación de ECF a ICF.	Escalabilidad, funcionalidad, mantenibilidad, y modularidad
P21	Acoplamiento estructural, Falta de cohesión, CPU media, Red media, Número de ejecuciones, Número de paquetes enviados.	Escalabilidad, rendimiento y disponibilidad
P29	Número de microservicios, número de interfaces, líneas de código, análisis de paquetes, análisis de jerarquía de clases, análisis de estructura estática, análisis de gráfico de llamadas estáticas, análisis estático y dinámico combinado.	Escalabilidad y rendimiento.

MB: Microservices backlog, esta propuesta doctoral.

P22 fue un algoritmo de agrupación aplicado a entidades de dominio agregadas, utilizó métricas de acoplamiento (puntaje “silhouette”) y de complejidad (número de singleton cluster, tamaño máximo del cluster). Propusieron un enfoque para la migración de las aplicaciones monolíticas a una arquitectura de microservicios que se centraba en el impacto de la descomposición en la lógica de negocio del monolito [68].

2.3. MÉTRICAS USADAS PARA EVALUAR LA GRANULARIDAD DE LOS MICROSERVICIOS.

Las métricas de software permiten medir y monitorear diferentes aspectos y características de un proceso y producto de software, se han propuesto métricas a nivel de diseño, implementación, prueba, mantenimiento y despliegue. Estas métricas permiten comprender, controlar y mejorar lo que sucede durante el desarrollo y mantenimiento del software para tomar acciones correctivas y preventivas.

En los métodos y modelos identificados, la mayoría (59%) utilizaron alguna métrica para determinar la granularidad de los microservicios. Se esperaba una mayor importancia de las métricas en los métodos automáticos para validar la granularidad en las aplicaciones basadas en microservicios y evaluar las descomposiciones obtenidas por los métodos.

Se identificaron métricas para el acoplamiento, la cohesión, la granularidad, la complejidad, el rendimiento, el uso de recursos computacionales, el equipo de desarrollo, el código fuente, etc. (ver figura 23). Las cuales se clasificamos en cuatro grupos: 1) sobre el equipo de desarrollo, 2) sobre el proceso de desarrollo de los microservicios, 3) sobre el sistema y 4) sobre cada microservicio. La mayoría de las métricas identificadas se centraron en el microservicio (40), y sólo dos se refieren a los procesos de desarrollo de microservicios. Existe un vacío de investigación en la propuesta de métricas que evalúen el proceso de desarrollo completo de las aplicaciones basadas en microservicios y su impacto en la granularidad de los microservicios.

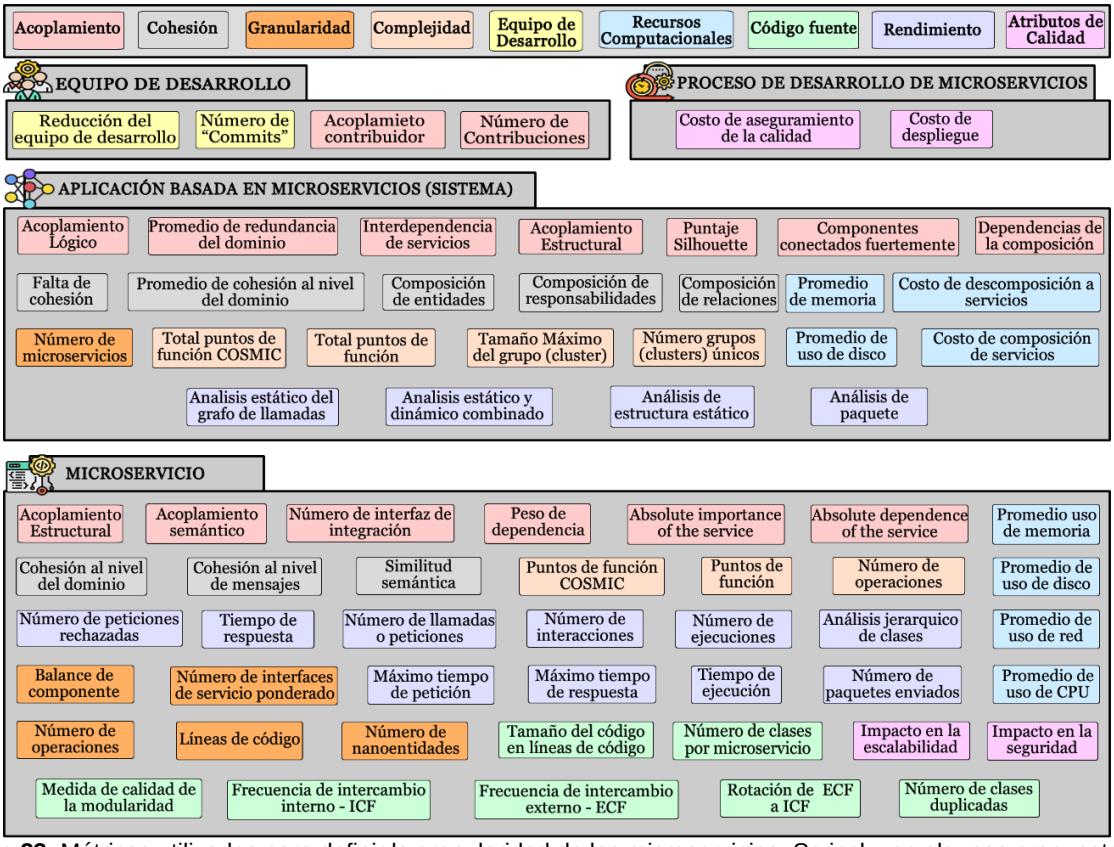


Figura 23. Métricas utilizadas para definir la granularidad de los microservicios. Se incluyen algunas propuestas por Bogner y otros [101], por Candela y otros [102] y por Rud y otros [103]

La tabla 12 presenta las métricas usadas por los trabajos relacionados con la definición de la granularidad de los microservicios; las métricas más utilizadas en los trabajos seleccionados están relacionadas con el acoplamiento (14 métricas propuestas), seguidas por el rendimiento y la cohesión (13 métricas), a continuación, las métricas de recursos computacionales (8 métricas), y la complejidad y el código fuente (7 métricas).

Nueve trabajos utilizaron métricas de acoplamiento (P11, P12, P13, P14, P15, P16, P21, P22, P24), y siete trabajos utilizaron métricas de cohesión (P12, P14, P16, P21, P24, P27, P28), mientras que las métricas de rendimiento fueron utilizadas por cinco trabajos (P4, P10, P12, P21, P23); las métricas de complejidad fueron consideradas por dos trabajos (P8, P22), aunque son características fundamentales de los microservicios. Se requieren más propuestas que incluyan más métricas de complejidad, así como métricas relacionadas con el proceso de desarrollo de microservicios. Las demás métricas fueron utilizadas por un solo documento cada una. Se encontraron que 11 documentos que utilizan métricas de acoplamiento o cohesión, y 5 documentos utilizan ambas. Sólo uno (P24) utilizó métricas de acoplamiento, cohesión y

complejidad. El modelo propuesto en este trabajo de investigación utiliza métricas de acoplamiento, cohesión, complejidad, rendimiento y granularidad.

Tabla 12. Métricas usadas por los trabajos relacionados con la definición de la granularidad.

Art.	Tipo De Métrica								
	Acoplamiento	**Cohesión**	**Granularidad**	**Complejidad**	**Equipo Des.**	**Recursos Comput.**	**Código**	**Rendimiento**	**Otros Atributos C.**
P3									Cost of quality assurance. Cost of deployment.
P4								Response time. Number of calls.	
P8				Cosmic function points.					
P10						Avg. Cpu. Used virtual machines. Allocated virtual machines.		Response time slo violations. Number of calls. Number of rejected request. Throughput.	
P11	Dependency weight.								Security impact. Scalability impact.
P12	Structural coupling.	Lack of cohesion.				Avg. Memory. Avg. Disk.		Number of requests. Execution time.	
P13	Logical coupling. Average domain redundance. Contributor coupling. Semantic coupling.				Commit count. Contributor count.		Lines of code		
P14	Afferent coupling. Efferent coupling. Instability.	Relational cohesion.							
P15	Coupling between microservice.						Number of classes. Number of duplicated classes.		
P16	Integrating interface number.	Cohesion at message level. Cohesion at domain level.					Structural modularity quality. Conceptual modularity quality. Internal and external co-change frequency (icf, ecf). Ration of ecf to icf.		
P21	Structural coupling.	Lack of cohesion.				Avg. Cpu Avg. Network.		Number of executions. Number of packets send.	
P22	Silhouette score.			Number of singleton clusters. Maximum cluster size.					
P23						Service composition cost. Service decomposition cost.		Request time	
P24	Dependencies composition. Strongly connected components.	Nano-entities composition. Relation composition. Responsibilities composition. Semantic similarity.	Number of nano-entities.						
P27		Cohesion at message level. Cohesion at domain level.	Operation number.					Interaction number.	
P28		Lack of cohesion.	Operation number.						
P29			Number of microservices. Number of interfaces.				Lines of code. Package analysis. Class hierarchy analysis.	Static structure analysis. Static call graph analysis. Combined static and dynamic analysis.	

El tamaño y el número de microservicios que componen una aplicación afecta directamente a su mantenimiento. La automatización de las pruebas, la integración continua y el despliegue son

esenciales, especialmente cuando los microservicios y muchos sistemas distribuidos deben ser gestionados independientemente.

Bogner y otros [101] realizaron una revisión de literatura para medir la mantenibilidad del software e identificaron métricas en cuatro propiedades de diseño: tamaño, complejidad, acoplamiento y cohesión; para los sistemas basados en servicios, también analizaron su aplicación a los sistemas basados en microservicios y presentaron un modelo de mantenibilidad para servicios (MM4S), que consiste en las propiedades de los servicios relacionadas con las métricas de los servicios recolectadas automáticamente. Las métricas propuestas por ellos pueden utilizarse o adaptarse para determinar la granularidad adecuada de los microservicios que van a formar parte de una aplicación.

Teniendo en cuenta [101], [102], [103] junto con los trabajos relacionados, detallamos las siguientes métricas, que pueden ser utilizadas o adaptadas para definir la granularidad de los microservicios y para evaluar las descomposiciones.

2.4.1. Métricas de acoplamiento.

El acoplamiento mide el grado de dependencia de un componente de software en relación con otro. Si hay un alto grado de acoplamiento, el componente de software no puede funcionar correctamente sin el otro componente; además, cuando se cambia un componente de software, debo cambiar obligatoriamente el otro componente. Por estas razones, al diseñar aplicaciones basadas en microservicios, se debe buscar un bajo grado de acoplamiento entre cada microservicio.

Mazlami y otros [90] representaron la información del monolito y crearon un gráfico G sin dirección y con bordes ponderados. Cada borde del gráfico tiene un peso definido por la función de peso; esta función de peso determina cuán fuerte es el acoplamiento entre los bordes vecinos, de acuerdo con la estrategia de acoplamiento en uso. Estas estrategias de acoplamiento pueden utilizarse como métricas para definir la granularidad. A continuación, se enumeran las métricas utilizadas por los trabajos relacionados:

- Peso de la dependencia. [87]
- Acoplamiento lógico. [104], [84]
- Acoplamiento semántico. [84].
- Conteo de contribuciones y acoplamiento de contribuciones. [84].
- Acoplamiento estructural. [102].
- Acoplamiento aferente (Ca). [105] cited by [63].
- Acoplamiento eferente (Ce). [105] cited by [63].
- Inestabilidad (I). [105] cited by [63].
- Acoplamiento entre microservicios (CBM). [64]
- Número de interfaces de integración (IFN). [61].
- Redundancia media de dominio (ADR). [84]
- Importancia absoluta del servicio (AIS). [103].
- Dependencia absoluta del servicio (ADS). [103].
- Interdependencia de los servicios en el sistema (SIY). [103].
- Composición de las dependencias. [70]
- Componentes fuertemente conectados (SCC). [70].
- Puntuación de la silueta (Silhouette score). [68]

Perepletchikov y otros [112] también propusieron un conjunto de métricas a nivel de diseño para medir el atributo estructural del acoplamiento en sistemas orientados a servicios, que pueden adaptarse a los microservicios.

2.4.2. Métricas de cohesión.

La cohesión y el acoplamiento son dos objetivos contrastantes. Una solución que equilibre la alta cohesión y el bajo acoplamiento es el objetivo de los desarrolladores. Candela y otros [102] emplearon un enfoque de dos objetivos que apunta a maximizar la cohesión de los paquetes software y minimizar el acoplamiento de los paquetes. Utilizaron las dependencias de clase y la información estructural para medir la cohesión estructural, que puede adaptarse a los microservicios. A continuación, se enumeran las métricas de cohesión utilizadas en los trabajos relacionados:

- Falta de cohesion [102], [60].
- Cohesión relacional. [106] cite by [63].
- Cohesión a nivel del dominio (CHD). [61].
- Promedio de cohesión a nivel del dominio (Avg. CHD). [61].
- Cohesión a nivel de mensajes (CHM). [61].
- Cohesión de datos de la interfaz de servicio (SIDC). [107].
- Cohesión de uso de la interfaz de servicios (SIUC). [107].
- Composición de las entidades [70].
- Composición de las relaciones [70].
- Composición de las responsabilidades [70].
- Similitud semántica [70]

2.4.3. Métricas de complejidad.

La complejidad de los microservicios debe ser baja, para que puedan ser modificados en un par de semanas, reescritos y mejorados rápidamente. Si la complejidad es alta, entonces el costo del cambio es mayor. La medición de la complejidad es fundamental para desarrollar aplicaciones basadas en microservicios. A continuación, se enumeran las métricas utilizadas por los autores de los trabajos relacionados:

- Puntos de función [108].
- Puntos de función COSMIC [74].
- Respuesta total para el servicio (TRS) [109].
- Número de grupos de una sola instancia (Singleton cluster) (NSC). [68]
- Tamaño máximo del grupo (MCS). [68]

Perepletchikov y otros [107] examinaron las categorías de cohesión propuestas inicialmente para los programas informáticos orientados a los objetos a fin de determinar su pertinencia conceptual para los diseños orientados a los servicios; y propusieron un conjunto de medidas de cohesión que pueden adaptarse a los microservicios.

2.4.4. Métricas de rendimiento.

El rendimiento es un punto crítico de las aplicaciones basadas en microservicios. Los trabajos seleccionados utilizaron ocho medidas de rendimiento:

- El número de llamadas o solicitudes.
- El número de solicitudes rechazadas.
- El tiempo de respuesta o ejecución.
- Número de interacciones [72].
- Número de ejecuciones.
- Tiempo máximo de solicitud.
- Tiempo máximo de respuesta.
- Número de paquetes enviados.

Ren y otros [84] (P29) utilizaron el análisis de paquetes (PA), el análisis de estructura estática (SSA), el análisis de jerarquía de clases (CHA), el análisis de grafo de llamadas estáticas (SCGA) y el análisis combinado estático y dinámico (CSDA) para evaluar el rendimiento de la migración. Sin embargo, no explicaron los detalles de las pruebas de análisis de rendimiento ni las métricas que utilizaron.

2.4.5. Métricas relacionadas a otros atributos de calidad.

Pocas métricas estaban directamente relacionadas con los atributos de calidad. Las métricas propuestas en los trabajos revisados se definen de la siguiente manera:

- El costo de aseguramiento de la calidad[80].
- El costo de despliegue [80].
- Impacto en la seguridad [87].
- Impacto en la escalabilidad [87].

2.4.6. Métricas de recursos computacionales.

Los recursos computacionales son todos los elementos de software y hardware necesarios para el funcionamiento de las aplicaciones basadas en microservicios. A continuación, se listan las métricas propuestas.

- Promedio de memoria.
- Promedio de disco
- Promedio de uso de red [67].
- Promedio de CPU.
- Costo de composición del servicio (SCC) [69].
- Costo de descomposición del servicio (SDC). [69]

2.4.7. Métricas relativas al equipo de desarrollo.

Cada microservicio puede ser desarrollado por un equipo diferente, y con diferentes lenguajes de programación y diferentes gestores de base de datos. Es importante considerar métricas que permitan analizar la granularidad de los microservicios y sus impactos en el equipo de desarrollo. Las métricas propuestas en el análisis son las siguientes:

- Reducción del tamaño del equipo (TSR). [84].
- Cuenta de envíos o confirmaciones en el repositorio de código (Commit count).

Encontramos muy pocas métricas relacionadas con el equipo de desarrollo. Este puede ser un tema interesante para futuras investigaciones.

2.4.8. Métricas relativas al código fuente.

El código fuente es una de las fuentes más importantes para analizar ciertas características de una aplicación. Algunos autores lo han usado para identificar microservicios y definir su granularidad. A continuación se describen las métricas propuestas:

- Tamaño del código en líneas de código.
- El número de clases por microservicio. [64].
- El número de clases duplicadas. [64].
- Frecuencia de co-cambio interna (ICF) [61].
- Frecuencia de co-cambio externo (ECF). [61].
- La proporción de ECF a ICF (REI). [61].
- Medida de calidad de la modularidad [102], [61].

2.4.9. Métricas de granularidad.

La medición de la granularidad es compleja. La granularidad está relacionada con el tamaño, incluyendo el número de funcionalidades o servicios que tendrá la aplicación o el microservicio. También está relacionada con el acoplamiento y la cohesión. Ser más granular implica que el microservicio no tiene dependencias y puede funcionar sólo, como una pieza independiente y encapsulada. Se identificaron seis métricas de granularidad:

- Recuento ponderado de la interfaz de servicio (WSIC) [110].
- Balance de componentes (CB). [111].
- Número de operación (OPN). [72].
- Número de microservicios.
- Líneas de código.
- Número de nanoentidades. [70].

El detalle y la definición de cada métrica se puede encontrar en el artículo de revisión sistemática de literatura resultado de este trabajo de investigación.

Los fundamentos de los microservicios sugieren que deben tener un bajo acoplamiento, alta cohesión y baja complejidad. Sobre la base de las métricas descritas, se podría definir un modelo o método que utilice inteligencia artificial para determinar el dimensionamiento y el tamaño más adecuado para los microservicios; en esta investigación se abordó este problema. Algunos ya han sido definidos, principalmente para las migraciones de aplicaciones monolíticas a microservicios. El número de microservicios, su tamaño y su complejidad computacional afectan directamente al uso de los recursos computacionales y, por lo tanto, afectan su costo de despliegue. Este es un tema interesante para futuras investigaciones.

En conclusión, algunos trabajos utilizaron métricas para evaluar la granularidad de los microservicios, incluyendo el acoplamiento, la cohesión, el número de llamadas, el número de solicitudes y el tiempo de respuesta, aunque pocos métodos o técnicas utilizan la complejidad como métrica, aunque es fundamental para los microservicios. Nunes y otros [68] utilizaron el número de cluster únicos y el tamaño máximo de los cluster como métricas de complejidad, y Vural et al. [74] utilizaron los puntos de función COSMIC (Common Software Measurement International Consortium). La complejidad cognitiva no fue considerada por ningún trabajo relacionado. Se necesita más investigación que considere métricas a nivel de diseño para definir

la granularidad de los microservicios que forman parte de una aplicación, así como investigación que proponga modelos, métodos o técnicas para determinar la granularidad más apropiada.

Las técnicas más utilizadas corresponden a aprendizaje automático, similitud semántica, programación genética y la ingeniería de dominio. Ninguno de los trabajos relacionados utilizó las prácticas del desarrollo ágil de software como insumos, (es decir, historias de usuarios, product backlog, planificación de lanzamiento, tableros Kanban, entre otros) para definir o evaluar la granularidad de los microservicios. Ninguna de las propuestas identificadas en el estudio se centró en el desarrollo de software ágil.

La escalabilidad y el rendimiento (características observables en tiempo de ejecución) fueron los atributos de calidad más tratados en los documentos examinados; la modularidad y la mantenibilidad (características del software como artefacto) fueron los atributos de calidad menos utilizados. Sólo un artículo [61] abordó tanto características del tiempo de ejecución como características del software como artefacto, entre estas escalabilidad, funcionalidad, modularidad y mantenibilidad. En ningún documento se utilizaron la funcionalidad, el rendimiento, la modularidad y la mantenibilidad al mismo tiempo.

3. PROCESO DE DESARROLLO DE APLICACIONES BASADAS EN MICROSERVICIOS.

El proceso se divide en dos partes fundamentales, primero el desarrollo de cada microservicio y segundo, el desarrollo de aplicaciones que utilizan esos microservicios. El proceso se presenta en la figura 24. Se hicieron unas adaptaciones al proceso presentado en [96], se incluyó las herramientas que permiten realizar el descubrimiento de servicios y las herramientas que permiten gestionar los “sidecars”. El “sidecar” es un proceso auxiliar que se ejecuta al lado de la aplicación, se ejecuta al lado de cada microservicio, con el fin de proporcionar características adicionales: como tolerancia a fallas, registro, descubrimiento y orquestación, evitando el acoplamiento [112].

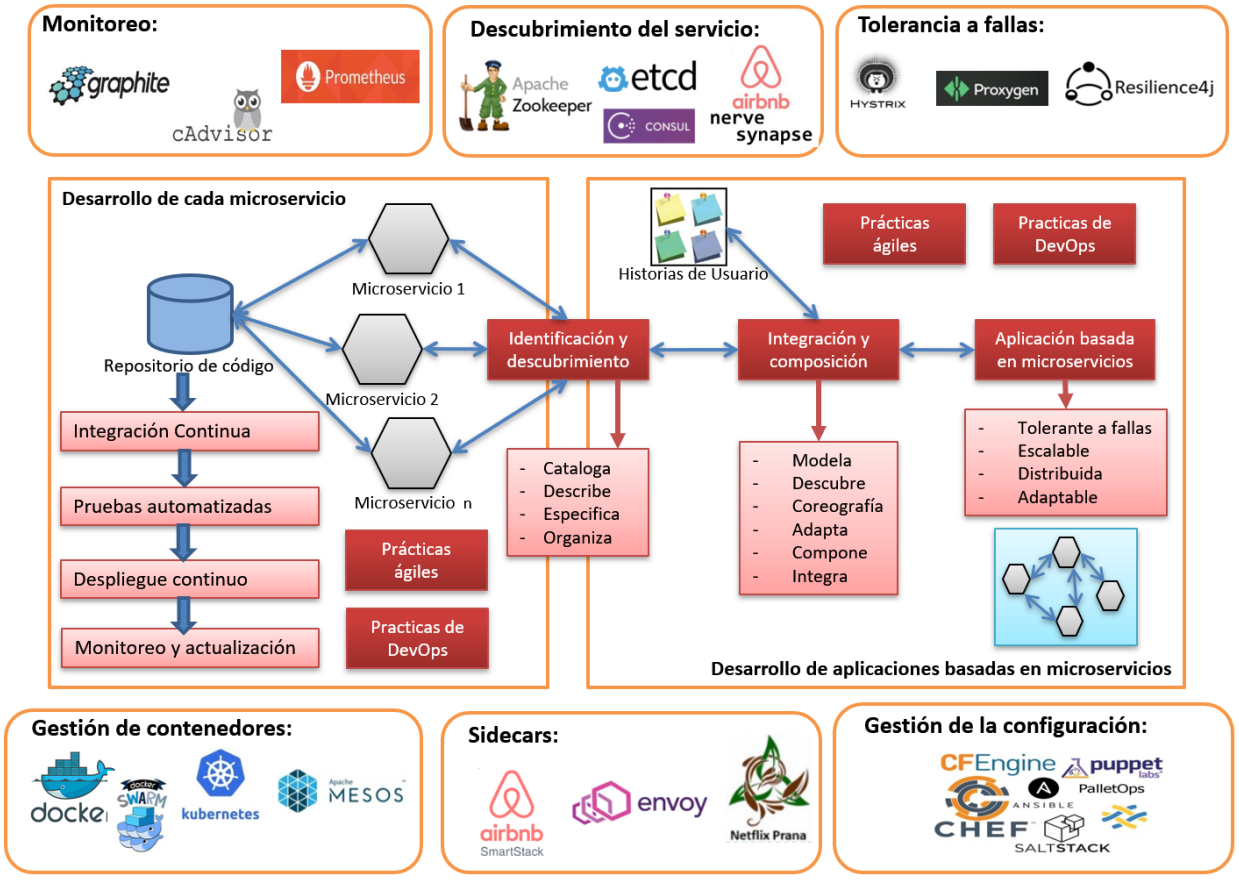


Figura 24. Caracterización del proceso de desarrollo de aplicaciones basadas en microservicios

A continuación, se enuncian las fases de desarrollo de cada microservicio:

- **Desarrollo del microservicio:** Cada microservicio es desarrollado independientemente por un equipo de desarrollo separado, incluyendo un lenguaje de programación diferente y una base de datos diferente. Normalmente se desarrolla usando prácticas ágiles, entregas iterativas e incrementales.
- **Integración continua:** Cada equipo sincroniza e integra diariamente el código del microservicio que se está desarrollando.

- **Pruebas automatizadas:** Una vez integrado el código, el repositorio ejecuta una serie de pruebas (unidad, funcionalidad, carga, otros) para garantizar la calidad del código implementado.
- **Despliegue continuo:** Si las pruebas son satisfechas, se despliega automáticamente al servidor de pruebas o de producción.
- **Monitoreo y actualización:** Cada microservicio se monitorea de forma independiente, de modo que se puedan identificar las fallas o necesidades de escalado o aumento de los recursos computacionales. La actualización se realiza de forma independiente, pudiendo cambiar un microservicio sin afectar a sus clientes.

Las fases de desarrollo de la aplicación basada en microservicios son las siguientes:

- **Identificación y descubrimiento:** Permitir a los clientes de los microservicios encontrar y establecer comunicación.
- **Integración y composición:** Identificados los microservicios que cumplen los requisitos de la aplicación, se procede a la integración y composición de la aplicación empresarial. En algunos casos, es necesario adaptar algún microservicio para que cumpla con los requisitos de la aplicación o construir uno nuevo desde cero.
- **Desplegar la aplicación empresarial:** La aplicación se empaqueta, despliega y monitoriza utilizando las prácticas de DevOps de la misma forma que en la construcción de cualquier microservicio. Por lo tanto, la aplicación implementada es tolerante a fallas, escalable, adaptable y distribuida.

Se puede resaltar que la arquitectura de microservicios ayuda a reducir la complejidad de gestionar y manejar las características de calidad necesarias para las aplicaciones de software como servicios (SaaS), como son: escalado automático, pruebas automatizadas, integración y despliegue continuo, seguridad y tolerancia a fallas.

En el proceso de desarrollo de aplicaciones basadas en microservicios se requieren actualizaciones permanentes, más rápidas y automatizadas usando las prácticas de DevOps. Las aplicaciones basadas en microservicios requieren una infraestructura DevOps más sofisticada, que generalmente requiere la construcción de una tubería “pipeline” de implementación y despliegue continuos que garantice la calidad de los microservicios y puesta en producción más rápida. El uso de las tecnologías de la nube, en especial los contenedores permiten la construcción de dichas tuberías.

DevOps (Dev: Developers – Ops: Operaciones) es un término que surge con la colisión de dos nuevas tendencias, infraestructura ágil u operaciones ágiles, y colaboración entre el personal de desarrollo y de operaciones a lo largo de todas las etapas del ciclo de vida del desarrollo [113], DevOps se trata de procesos de desarrollo y aprovisionamiento de negocios rápidos y flexibles. Integra de manera eficiente el desarrollo, la entrega y las operaciones, lo que facilita una conexión ágil y fluida de estos silos tradicionalmente separados [114].

Cada microservicio se desarrolla de forma independiente de los demás, el desarrollo se puede hacer de forma paralela, cada uno por un equipo de desarrollo independiente, teniendo en cuenta las dependencias entre cada uno de ellos, de esta forma el desarrollo de una aplicación se puede realizar en menor tiempo. Aun se presentan desafíos de investigación en este punto, J. Soldani y otros [40], afirman que tener muchos equipos autónomos que desarrollan servicios desplegados de forma independiente puede ser un arma de doble filo. Por un lado, cada equipo puede tomar decisiones locales sin tener que negociar con otros equipos. Por otro lado, aumenta el riesgo de que los equipos no vean el panorama general, es decir, que entiendan si sus decisiones locales

son justificables y coherentes en el contexto de la arquitectura general y los objetivos comerciales de la aplicación.

Otro punto importante dentro del proceso de desarrollo de aplicaciones basadas en microservicios es el descubrimiento de los microservicios. En una aplicación basada en microservicios, el conjunto de instancias de servicio en ejecución cambia dinámicamente, incluidas las ubicaciones de red. En consecuencia, para que un cliente pueda realizar una solicitud a un servicio, debe utilizar un mecanismo de descubrimiento de servicios. Una parte clave del descubrimiento de servicios es el registro de servicios. El registro de servicios es una base de datos de instancias de servicios disponibles. El registro de servicios proporciona una API de administración y una API de consulta. Las instancias de servicio se registran y se cancelan del registro de servicio mediante la API de administración, quien se encarga de garantizar que los microservicios registrados estén disponibles, funcionando y sin errores. Los componentes del sistema utilizan la API de consulta para descubrir la instancia de servicio disponible [115]. Las herramientas que permiten administrar el descubrimiento de servicios son: Etcd, Consul, Nerve y Synapse.

Finalmente, la integración y composición de la aplicación permite a partir de las historias de usuario o requerimientos funcionales de la aplicación que se quiere construir, identificar, integrar y/o adaptar los microservicios que van a ser parte de la aplicación.

3.1. CASO DE ESTUDIO: DESARROLLO DE SINPLAFUT UNA APLICACIÓN BASADA EN MICROSERVICIOS

Sinplafut es el sistema de información que permite mejorar y agilizar la planificación del entrenamiento de fútbol utilizando técnicas modernas de entrenamiento deportivo. Sinplafut puede ser utilizado por equipos de fútbol, clubes, preparadores físicos, entrenadores y directores técnicos, tanto a nivel aficionado como profesional, como una aplicación de software como servicio - SaaS. Con Sinplafut puede administrar su perfil, la ficha de cada jugador, la información fisiológica y las lesiones. Permite crear y configurar los macrociclos, los mesociclos, los microciclos y sesiones de entrenamiento, utilizando métodos y test preestablecidos, administrados y controlados por Sinplafut [116].

El desarrollo de Sinplafut comienza con la especificación de los requerimientos funcionales como historias de usuario, priorizadas y estimadas en un “product backlog”, para la gestión del desarrollo y siguiendo la metodología ágil Scrum se utilizó la herramienta Taiga, Taiga es una plataforma de gestión de proyectos para startups y desarrolladores y diseñadores ágiles, lleva el control del avance del desarrollo de la aplicación.

Teniendo identificados y priorizados los requerimientos funcionales, se definió la granularidad de los microservicios que van a ser parte de Sinplafut, siguiendo el principio propuesto por Evans "Se debe definir un contexto delimitado para cada concepto de dominio que se expondrá como servicio"[24]. Al momento de iniciar la implementación, el equipo de desarrollo de Sinplafut no contaba con microservicios ya implementados, por lo tanto, cada microservicio se debe desarrollar desde cero. La tabla 4 muestra este resultado, se detallan los microservicios y las historias de usuario asignadas. Esta asignación se hizo a prueba y error, a criterio del equipo de desarrollo, en el momento de realizar el diseño de Sinplafut, no se conoce o no se ha publicado algún modelo, método o herramienta que permita definir bajo algún criterio la granularidad optima de los microservicios. Este punto es un tema de investigación abierto y de mucha discusión, esto lo confirman Lewis et al, afirmando que se el problema de la granularidad es la falta de acuerdo sobre el tamaño correcto de los microservicios. El hecho de que estén etiquetados como

"microservicios" muestra que existe la posibilidad de establecer un conjunto de patrones para ayudar con las decisiones de diseño cuando se está dividiendo un dominio en microservicios y dimensionando cada servicio [23].

Tabla 13. Requerimientos de la aplicación y granularidad de los microservicios de Sinplafut

Historias de usuario épicas	Microservicio	Número de historias
Gestión del club deportivo Gestión de equipo del club deportivo Gestión de jugadores del equipo Gestión del cuerpo técnico	Equipoapp	27
Gestión del plan de entrenamiento Gestión de mesociclos Gestión de microciclos Gestión de sesiones de entrenamiento Gestión de métodos del entrenamiento deportivo Generar cronograma de las sesiones de entrenamiento	Planentrenamientoapp	33
Generar el landing page de sinplafut - frontend	Sinplafutapp	1
Gestión de test deportivos	Testdeportivosapp	14
Gestión de usuario y seguridad	Seguridadapp	17
Total	5 microservicios	92

La idea principal es desarrollar cada microservicio de forma individual e independiente de los otros, por lo tanto, es necesario crear un repositorio de código para el control de versiones por cada microservicio, de igual forma se requiere una plataforma de integración continua, pruebas automatizadas y despliegue continúo manejadas individualmente para cada microservicio. La Figura 25 muestra la plataforma de desarrollo implementada para cada microservicio. Para el caso de Sinplafut se crearon 5 canales o “pipelines” de integración continua, despliegue continuo y pruebas automatizadas.

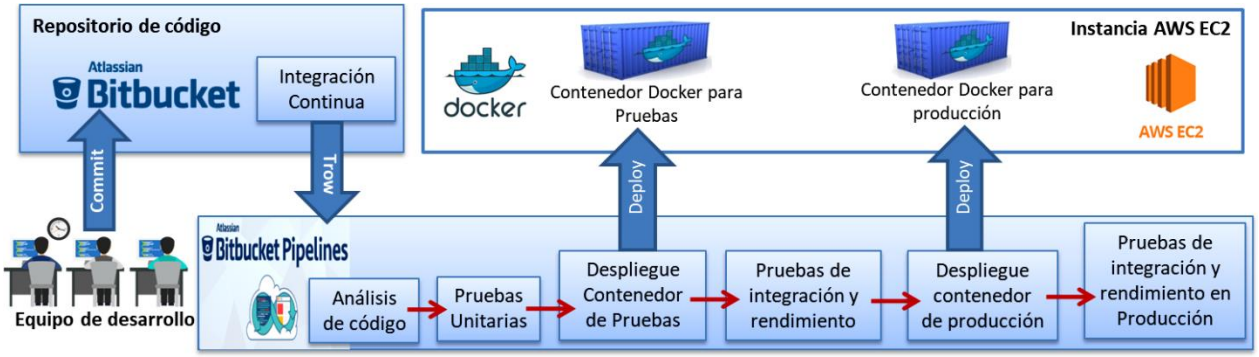


Figura 25. Plataforma de desarrollo implementada para cada microservicio: integración continua, despliegue continuo, pruebas automatizadas, utilizando Docker y AWS EC2.

Se implemento un prototipo de cada microservicio en Django (Front -End) y el framework Rest. Cada microservicio se implementó, probó y desplegó utilizando las prácticas de DevOps y un canal de implementación creado en Bitbucked. Bitbucked pipeline permite la escritura de scripts dentro de contenedores donde se pueden instalar y ejecutar herramientas para realizar las pruebas y el despliegue. Con el canal de despliegue continuo se garantiza entregas rápidas y de calidad, probadas tanto en un ambiente de pruebas y en producción, reduciendo el tiempo de puesta en producción. Por cuestiones de tiempo y del alcance de este trabajo de investigación doctoral no se implementaron la totalidad de las historias de usuario.

La figure 26 muestra la arquitectura de implementación de cada microservicio. Se puede apreciar el uso de “sidecars” utilizando la herramienta SmartStack desarrollada por Airbnb, la cual utiliza para el registro Apache Zookeeper junto con Nerve y Synapse para el descubrimiento de servicios y tolerancia a fallas. Nerve se encarga de verificar y que cada microservicio se encuentre en

línea y funcionando correctamente, en este caso registra en ZooKeeper el puerto y el estado del microservicio, en caso contrario lo borra de zookeeper y lo marca como no disponible. La puerta de enlace API es el punto de entrada única para todos los clientes. También es capaz de exponer diferentes API a diferentes clientes. Todas las solicitudes de los clientes se dirigen primero a una puerta de enlace API. Luego, la puerta de enlace de la API los encamina a su microservicio correspondiente a través del Load Balancer. También realiza la validación de solicitudes básica y el almacenamiento en caché de respuestas [117].

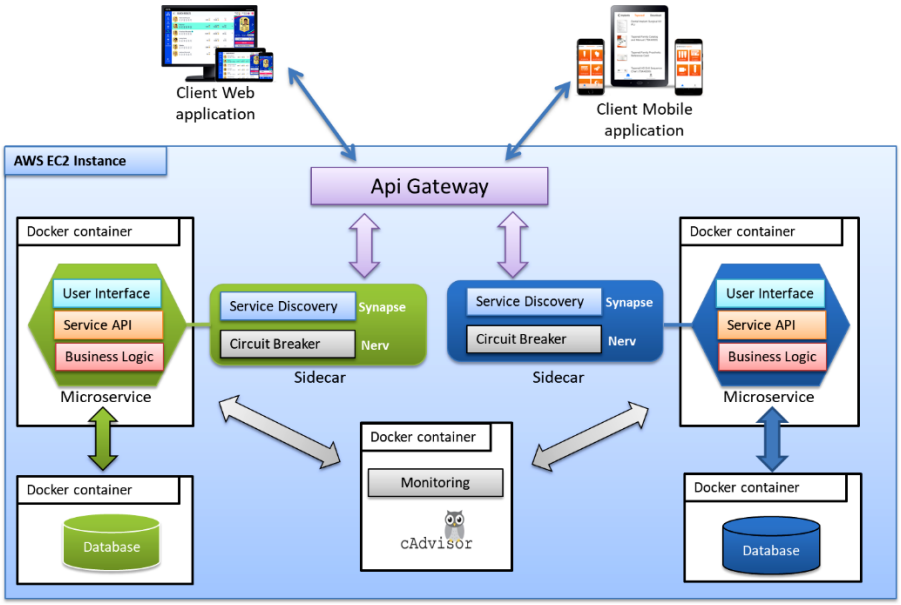


Figura 26. Diagrama de dependencias de los microservicios de Sinplafut. Cada microservicio es una aplicación independiente.

El monitoreo permite mantener un control centralizado de los microservicios y poder prevenir fallas y caídas de la aplicación. CAdvisor en una herramienta implementada por Google, que permite monitorear la instancia EC2 donde están desplegados los contenedores y cada uno de los contenedores de los microservicios. Se puede visualizar el uso de CPU, Memoria Ram y la capacidad de disco disponible. Prometheus es una herramienta más especializada para realizar el monitoreo, permite crear alertas y métricas para los microservicios que hacen parte de una aplicación. Como trabajo futuro se pretende crear un cuadro de mando con diferentes alertas y métricas con el uso de Prometheus para Sinplafut.

La definición del proceso facilita la implementación de los microservicios porque los desarrolladores tienen una guía inicial y detallada de las fases, métodos, herramientas y buenas prácticas que pueden utilizar durante la construcción de aplicaciones basadas en microservicios. Para que el proceso propuesto sea completo se quiere proponer y detallar patrones de diseño y desarrollo que permitan a los desarrolladores agilizar la construcción de microservicios.

El desarrollo de aplicaciones basadas en microservicios es complejo, implica el uso de una gran cantidad de herramientas necesarias para su desarrollo, despliegue, monitoreo y mantenimiento. Se requiere administrar una infraestructura más compleja y distribuida que la manejada en una aplicación monolítica. Las herramientas de gestión de la configuración son esenciales y permiten reducir la complejidad del aprovisionamiento y despliegue de contenedores y microservicios. El uso de contenedores en ambientes de desarrollo, pruebas y producción permiten reducir los

errores causados por diferencias en las configuraciones, librerías, versiones y dependencias, logrando el concepto de servidor inmutable.

Una vez aprovisionada toda la plataforma tecnológica necesaria para el desarrollo de cada microservicio, los beneficios se amplifican considerablemente, el tiempo de puesta en producción de un incremento de producto se reduce, la calidad de ese incremento se garantiza por medio de la canalización o tubería de despliegue continuo y las pruebas automatizadas, la aplicación basada en microservicios es tolerante a fallas, si falla un microservicio no se afecta a toda la aplicación. La gestión de recursos computacionales se realiza de forma individual de acuerdo con las necesidades propias de cada microservicio, de igual forma que el escalado, se realiza de acuerdo con las necesidades de cada microservicio.

4. MICROSERVICE BACKLOG – MODELO DE ESPECIFICACIÓN DE LA GRANULARIDAD DE LOS MICROSERVICIOS

El problema que se aborda en esta investigación comienza cuando el equipo de desarrollo o el arquitecto, después de realizar un análisis, determina que la aplicación a crear necesita ser implementada utilizando la arquitectura de microservicios. En la revisión de literatura no encontramos una práctica ágil que nos permita modelar, crear, operar y gestionar los microservicios de la aplicación en tiempo de diseño. Determinar el número de microservicios, su tamaño o granularidad es una tarea compleja.

El "Microservice Backlog" se centró en tres actividades relevantes: 1) Determinar y evaluar la granularidad de los microservicios, 2) establecer el número de historias de usuarios asignadas a un microservicio, y 3) establecer el número óptimo de microservicios que formarán parte de la aplicación. Estas actividades se basan en las propiedades de bajo acoplamiento, alta cohesión y baja complejidad de los microservicios. Las métricas fueron adaptadas y calculadas para evaluar la descomposición o de las aplicaciones basadas en microservicios.

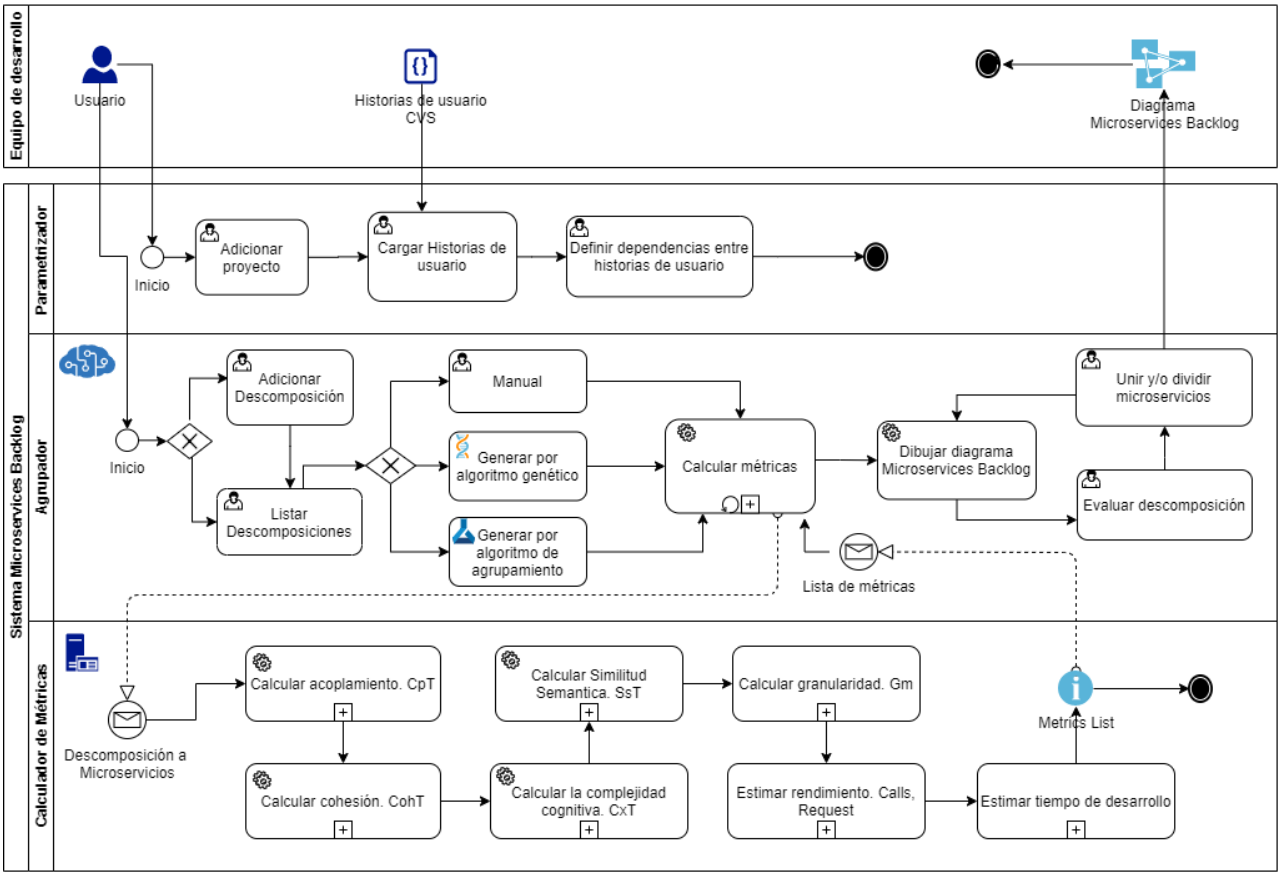


Figura 27. Modelo Microservices Backlog. Descomposición semiautomática de historias de usuarios a microservicios.

El Microservice Backlog es una técnica, diseñada para analizar gráficamente la granularidad de los microservicios, a partir de un conjunto de requisitos funcionales expresados como historias de usuarios dentro de un "product backlog". El modelo especifica la arquitectura de las aplicaciones basadas en microservicios. Después de esto, el arquitecto o el equipo de desarrollo puede evaluar la granularidad o el tamaño apropiado de cada microservicio considerando algunas características como la complejidad, el acoplamiento, la cohesión y el tiempo de desarrollo en

tiempo de diseño. De esta manera, el arquitecto o el desarrollador puede encontrar una estrategia para su implementación. Véase la figura 27.

El modelo se implementó en una aplicación web (Django - Python) y consta de los siguientes componentes:

1. Especificación formal del modelo de granularidad.
2. Componente parametrizador.
3. Componente agrupador, el cual implementa las técnicas de agrupamiento.
4. Componente calculador de métricas
5. Diagrama Microservices Backlog y evaluador de descomposiciones.

4.1. ARQUITECTURA DEL SISTEMA MICROSERVICES BACKLOG

La arquitectura del sistema se basa en la arquitectura de microservicios, se implementaron tres componentes: 1) El componente parametrizador, 2) El componente agrupador y 3) El componente calculador de métricas, estos corresponden al Front-End de la aplicación; estos componentes se comunican con los microservicios que implementan la lógica del negocio y representan el Back-End del sistema a través de la puerta de enlace (Api Gateway) quien se encarga de enviar las peticiones a cada microservicio por medio del descubridor de servicios (Service Discovery), cada microservicio expone sus operaciones a través de un API REST (Ver figura 28).

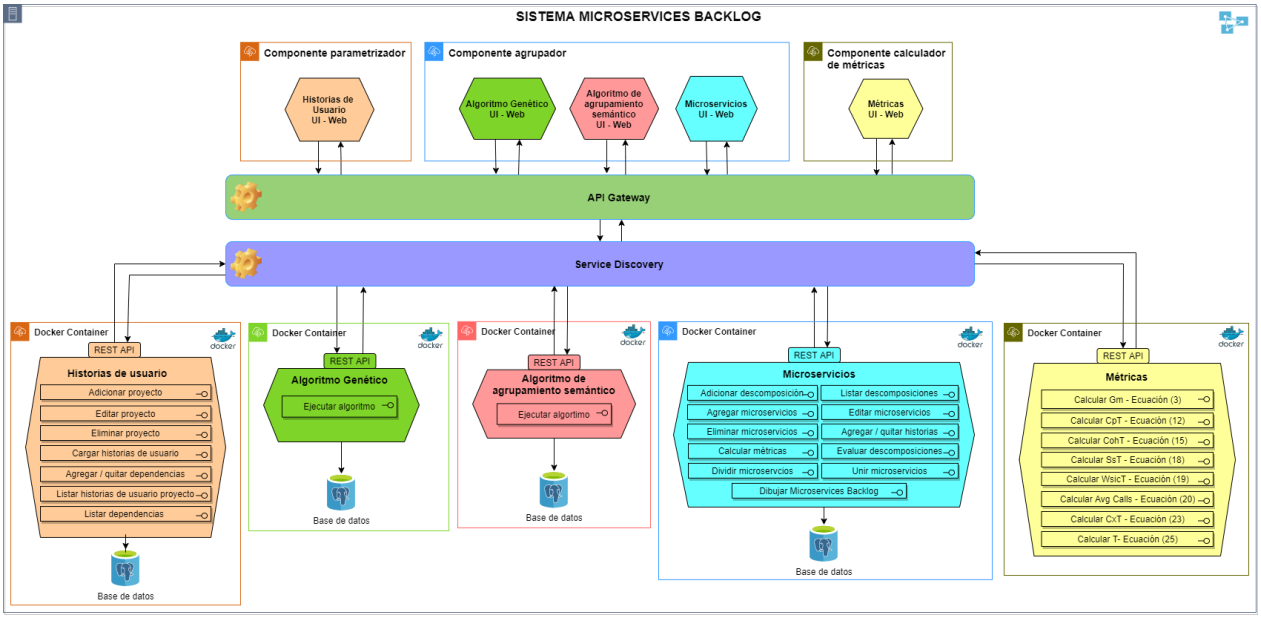


Figura 28. Arquitectura del sistema Microservices Backlog

4.1.1. Componente parametrizador:

Este componente corresponde a la interfaz gráfica de captura de los datos del proyecto, de las historias de usuario, de la definición del “Product Backlog” y de la definición de las dependencias entre las historias de usuario. Las plantillas (Templates) desarrolladas en este componente se listan a continuación:

- Historiadependencia_form.html

- Historiadeusuario_confirm_delete.html
- Historiausuario_detail.html
- Historiausuario_form.html
- Historiausuario_list.html
- Historiausuario_upload.html
- Proyecto_confirm_delete.html
- Proyecto_detail.html
- Proyecto_form.html
- Proyecto_list.html

Cada componente del Front-end contiene un archivo llamado “URL” donde se especifican las peticiones y un archivo “Views” que es el encargado de renderizar la paginas y de comunicarse con la puerta de enlace y ejecutar los servicios necesarios para cumplir con la lógica del negocio (Ver figura 29).

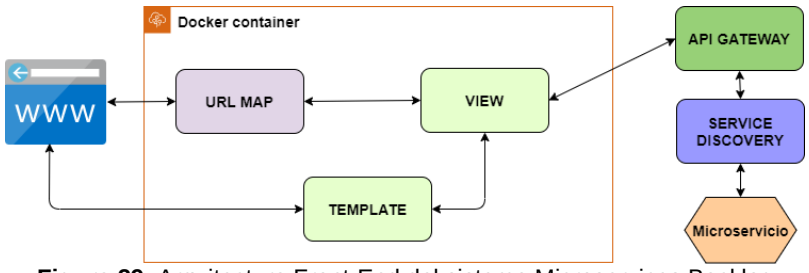


Figura 29. Arquitectura Front-End del sistema Microservices Backlog

El URL Map toma la petición del usuario desde el explorador y la direcciona a la vista correspondiente, la cual se encarga de ejecutar el servicio que requiere a través de la puerta de enlace y el descubridor de servicios, quienes envían la respuesta de nuevo a la vista para ser renderizada a través de la plantilla para poder mostrar el resultado al usuario. De esta misma forma se organizan los componentes agrupador y calculador de métricas que se detallan a continuación.

4.1.2. Componente agrupador:

El componente agrupador permite asociar las historias de usuario a los microservicios candidatos que forman una descomposición. El usuario puede generar varias descomposiciones para un proyecto, de tal forma que pueda evaluar que descomposición es mejor, en términos de acoplamiento, cohesión, similitud semántica, complejidad cognitiva, dependencias y comunicaciones. Las plantillas implementadas fueron las siguientes:

- clustering.html
- compare.html
- genetic.html
- microservicesbacklog.html
- microservicio_confirm_delete.html
- microservicio_detail.html
- microservicio_form.html
- microservicio_list.html
- microservicio_upload.html
- microservicioapp_confirm_delete.html

- microservicioapp_detail.html
- microservicioapp_form.html
- microservicioapp_list.html
- microserviciohistorias_form.html

4.1.3. Componente calculador de métricas:

El componente calculador de métricas usa el modelo matemático propuesto en la sección 4.1, para poder calcular las métricas y para evaluar y comparar las descomposiciones o aplicaciones basadas en microservicios. Consta de una sola plantilla que recibe el identificador de la aplicación y calcula todas las métricas. Esta plantilla es:

- calcularmetricas_form.html

4.1.4. Puerta de enlace (Api Gateway):

La puerta de enlace sirve como un índice que traduce las peticiones hechas del Front-End a los servicios específicos que deben ser ejecutados, también sirve como un balanceador de carga, redirige las peticiones a la instancia del servicio correspondiente. La puerta de enlace (Api Gateway) se encarga de enviar las peticiones a cada microservicio usando el descubridor de servicios.

La puerta de enlace se implementó en el servidor apache, con el módulo mod_proxy y las siguientes directivas:

- Redirect
- ProxyPass
- ProxyPassReverse

Estas directivas permiten redireccionar una petición URL a otra más específica del servidor, principalmente se realiza para evitar exponer los puertos de conexión de los servicios implementados.

4.1.5. Descubridor de servicios (Service Discovery):

El descubridor de servicios permite ejecutar un servicio determinado sin conocer la dirección física del servicio, para esto, se utiliza un balanceador de cargas que se apoyará del “Service Registry” para conocer la ubicación real del servicio.

Primero, los servicios se registrarán ante el “Service Registry”, con la finalidad de saber qué servicios se tienen disponibles, por otra parte, los clientes ejecutarán llamadas a los servicios por medio de la puerta de enlace, la cual se apoyará del Service Registry para determinar la dirección física del servicio y redireccionar la petición a la instancia correcta del servicio [1].

La lógica detrás del registro del servicio es simple, aunque la implementación de esa lógica puede ser complicada. Cada vez que se despliega un servicio, sus datos (IP y puerto como mínimo) deben almacenarse en el registro de servicio. Las cosas son un poco más complicadas cuando un servicio se destruye o se detiene. Si eso es el resultado de una acción decidida, los datos del servicio deben eliminarse del registro. Sin embargo, hay casos en que el servicio se detiene

debido a una falla y en tal situación podríamos elegir realizar acciones adicionales para restablecer el correcto funcionamiento de ese servicio [2].

En la implementación del sistema Microservice Backlog usamos Synapse [118] un sistema de descubrimiento de servicios implementado por Airbnb que permite registrar y gestionar los servicios de la aplicación.

4.1.6. Microservicios:

Los microservicios implementan la lógica de la aplicación y expone sus funcionalidades como un API de servicios REST. Se implementaron 5 microservicios: Historias de Usuario, algoritmo genético, algoritmo de agrupamiento, microservicios y métricas. El detalle de la arquitectura de implementación de los microservicios (Back-End) se puede apreciar en la figura 30.

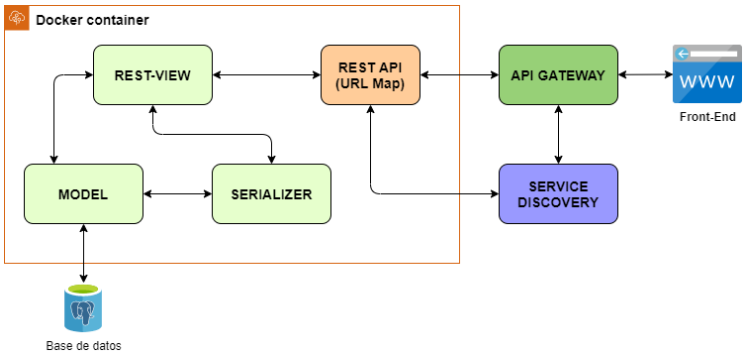


Figura 30. Arquitectura Back-End del sistema Microservices Backlog

El modelo implementa el mapeo objeto-relacional entre las entidades de la base de datos y las clases (Python) que la implementan, el serializador (SERIALIZER) se encarga de traducir los objetos / entidades al formato JSON de tal forma que se puedan enviar como un servicio REST. Las vistas Rest (Rest-View) implementan la lógica del negocio de la aplicación, toman las peticiones que le hacen al API Rest las procesa y devuelve el resultado en un formato JSON para poder ser transmitidos a sus clientes. Los microservicios implementados se detallan a continuación.

Historias de Usuario:

Este microservicio permite la creación del proyecto junto con sus historias de usuario y la definición de las dependencias entre las historias de usuario, se relaciona principalmente con el componente parametrizador, sus servicios son:

- Adicionar proyecto
- Editar proyecto
- Eliminar proyecto
- Cargar historias de usuario
- Agregar / quitar dependencias entre historias de usuario
- Listar historias de usuario del proyecto
- Listar las dependencias entre historias de usuario

Algoritmo genético:

Microservicio que permite ejecutar el algoritmo genético y obtener una descomposición de las historias de usuario en microservicios candidatos. Su único servicio público es:

- Ejecutar algoritmo

Algoritmo de agrupamiento semántico:

Este microservicio permite generar una descomposición usando la similitud semántica entre las entidades de las historias de usuario usando las ecuaciones (17) y (18) y también a partir de la distancia de acoplamiento definida por la ecuación (26). Su único servicio público es:

- Ejecutar algoritmo

Microservicios:

Este microservicio permite la gestión de las descomposiciones o aplicaciones basadas en microservicios, permite crear microservicios, asignarle historias de usuario y calcular las métricas; también permite comparar las descomposiciones generadas automáticamente por el algoritmo genético y por el algoritmo de agrupamiento contra las descomposiciones ingresadas por el usuario. Los principales servicios expuestos son:

- Adicionar descomposición o aplicación basada en microservicios MSBA
- Listar descomposiciones
- Agregar microservicios
- Editar microservicios
- Agregar / quitar historias de usuario a un microservicio
- Calcular métricas
- Evaluar descomposiciones del proyecto
- Dividir microservicio
- Unir microservicios
- Dibujar el diagrama Microservices Backlog

Métricas:

Microservicio encargado de calcular las métricas para evaluar las descomposiciones o aplicaciones basadas en microservicios, los servicios que expone son:

- Calcular métrica de granularidad Gm (ecuación 3)
- Calcular acoplamiento CpT (ecuación 12)
- Calcular cohesión CohT (ecuación 15)
- Calcular similitud semántica SsT (ecuación 18)
- Calcular mayor número de historias asociadas a un microservicio (ecuación 19)
- Calcular el promedio de llamadas Avg. Calls (ecuación 20)
- Calcular complejidad cognitiva (ecuación 23)
- Calcular tiempo estimado de desarrollo T (ecuación 25)

Para explicar y detallar el funcionamiento del sistema Microservices Backlog, en el capítulo 8, anexos, se explican y muestran cómo se obtuvieron las descomposiciones para el caso de estudio Cargo Tracking, explicado en la sección 5.4.1.

4.2. ESPECIFICACIÓN FORMAL DEL MODELO DE GRANULARIDAD

Le especificación formal corresponde a las expresiones matemáticas que permiten calcular las métricas y evaluar la función objetivo del modelo de granularidad. La especificación formal del modelo de granularidad se dará en términos de las métricas de acoplamiento, cohesión, número de historias asociadas a los microservicios, complejidad cognitiva, similitud semántica y por la métrica de granularidad (*Gm*). Adicionalmente para el proceso de evaluación se incluyen métricas de complejidad (puntos de historia), comunicación, rendimiento y el tiempo estimado de desarrollo.

El modelo matemático planteado se fundamente en la definición dada por S. Hassan y otros (2020) afirman que un nivel de granularidad determina "el tamaño del servicio y el alcance de la funcionalidad que expone". La adaptación de la granularidad implica la unión o descomposición de los microservicios, por lo que se pasa a un nivel de granularidad de grano más fino o más grueso. El grano más fino para este caso corresponde a tener una historia asociada a un solo microservicio y el grano más grueso sería la aplicación monolítica donde se tiene un solo microservicio con todas las historias de usuario asociadas.

Homay y otros (2020), afirmaron que "el problema para encontrar la granularidad de los servicios es identificar un límite correcto (tamaño) para cada servicio en el sistema. En otras palabras, cada servicio del sistema debe tener un propósito concreto, lo más desacoplado posible, y añadir valor al sistema. Un servicio tiene una granularidad correcta o buena si maximiza la modularidad del sistema y al mismo tiempo minimiza la complejidad. Modularidad en el sentido de flexibilidad, escalabilidad, mantenibilidad y rastreabilidad, mientras que complejidad en términos de dependencia, comunicación y procesamiento de datos".

De esta definición se evidencia una relación entre la granularidad de los microservicios con el acoplamiento, la modularidad y la complejidad. Al cambiar el tamaño y el alcance de los microservicios implica cambios en el acoplamiento, la modularidad y la complejidad del sistema. En este caso la granularidad corresponde a definir el número de historias de usuario asociadas a un microservicio (tamaño del servicio) y el número de microservicios que forman la aplicación (tamaño de la aplicación).

Un microservicio puede tener una muy baja granularidad (menor tamaño), pero al interactuar con otros microservicios el acoplamiento aumenta, si el acoplamiento es muy alto es una mala decisión mantener esa granularidad en un sistema basado en microservicios, entonces se debe unir con otros microservicios para reducir el acoplamiento, al unirse con otros microservicios su granularidad aumenta, porque va a tener asociado más historias u operaciones que exponer. Con el modelo propuesto, se busca determinar la granularidad adecuada de tal forma que su acoplamiento con los otros microservicios de la aplicación sea bajo, que tenga pocas dependencias y poca comunicación con otros microservicios, siendo coherente con la definición teórica dada.

La cohesión se refiere a la medida en que las operaciones de un servicio contribuyen a una y solo una tarea, es decir que se refieren al mismo propósito [41]. También un microservicio que tenga alta cohesión debe funcionar por si solo sin depender en un alto grado de otros microservicios y no tener interdependencias con los otros microservicios.

Los microservicios deben tener un propósito específico, por lo tanto, servicios / operaciones / historias que se refieren al mismo propósito deben ser agrupados en el mismo microservicio, de ahí la importancia de la similitud semántica, si las historias de usuario que se refieren a los mismo, es decir que tienen una alta similitud semántica deben ser agrupadas en el mismo microservicio, siendo así un indicador de alta cohesión. Otro punto importante que aporta la similitud semántica corresponde a las entidades las cuales se traducen en tablas de la base de datos, si se agrupan historias de usuario / operaciones que se refieren a la misma entidad entonces se reduce la redundancia y duplicación de información, partiendo del hecho que cada microservicio tiene su propia base de datos independiente y solo se accede a los datos a través de su interfaz del servicio.

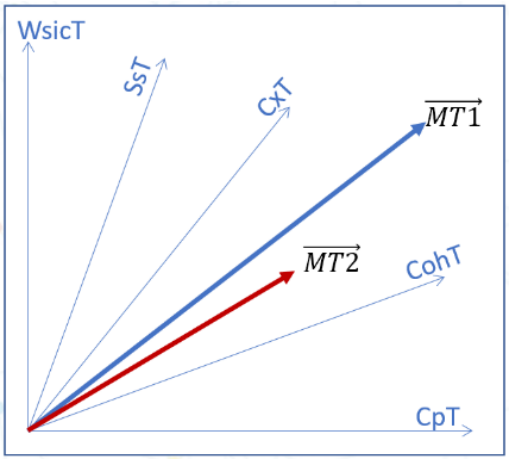


Figura 31. Fundamento del modelo matemático: Los vectores.

El modelo matemático propuesto se basa en vectores (Ver figura 31), en la figura aparecen 2 vectores en un espacio de 5 dimensiones, cada vector representa una descomposición de las historias de usuario en microservcios. MT es un vector de 5 dimensiones (1. Acoplamiento (CpT), 2. Falta de cohesión (CohT), 3. Granularidad (WsicT), 4. Complejidad (CxT), 5. similitud semántica (SsT)), donde cada dimensión corresponde a una métrica que directa o indirectamente es afectada por la granularidad de los microservicios que forman la aplicación. Por lo tanto, se define una aplicación basada en microservicios (*MSBA)* como:

$$MSBA = (MS, \overrightarrow{MT}) \quad (1)$$

Donde *MS* es un conjunto de microservicios, *MS = {ms₁, ms₂, … msₙ}* y *MT* es un vector que contiene de las métricas calculadas para *MSBA*. En este caso, las métricas calculadas y utilizadas para MSBA corresponden al acoplamiento (*CpT*), la cohesión (*CohT*), el número de historias asociadas al microservicio (*WsicT*), los puntos de complejidad cognitiva (*CxT*), y la similitud semántica (*SsT*). Así:

$$\overrightarrow{MT} = [\, CpT, CohT, WsicT, CxT, (100 - SsT)] \quad (2)$$

Donde *SsT* corresponde al valor de la similitud semántica obtenida por la librería Spacy, de la cual se obtiene un valor entre cero y uno, entre más cercano a uno sea, mayor similitud semántica tiene; para este modelo se amplificó en un número entre 0 y 100 (porcentaje) de tal forma que la dimensión de esa variable sea similar a la dimensión de las otras variables. Se incluyó *(100 – SsT)* para invertir su relación, teniendo mayor similitud semántica cuando sea cercano a 0, para

que de esta forma se pueda minimizar luego la norma de $\overrightarrow{MT}$ en los cálculos de *Gm* en la función objetivo del algoritmo genético. Por lo tanto, definimos *Gm* de la siguiente forma:

$$Gm = \left|\overrightarrow{MT}\right| = \sqrt[2]{(Cpt^2 + CohT^2 + CxT^2 + WsicT^2 + (100 - SsT)^2} \qquad (3)$$

Esta expresión matemática permite determinar cuán buena o mala es la descomposición o la granularidad de *MSBA*; un menor valor de *Gm* implica una buena granularidad. El objetivo es obtener una solución de baja complejidad, bajo acoplamiento, baja falta de cohesión (lack of cohesion), pequeño *WsicT* y alta similitud semántica (*SsT*).

En los vectores mostrados en la figura 28 se puede apreciar que $Gm_1 > Gm_2$. Para que la norma del vector $\overrightarrow{MT1}$ (Gm_1) sea mayor que la norma del vector $\overrightarrow{MT2}$ (Gm_2) es porque alguna de las dimensiones de $\overrightarrow{MT1}$ son mayores que las dimensiones de $\overrightarrow{MT2}$, asi la solución que representa $\overrightarrow{MT1}$ tiene mayor acoplamiento, mayor falta de cohesión, mayor complejidad o menor similitud semántica. Por lo tanto, Gm_2 representa una mejor solución, una mejor distribución de las historias de usuario en microservicios, y una mejor granularidad de los microservicios.

Los microservicios tienen asociadas historias de usuarios y métricas, entonces:

$$ms_i = (HU_i, MTS_i) \qquad (4)$$

Donde ms_i es el i-ésimo microservicio, HU_i es el conjunto de historias de usuario asociadas al i-ésimo microservicio, entonces $HU_i = \{hu_1, hu_2, \ldots, hu_m\}$. MTS_i es un conjunto de métricas calculadas para ms_i.

4.2.1. Acoplamiento de MSBA.

El acoplamiento mide el grado de dependencia de un componente de software en relación con otro. En este caso el acoplamiento se definió por tres métricas: 1) importancia absoluta del microservicio (*AIS*), 2) dependencia absoluta del microservicio (*ADS*), y 3) interdependencia de los microservicios (*SIY*). Estas métricas se calculan con base a las dependencias de las historias de usuario para cada microservicio.

Importancia absoluta del microservicio (AIS): AIS es el número de otros microservicios que invocan al menos una operación de la interfaz de un microservicio [103]. AIS_i es el número de clientes (otros microservicios) que invocan al menos una operación de ms_i. A nivel de sistema, se define el vector $\overrightarrow{AIS}$, que contiene el valor *AIS* calculado para cada microservicio. Para calcular el valor total de *AIS* a nivel de sistema (*AisT*), se calcula la norma del vector $\overrightarrow{AIS}$. Donde *n* es el número de microservicios de *MSBA*, por lo tanto:

$$\overrightarrow{AIS} = [AIS_1, AIS_2, \ldots, AIS_n] \qquad (5)$$

$$AisT = \left|\overrightarrow{AIS}\right| = \sqrt[2]{{AIS_1}^2 + {AIS_2}^2 + \ldots + {AIS_n}^2} \qquad (6)$$

Dependencia absoluta del microservicio (ADS): ADS el número de otros microservicios de los que depende el microservicio. ADS_i es el número de microservicios de los cuales se invoca al menos una operación de ms_i [103]. Para calcular el valor total de *ADS* a nivel de sistema *(AdsT)*, se calcula la norma del vector *ADS*. Entonces:

$$\overrightarrow{ADS} = [ADS_1, ADS_2, \ldots, ADS_n] \quad (7)$$

$$AdsT = \left|\overrightarrow{ADS}\right| = \sqrt[2]{ADS_1{}^2 + ADS_2{}^2 + \ldots + ADS_n{}^2} \quad (8)$$

Interdependencia de los microservicios (SIY): Número de pares de microservicios interdependientes [103]. En este caso SIY_i define el número de pares de microservicios que dependen bidireccionalmente uno del otro dividido por el número total de microservicios. A nivel de sistema, se definió el vector *SIY*:

$$\overrightarrow{SIY} = [SIY_1, SIY_2, \ldots, SIY_n] \quad (9)$$

$$SiyT = \left|\overrightarrow{SIY}\right| = \sqrt[2]{SIY_1{}^2 + SIY_2{}^2 + \ldots + SIY_n{}^2} \quad (10)$$

Se definió el vector $\overrightarrow{Cp}$ como el vector que contiene las métricas de acoplamiento calculadas para *MSBA*, calculando la norma del vector $\overrightarrow{Cp}$ tenemos el valor de acoplamiento para la aplicación *(CpT)*:

$$\overrightarrow{Cp} = [AisT, AdsT, SiyT] \quad (11)$$

$$CpT = \left|\overrightarrow{Cp}\right| = \sqrt[2]{AisT^2 + AdsT^2 + SiyT^2} \quad (12)$$

En la figura 32 se ilustra el cálculo de las métricas de acoplamiento para un caso hipotético en el cual hay 3 microservicios que forman *MSBA*. De la siguiente forma ms_1 = {hu_1, hu_2}, ms_2 = {hu_3} y ms_3={hu_4}. Donde hu_1 tiene como dependencias a {hu_3, hu_4}, hu_2 a {hu_4}, hu_3 a {hu_1} y hu_4 no tiene dependencias.

Para este caso la métrica *AIS* del ms_1 *(*AIS_1*)* se calcula contando el número de otros microservicios que llaman a ms_1, en este caso sería uno solo *(*ms_2*)*, ya que *hu3* llama o ejecuta a hu_1. De la misma forma para los otros dos microservicios *AIS* es 1, porque cada uno de ellos sólo tiene un cliente. Para calcular *ADS*, se cuenta el número de microservicios a los cuales llama ms_i, en este caso ADS_1 sería 2, ADS_2 sería 1 y ADS_3 sería 0 ya que no llama a ningún otro microservicio. Ahora *SIY*, ms_1 tiene un microservicio interdependiente, al igual que ms_2, mientras que ms_3 no tiene microservicios interdependientes, entonces SIY_1 = 1/3, SIY_2 = 1/3 y SIY_3 =0. Así $\overrightarrow{Cp} = [1.7, 2.2, 0.47]$ al calcular la norma de ese vector se obtiene *CpT* = 2.82, que corresponde al valor del acoplamiento de *MSBA* para ese caso específico.

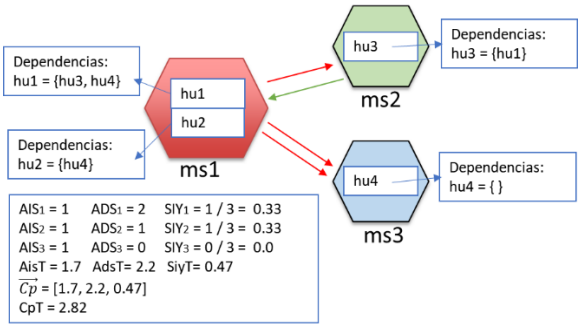


Figura 32. Ejemplo del cálculo del acoplamiento de MSBA

4.2.2. Cohesión de MSBA.

La cohesión y el acoplamiento son dos propiedades contrastantes. Una solución que equilibre la alta cohesión y el bajo acoplamiento es el objetivo de los desarrolladores cuando implementan aplicaciones basadas en microservicios. Utilizamos las siguientes métricas de cohesión:

Falta de cohesión (LC): LC mide el número de pares de microservicios que no tienen ninguna dependencia entre ellos, adaptado de [102]. *LC* de ms_i fue definida como el número de pares de microservicios que no tienen ninguna interdependencia con ms_i.

Grado de la falta de cohesión (Coh): El grado de cohesión *Coh* de cada microservicio se define como la proporción de la métrica de falta de cohesión dividida por el número total de microservicios que forman parte de la aplicación; de la siguiente forma:

$$Coh_i = LC_i / n \quad (13)$$

Donde *n* es el número de microservicios. A nivel de sistema, se definió el vector $\overrightarrow{Coh}$, calculando la norma del vector $\overrightarrow{Coh}$ tenemos el valor de cohesión para *MSBA (CohT)*:

$$\overrightarrow{Coh} = [Coh_1, Coh_2, \ldots, Coh_n] \quad (14)$$

$$CohT = \left|\overrightarrow{Coh}\right| = \sqrt[2]{Coh_1{}^2 + Coh_2{}^2 + \ldots + Coh_n{}^2} \quad (15)$$

Siguiendo con el ejemplo anterior, la figura 33 detalla la forma de calcular la cohesión del MSBA.

La métrica falta de cohesión *LC* corresponde al número de microservicios con los cuales ms_i es bidireccional, para ms_1 corresponde a 1, para ms_2 a 1 también y para ms_3 a 0, ningún otro microservicio tiene interdependencia con ms_3. La interdependencia es un mal indicador de la solución propuesta porque implica mayor complejidad al mantener y desarrollar los microservicios, por lo tanto, debe ser reducida a cero. Luego calculamos el grado de cohesión Coh_1 y Coh_2 corresponden a 1/3 y Coh_3 a 2/3 y por lo tanto el vector $\overrightarrow{Coh} = [0.33, 0.33, 0.67]$, calculando la norma del vector obtenemos el valor de la cohesión *CohT = 0.82*

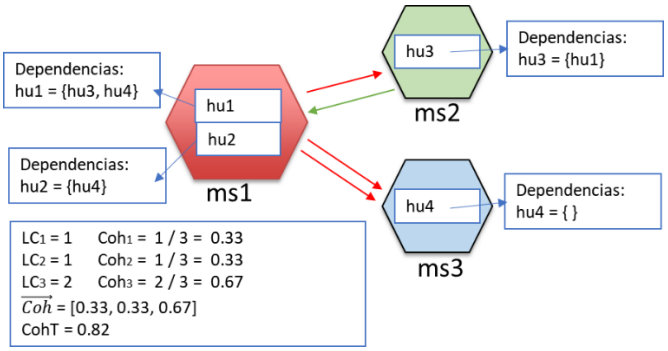


Figura 33. Ejemplo del cálculo de la cohesión de MSBA

Similitud semántica (SsT) de MSBA: Según Cojocaru y otros [70] "la similitud semántica utiliza algoritmos de evaluación de la distancia léxica para señalar los servicios que contienen componentes no relacionados o acciones no relacionadas que obstaculizan la cohesión". Utilizamos una librería de procesamiento de lenguaje natural llamada Spacy [119] para calcular la similitud semántica entre las historias de usuarios; cuya similitud se determina comparando vectores de palabras o "incrustaciones de palabras", las cuales son representaciones de

significado multidimensional de una palabra [120]. Se calculó la similitud semántica entre cada historia de usuario, uniendo su nombre y su descripción. Se calculó *SsT* de la siguiente manera:

1. Seleccionamos los sustantivos (entidades) del nombre y la descripción de la historia del usuario.
2. Identificamos los lemas del sustantivo en cada historia de usuario.
3. Definimos un diccionario que contiene los valores de similitud semántica entre las historias de usuarios como sigue:

$$DSS = \{< "hu_1 - hu_2", a_{1-2} >, < "hu_1 - hu_3", a_{1-3} >, \ldots, < "hu_j - hu_k", a_{j-k} >\} \quad (16)$$

Donde:
DSS: Diccionario de similitud semántica entre las historias de usuario.
hu_i y hu_j son los identificadores de las historias de usuario que se están comparando.
''hu_j - hu_k*''* corresponde a la llave del diccionario, la cual se forma concatenando los identificadores de las historias de usuario que se comparan.
a_{j-k} corresponde al valor del diccionario, son los valores de similitud semántica entre las historias hu_j y hu_k obtenidos por Spacy, Son números reales entre 0 y 1. Entre más cercano a uno sea el valor, más similares son las historias de usuario.

4. Se calculó la similitud semántica *(SS)* del i-ésimo microservicio como el promedio de los valores de similitud semántica entre sus historias de usuarios. La similitud semántica total de *MSBA* fue el promedio de la similitud semántica de cada microservicio. Para obtener un valor de similitud semántica entre 0 y 100 multiplicamos el promedio por 100.

$$SS_i = \frac{1}{c}\sum_{j=1,k=j+1}^{m} a_{j-k} \quad (17)$$

$$SsT = \frac{100}{n}\sum_{i=1}^{n} SS_i \quad (18)$$

Donde:
a_{j-k} corresponden a los valores de similitud semántica de la comparación realizada entre las historias de usuarios de ms_i.
c corresponde al número de comparaciones realizadas para calcular *SS.*
n corresponde al número de microservicios de *MSBA.*

4.2.3. Granularidad de MSBA (WsicT).

La granularidad corresponde al tamaño de cada microservicio y al tamaño de la aplicación. Utilizamos las métricas de granularidad que se indican a continuación.

Número de microservicios (n): El número de microservicios que forman parte del sistema o aplicación *MSBA*.

Recuento ponderado de la interfaz de servicios (WSIC): El WSIC es el número de operaciones de interfaz expuestas del microservicio ms_i [110], se pondera dividiendo el número de operaciones por el número de parámetros o el número de microservicios. Para nuestro modelo, una historia de usuario está relacionada con una operación (uno a uno); por lo tanto, adaptamos esta métrica como el número de historias de usuario asociadas con el microservicio ms_i. Otros autores llamaron a esta métrica como el número de operaciones. Definimos *WsicT* como el

número máximo de historias de usuario asociadas a un microservicio, por lo que *WsicT* es el máximo *WSIC* de *MSBA*, entonces:

$$WsicT = Max(WSIC_1, WSIC_2, \ldots, WSIC_n) \qquad (19)$$

4.2.4. Métricas de rendimiento de MSBA.

Medir o estimar el rendimiento de una aplicación en tiempo de diseño es difícil e impreciso. Usamos las llamadas y solicitudes entre microservicios para estimar el rendimiento. Asumimos que, si hay más llamadas y solicitudes entre los microservicios, entonces la comunicación, la latencia y el tiempo de respuesta de la aplicación se incrementa, por lo tanto, el rendimiento de la aplicación se ve directamente afectado. Lo ideal en una aplicación basada en microservicios sería tener microservicios que no tengan comunicación entre ellos y que trabajen de forma independiente. Por lo tanto, definimos tres métricas:

Llamadas de un microservicio ($calls_i$): Las llamadas corresponden al número de invocaciones de ms_i a otros microservicios.

Solicitudes de un microservicio ($request_i$): Las solicitudes corresponde al número de invocaciones de otros microservicios a ms_i.

Promedio de llamadas de MSBA: Corresponde a la suma de las llamadas entre los microservicios dividida entre el número de microservicios (*n*) de *MSBA*. Entonces:

$$Avg.Calls = \frac{1}{n} \sum_{i=1}^{n} Calls_i \qquad (20)$$

La figura 34 ilustra el cálculo de estas métricas para el ejemplo anterior.

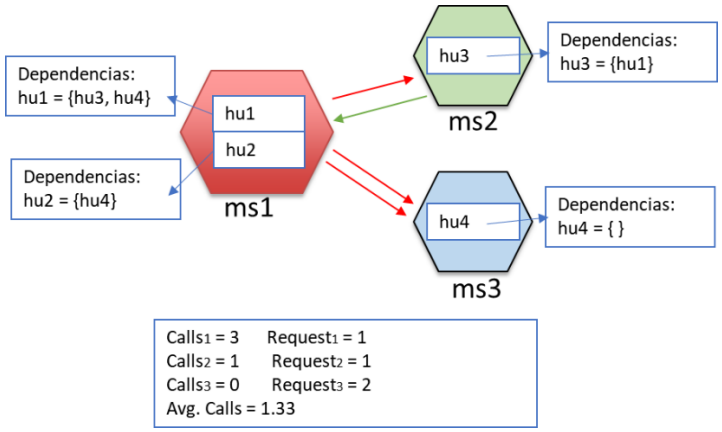


Figura 34. Ejemplo del cálculo del "Avg. Calls" de MSBA

4.2.5. Complejidad de MSBA.

Medir la complejidad es fundamental para desarrollar aplicaciones basadas en microservicios. Si la complejidad es alta, el costo de cambio es mayor, también el costo de implementación y el tiempo de desarrollo se incrementan. Las métricas de complejidad utilizadas se detallan a continuación.

Puntos de historia de usuario del microservicio *(P_i)*: Los puntos de historia de usuario son un punto estimado del esfuerzo necesario para desarrollar la historia del usuario. Los puntos son un indicador de la velocidad del equipo de desarrollo; por lo tanto, definimos P_i como la sumatoria de los puntos de historia de usuario de ms_i de la siguiente manera:

$$P_i = \sum_{j=1}^{m} PH_j \qquad (21)$$

Donde:
P_i es el total de puntos de historia asociados a ms_i.
m es el número de historias de usuario asociadas al microservicio ms_i.
PH_j corresponde a los puntos de historia estimados por el equipo de desarrollo para j-ésima historia de usuario de ms_i.

Puntos de complejidad cognitiva de MSBA (CxT): Se añadieron puntos según la complejidad del microservicio, sus relaciones y dependencias. Se estimó la dificultad de desarrollar y mantener una aplicación basada en microservicios. Se planteó de acuerdo con los siguientes postulados:

- Se estima la dificultad de desarrollar, mantener y entender una aplicación basada en microservicios.
- El punto de partida fue la estimación de los puntos de historia usuario que realiza el equipo de desarrollo.
- Se agregan puntos de acuerdo con la complejidad del microservicio, sus relaciones y dependencias.
- Se basa en la complejidad de un grafo y su profundidad.
- Se tiene en cuenta un caso base, que corresponde a la menor complejidad; este caso sería una *MSBA* con un solo microservicio, con una sola historia de usuario y un punto de historia estimado para su desarrollo, que sería el caso más simple de desarrollar. Para este caso $Cx_0 = 2$.
- Corresponde al número de veces que *MSBA* es más compleja en relación con el caso base.

Los puntos cognitivos se incrementan de acuerdo con:

- Los puntos estimados totales para cada microservicio que hace parte de la aplicación.
- El número de microservicios que hacen parte de la aplicación.
- El número de historias de usuario u operaciones asociadas a cada microservicio.
- El número de llamadas - calls (out) y peticiones – request (in) del microservicio.
- La profundidad del número de llamadas que hace un microservicio a otros microservicios. Corresponde al número de nodos consecutivos usados en la llamada de un microservicio a otro.

Formalmente *CxT* se definió de la siguiente manera:

$$Cx = \left(\left(\sum_{i=1}^{n} Cg_i \right) + Max(P_1, \dots, P_n) + (n * WsicT) + \left(\sum_{i=1}^{n} Pf_i \right) + \left(\sum_{i=1}^{n} LC_i \right) \right) \qquad (22)$$

$$CxT = \frac{Cx}{Cx_0} \qquad (23)$$

Donde:
i = *i-esimo* microservicio
$Cg_i = P_i * (Calls_i + Request_i)$

P_i = Número total de historias de Usuario del i-esimo microservicio (Suma de los puntos de historia de Usuario de cada historia de usuario asociada al microservicio).
Max(P_1, …, P_n): Máximo P_i de *MSBA*.
n = número de microservicios de *MSBA*.
WsicT: Mayor *WSIC* de la aplicación, definida en la ecuación (18).
Pf_i: Número de nodos usados secuencialmente desde una llamada que realiza un microservicio a otros microservicios, contados desde el i-ésimo microservicio; Una mayor profundidad implica una mayor complejidad de implementación y mantenimiento de la aplicación.
LC: Interdependencia del microservicio, definida en la sección 4.1.1.
Cx_0: El caso base en el que la aplicación tiene un microservicio, una historia de usuario con un punto de historia estimado. Entonces $Cg_1 = 0$, $Max(P_1) = 1$, $n=1$, $WsicT=1$, $Pf_1 =0$, $SIY=0$, y $Cx = 2$. Por lo tanto, $Cx_0 = 2$.

La figura 35 presenta los ejemplos de cálculo de la complejidad cognitiva para algunos casos ilustrativos; caso 1: el caso base, caso 2: aplicación con un microservicio con una historia y un punto estimado de historia, caso 3: 1 microservicio con 10 historias de usuario cada una con 1 punto de historia estimado.

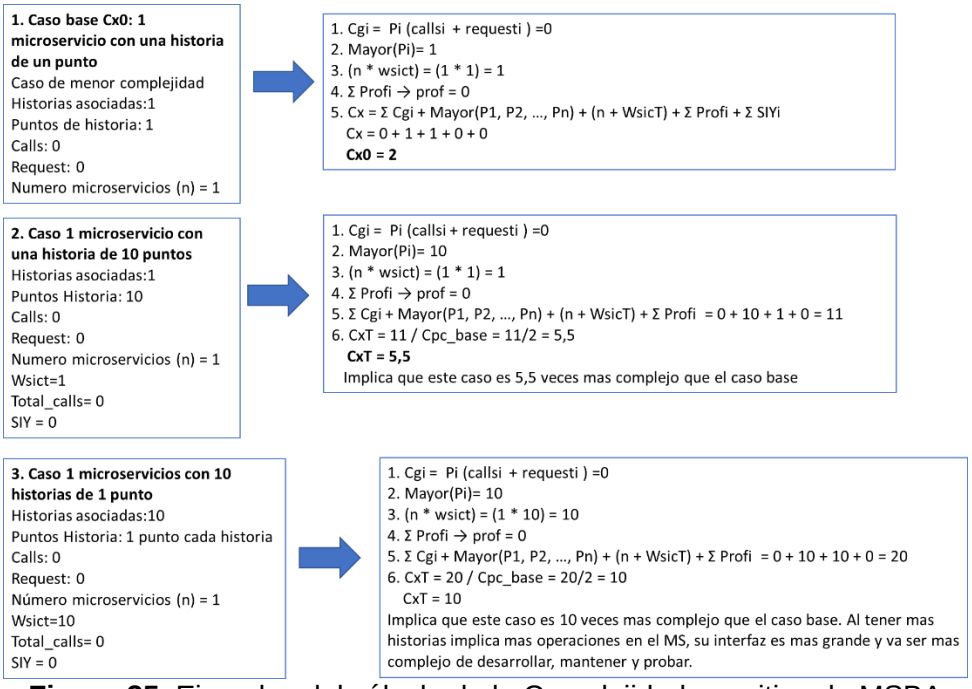


Figura 35. Ejemplos del cálculo de la Complejidad cognitiva de MSBA

4.2.6. Tiempo estimado de desarrollo de MSBA (T).

Los microservicios se implementan y organizan en torno a las capacidades del negocio, idealmente cada uno es administrado por un equipo de desarrollo independiente. Para las evaluaciones realizadas con este modelo se ha supuesto que cada microservicio se desarrolla en paralelo y de forma independiente, por lo que el tiempo de desarrollo estimado de la aplicación corresponde al tiempo de desarrollo estimado más largo de los microservicios que forman parte de la aplicación. En la vida real esto no es del todo cierto, un equipo de desarrollo podría tener a cargo varios microservicios y varios microservicios se pueden desarrollan de forma secuencial; esta restricción se considerará en futuros trabajos.

El equipo de desarrollo estima los puntos de historia de usuario y el tiempo de desarrollo en la planificación del lanzamiento. Muchas empresas de desarrollo de software definen una escala de conversión de los puntos de historia de usuario a tiempo de desarrollo (horas). Se considera que el tiempo de desarrollo estimado de las historias de usuario son datos de entrada de esta técnica. Definimos dos métricas de evaluación de la siguiente forma.

El tiempo de desarrollo de cada microservicio *(t_i)*: El tiempo de desarrollo del microservicio corresponde a la suma del tiempo de desarrollo estimado de cada historia de usuario que forma parte del microservicio.

$$t_i = \sum_{j=1}^{m} TH_j \qquad (24)$$

Donde:
t_i es el tiempo estimado de desarrollo de ms_i.
m es el número de historias de usuario de ms_i.
TH_j es el tiempo estimado de desarrollo de la j-ésima historia de usuario de ms_i. Es un dato de entrada.

Tiempo de desarrollo de la aplicación (T): Corresponde al mayor tiempo de desarrollo estimado de los microservicios que forman parte de la aplicación.

$$T = Max(t_1, t_2, \ldots, t_n) \qquad (25)$$

4.3. COMPONENTE PARAMETRIZADOR

Se encarga de tomar los datos de entrada y convertirlos en un formato que pueda ser procesado por el componente agrupador. Extrae los datos clave, como el identificador, el nombre, la descripción, los puntos estimados, el tiempo estimado, el escenario, las observaciones y las dependencias de la historia del usuario. Más tarde, con estos datos, el "Microservices Backlog" puede agrupar las historias de usuarios en microservicios y calcular las métricas enunciadas en la sección 4.1. El formato de las historias de usuarios es un archivo CVS donde se suministran los datos clave y se cargan al sistema. Este componente cuenta con tres actividades principales que son realizadas por el arquitecto de software o el equipo de desarrollo 1) Adicionar y listar proyectos, 2) Cargar o definir las historias de usuario y 3) Definir las dependencias entre las historias de usuario.

4.3.1. Adicionar y listar proyectos

Un proyecto en el Microservice Backlog corresponde a una aplicación que se quiera implementar usando la arquitectura basadas en microservicios, para el proyecto se especifican las historias de usuario y el Product Backlog que se va a implementar. Luego el usuario puede generar distribuciones o descomposiciones de las historias de usuario a microservicios, ya sea generadas automáticamente por el sistema o ingresadas manualmente por el usuario; para cada descomposición creada o generada el Microservice Backlog calcula las métricas y se presentan datos comparativos para ser analizadas por el arquitecto o equipo de desarrollo, también presenta un grafo con las dependencias de la solución propuesta y candidata a implementar.

La figura 36 muestra la gestión de proyectos realizada por el Microservices Backlog. Al adicionar un nuevo proyecto el usuario ingresa el nombre, la abreviatura, una descripción y el lenguaje en el que van a estar escritas las historias de usuario; la aplicación implementada soporta español e

inglés como lenguaje de escritura de las historias de usuario. Para cada proyecto se pueden definir varias descomposiciones o aplicaciones basadas en microservicios para que el arquitecto o equipo de desarrollo pueda comparar diferentes distribuciones de las historias de usuario en microservicios, comparar las métricas y el Microservices Backlog. (La sección 4.4 explica la gestión de descomposiciones o aplicaciones basadas en microservicios de un proyecto)

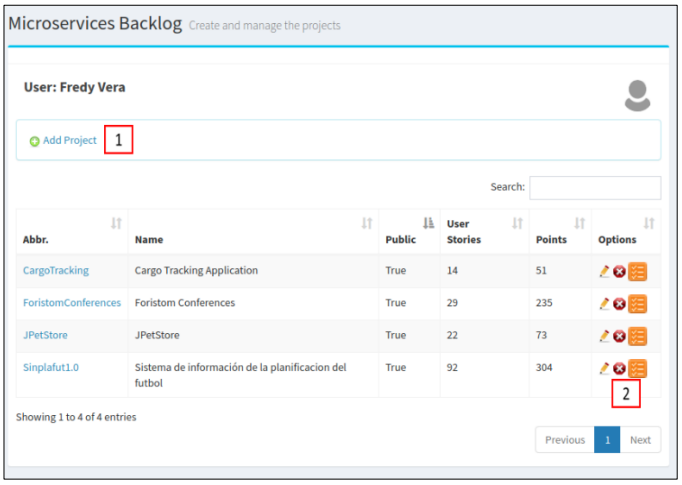


Figura 36. Componente parametrizador, Agregar proyectos.

En 1) aparece la opción de adicionar un nuevo proyecto y en 2) Aparecen las opciones de modificar y eliminar los datos proyecto, también la opción de gestionar las historias de usuario del proyecto.

4.3.2. Gestionar historias de usuario

Para cada proyecto el usuario debe subir o crear las historias de usuario que se quieren agrupar en microservicios. Este proceso, el usuario lo puede realizar una por una por medio de la opción de agregar historia de usuario, o también puede cargar todas las historias de usuario de una vez, a través de un archivo plano en formato CVS. Ver figura 37.

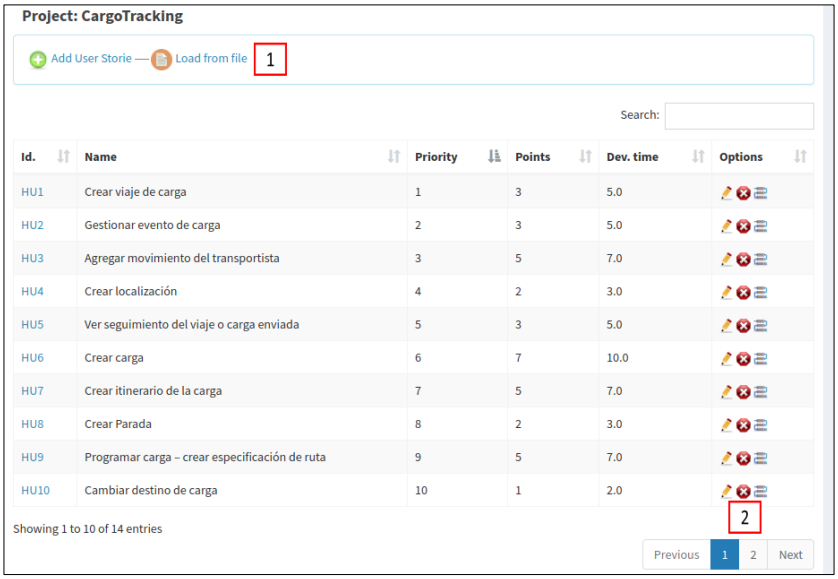


Figura 37. Microservices Backlog, Gestión de historias de usuario.

En 1) aparecen las opciones de agregar historia de usuario a través de un formulario o a través de un archivo plano CVS y en 2) Aparecen las opciones de modificar y eliminar los datos de la historia de usuario, también la opción de definir las dependencias entre las historias de usuario. La figura 37 corresponde al “product backlog” para el proyecto ingresado al sistema. Luego de crear las historias de usuario se deben definir las dependencias.

4.3.3. Dependencias entre las historias de usuario

Este paso es un punto crítico del modelo propuesto, a partir de las dependencias se definen las métricas y los algoritmos de agrupamiento. Es un proceso realizado por el usuario (arquitecto o equipó de desarrollo), debe determinar las dependencias entre las historias de usuario que hacen parte del proyecto. Por definición una historia de usuario es atómica, indivisible y muchas veces se implementa como un método que hace parte de una clase de la aplicación. El método usado para determinar las dependencias se detalla a continuación:

1. Definir el escenario de la historia de usuario, el escenario determina el algoritmo que detalla el funcionamiento de la historia de usuario.
2. Analizar las dependencias de datos entre las historias de usuario. Revisar el diagrama de datos y verificar dependencias.
3. Identificar las dependencias de invocación entre historias de usuario.
4. Analizar las posibles transacciones entre las historias de usuario. Si se identifica una transacción se debe incluir dependencias entre las historias que hacen parte de la transacción.
5. En las migraciones de aplicación monolítica a microservicios, se puede analizar el código fuente de la aplicación, o los logs de ejecución entre las operaciones (métodos) que implementan las funcionalidades de la aplicación. Se puede relacionar una operación a una historia de usuario, definir las dependencias a partir de las invocaciones entre las operaciones. Normalmente las historias de usuario se implementan como una operación.
6. Una historia de usuario no puede tener una dependencia con ella misma.

En la figura 38 se muestra la interfaz de captura de las dependencias entre las historias de usuario de la aplicación en el sistema Microservices Backlog.

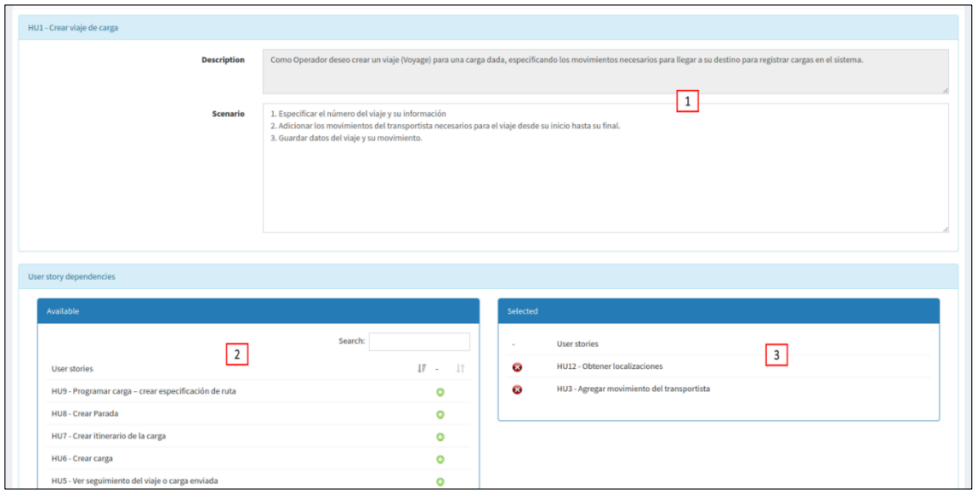


Figura 38. Microservices Backlog, definición de dependencias entre las historias de usuario.

En 1) aparece la descripción de la historia de usuario y el escenario registrado. 2) Historias de usuario disponibles. 3) Historias de usuario asignadas como dependencias.

Mas adelante en el capítulo 5 se aborda con más detalle la definición de las dependencias en cada uno de los casos de estudio.

4.4. TÉCNICAS DE AGRUPAMIENTO – COMPONENTE AGRUPADOR

Este componente agrupa las historias de usuario en microservicios. El usuario o arquitecto pueden agregar y generar descomposiciones automáticas de las historias de usuario en microservicios (utilizando un algoritmo genético, o un algoritmo de agrupamiento), o crear la descomposición manualmente.

El sistema que implementa el modelo del Microservice Backlog contiene las funcionalidades necesarias para ingresar las descomposiciones (aplicaciones) que el usuario requiera para hacer sus comparaciones; antes de ingresar los detalles de cada microservicio debe crear una aplicación para el proyecto seleccionado, especificar el método de generación (Algoritmo de agrupamiento, algoritmo genético o manual), su descripción y su nombre. La figura 39 presenta la interfaz de gestión de aplicaciones.

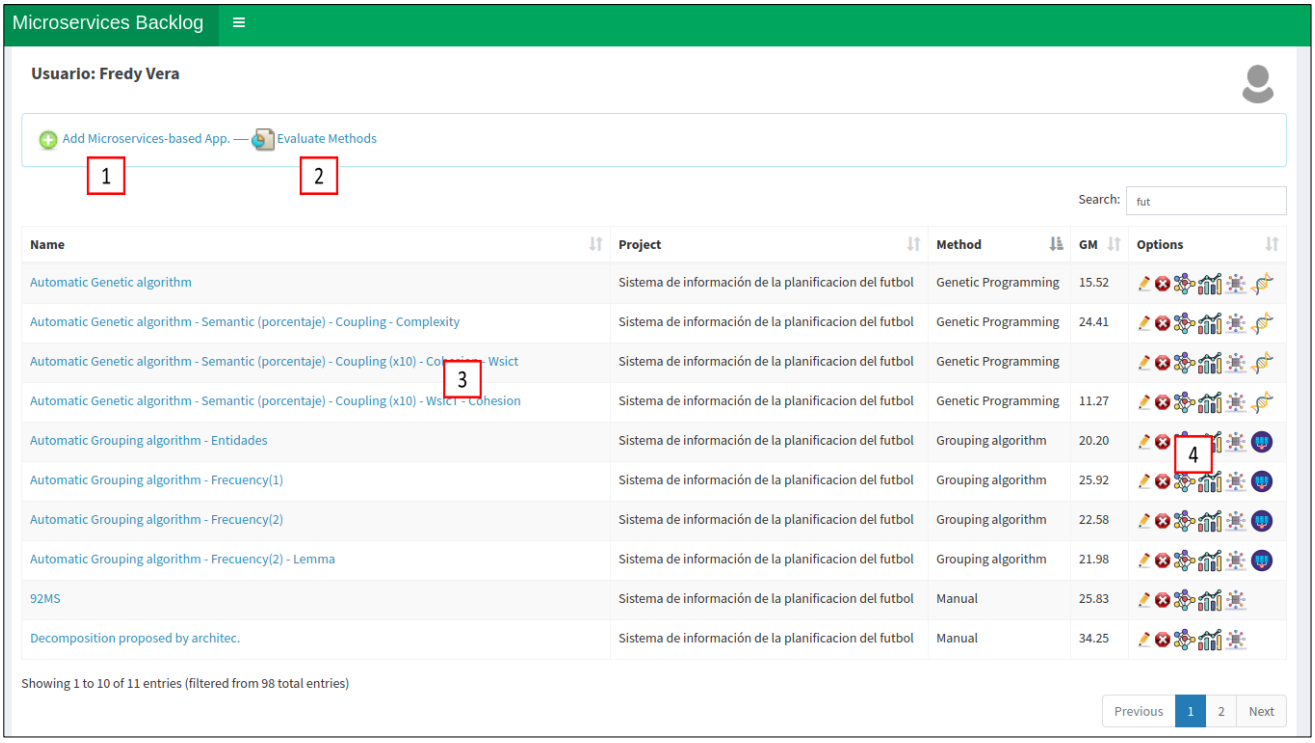


Figura 39. Microservices Backlog, gestión de aplicaciones o descomposiciones.

En 1) Opción para agregar una nueva aplicación o descomposición. 2) Opción que genera un análisis comparativo entre las aplicaciones obtenidas para un proyecto. 3) Aplicaciones o descomposiciones generadas para un proyecto. 4) Funcionalidades disponibles para cada aplicación como son (en orden de izquierda a derecha): editar, eliminar, gestionar los microservicios, evaluador de métricas, diagrama de dependencias (Microservices Backlog) de la aplicación, por último, las opciones de generar la descomposición usando el algoritmo genético o el algoritmo de agrupamiento.

Para un proyecto el usuario puede crear cualquier número de aplicaciones, cada aplicación contiene una distribución diferente de las historias de usuario en microservicios, y diferente número de microservicios. Luego de generar las descomposiciones las puede comparar y evaluar gráficamente por medio de un grafo de dependencias y las métricas correspondientes tanto a nivel de aplicación como para cada microservicio, ese diagrama corresponde al Microservice Backlog que es la propuesta principal de esta tesis doctoral.

4.4.1. Crear descomposición manualmente

El usuario (arquitecto o equipo de desarrollo) puede ingresar su propia solución al problema, o ingresar alguna generada por otro método de descomposición; para esto debe determinar el número de microservicios que van a hacer parte de la aplicación y crear cada microservicio especificando su nombre y descripción. En la figura 40 se puede apreciar la interfaz para gestionar los microservicios de una aplicación.

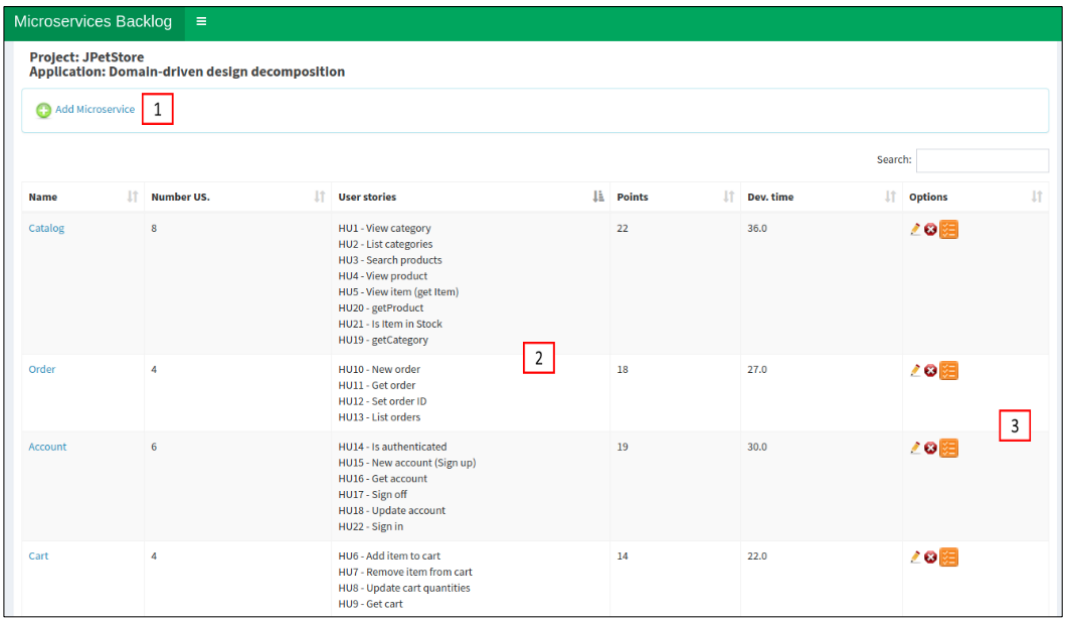


Figura 40. Microservices Backlog, Gestión de microservicios.

En 1) agregar un nuevo microservicio. 2) Detalles de los microservicios que hacen parte de la aplicación (nombre, número de historias de usuario, puntos estimados totales y el tiempo estimado de desarrollo del microservicio). 3) Opciones a ejecutar sobre cada microservicio, eliminar microservicio, editar su nombre o descripción y agregar / quitar historias de usuario.

La interfaz de agregar y quitar historias de usuario del microservicio es muy similar a la interfaz de relacionar las dependencias entre las historias de usuario. Esta interfaz se puede apreciar en la figura 41. Una historia de usuario solo puede ser asignada a un solo microservicio. Un Microservicio puede tener una o varias historias de usuario. Después de relacionar las historias de usuario y los microservicios se pueden calcular las métricas a través del componente calculador de métricas y obtener el Microservices Backlog que detalla las métricas y dependencias de los microservicios candidatos.

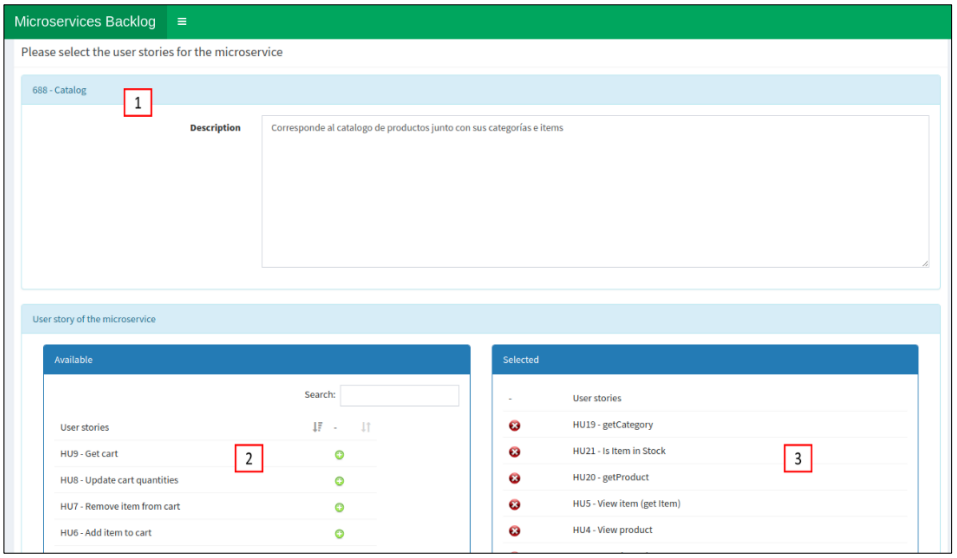


Figura 41. Microservices Backlog, Gestión de microservicios.

En 1) aparece el nombre y la descripción del microservicio. En 2) aparecen las historias de usuario disponibles para agregar al microservicio y en 3) aparecen las historias de usuario que han sido agregadas al microservicio.

4.4.2. Algoritmo genético

El algoritmo genético busca encontrar la mejor combinación, la mejor asignación de historias de usuario a los microservicios de tal manera que la función objetivo sea menor. Los algoritmos genéticos fueron establecidos por Holland [121], es iterativo, en cada iteración se seleccionan los mejores individuos, cada uno tiene un cromosoma, que se cruza con otro individuo para generar la nueva población (reproducción), se generan algunas mutaciones para encontrar la solución óptima al problema [122]. El algoritmo genético propuesto consiste en distribuir o asignar historias de usuarios a los microservicios de forma automática, teniendo en cuenta las métricas de acoplamiento, cohesión, granularidad, complejidad y similitud semántica. Diseñamos el algoritmo genético de la siguiente manera.

La figura 42 detalla el diseño del algoritmo genético implementado en el Microservices Backlog para obtener la distribución optima de historias de usuario en microservicios.

Inicialmente se asumen que hay *n* microservicios y que las *n* historias de usuario pueden ser asignadas a ellos, por lo tanto, la granularidad más fina sería que cada posible microservicio contenga solo una historia de usuario, de la misma forma la granularidad más gruesa seria si todas las historias de usuario queden asignadas en un solo microservicio, este caso sería la aplicación monolítica. A continuación, se explican los métodos del algoritmo genético.

Método para obtener la población inicial.

Se tiene un conjunto de historias de usuarios $HU = \{hu_1, hu_2, hu_3, ..., hu_m\}$, que deben ser asignadas a los microservicios. Tenemos un conjunto de microservicios $MS = \{ms_1, ms_2, ms_3, ..., ms_n\}$ y algunas métricas calculadas a partir de la información contenida en la historia de usuario.

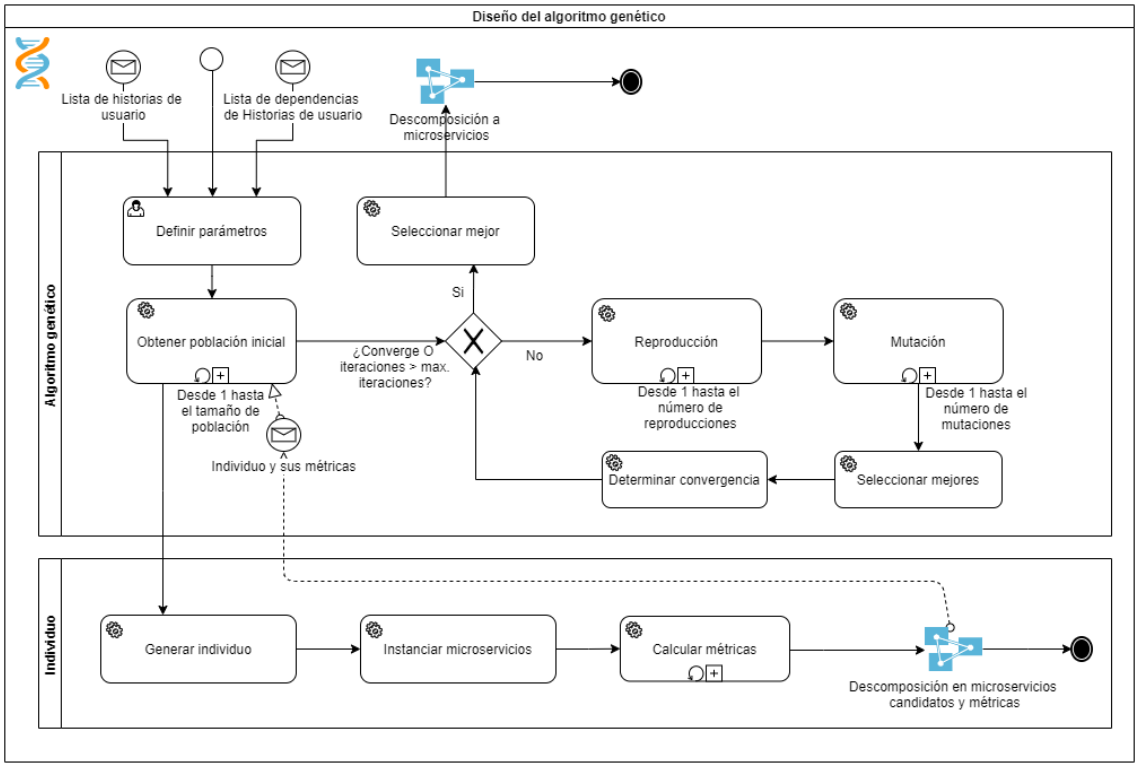


Figura 42. Diseño del algoritmo genético

Los individuos se definen a partir de la asignación de historias a los microservicios, por lo tanto, el cromosoma de cada individuo se define a partir de una matriz de asignación de unos y ceros, donde las columnas corresponden a las historias de usuarios y las filas a los microservicios, y el cruce fila/columna contiene un 1 cuando la historia de usuario se asigna al microservicio o un cero si no. En la tabla 14, se presenta un ejemplo para dos microservicios *MS = {ms₁, ms₂}* y cinco historias de usuario *HU = {hu₁, hu₂, hu₃, hu₄, hu₅}*.

Tabla 14. Ejemplo de la matriz de asignación

Microservicios	hu_1	hu_2	hu_3	hu_4	hu_5
ms_1	1	0	0	1	1
ms_2	0	1	1	0	0

El cromosoma resultante sería la unión de las asignaciones de cada historia de usuario a cada microservicio, para este caso, sería:

Cromosoma: 10011 01100.

A partir de este cromosoma, fue posible definir la función de adaptación o función objetivo, la cual se basa en la ecuación (25), utiliza una combinación de las métricas de acoplamiento (*CpT*, ecuación 11), cohesión (*CohT*, ecuación 14), granularidad (*WsicT*, ecuación 18), complejidad (*CxT*, ecuación 22) y similitud semántica (*SsT*, ecuación 17), sección 4.1. Las funciones objetivo usadas se detallan a continuación:

1. $F1 = \sqrt[2]{(10\,Cpt)^2 + CxT^2 + WsicT^2 + (100 - SsT)^2}$
2. $F2 = \sqrt[2]{(10\,Cpt)^2 + WsicT^2 + (100 - SsT)^2}$
3. $F3 = \sqrt[2]{CxT^2 + (100 - SsT)^2}$

4. $F4 = \sqrt[2]{(10\ Cpt)^2 + CohT^2 + (100 - SsT)^2}$
5. $F5 = \sqrt[2]{(10\ Cpt)^2 + (100 - SsT)^2}$
6. $F6 = \sqrt[2]{(10\ Cpt)^2 + CohT^2 + WsicT^2 + (100 - SsT)^2}$
7. $F7 = \sqrt[2]{(10\ Cpt)^2 + CxT^2 + (100 - SsT)^2}$
8. $F8 = \sqrt[2]{(10\ Cpt)^2 + CohT^2 + WsicT^2}$

Método de reproducción.

Se genera una asignación diferente a partir de los padres seleccionados. En este método, el padre y la madre se seleccionan al azar de la población; para generar el hijo se toma la información del padre y la madre, de la matriz de asignación se toman las primeras columnas del padre y se unen las últimas columnas de la madre, generando una nueva asignación. Hay que tener en cuenta que una historia de usuario no puede ser asignada dos veces, esto significa que en la matriz de asignación sólo puede aparecer un uno en cada columna. Ejemplo: Dados los dos cromosomas:

1) Father: 10011 01100.
2) Mother 01000 10111.
El hijo sería 10000 01111.

Método de mutación.

La mutación indica el cambio de un bit aleatorio del cromosoma, el cambio de un bit del cromosoma de este problema de 1 a 0 o de 0 a 1, implica que una historia de usuario está asignada o no asignada a un microservicio y ésta debe ser asignado y desasignada a otro microservicio. Esto implica que la mutación se hace en dos bits. Ejemplo:

Mutación del bit 7 del cromosoma obtenido: 01011 10100.
Cromosoma mutado: 00011 11100.
En este caso se debe cambiar el bit 7 que es un cero a uno, es decir la historia de usuario en la columna 2 de la matriz debe ser asignada al segundo microservicio y al mismo tiempo ser desasignada del primer microservicio.

Los cromosomas mutados deben ser incluidos en la población. Este proceso se lleva a cabo de forma aleatoria, los individuos que van a mutar se seleccionan de la población, la mutación de un bit también se lleva a cabo de forma aleatoria, para la mutación se calcula el valor de la función objetivo y se incluye en la población.

Método para seleccionar el mejor individuo.

En los procesos de selección genética, los más fuertes sobreviven, en el caso del problema de la generación automática de la asignación de historias de usuario a los microservicios, sobreviven los *n* individuos que mejor se adaptan a las condiciones del problema, corresponden a las asignaciones que implican una menor función objetivo.

La selección se hizo a partir de la función objetivo, se aplicó a cada individuo y se ordenó la población en forma ascendente, considerando en los primeros lugares a los mejores individuos, correspondientes a las asignaciones que implican una menor función objetivo utilizando la ecuación (25).

Convergencia.

Para determinar la convergencia del método, se definió el número de iteraciones o generaciones de la población a procesar, definimos la convergencia cuando el 10% de la población converge al mismo valor de la función objetivo. Si no convergen, al final de las iteraciones se detiene el algoritmo y se selecciona el cromosoma situado en primer lugar, que sería la mejor asignación de las historias de usuarios a los microservicios. Para los estudios de casos utilizados para evaluar el método propuesto, se generó una población de 1000 individuos, con un máximo de 400 iteraciones o generaciones, con 500 hijos y 500 mutaciones en cada generación. El algoritmo se probó varias veces obteniendo el mismo resultado, incluso con más individuos y más iteraciones.

4.4.3. Algoritmo de agrupamiento

El problema consiste en asignar *m* historias de usuarios a *n* microservicios, agrupando las historias con mayor similitud semántica, es decir, agrupando las historias que se refieren al mismo tema o tópico. También se desea que las historias agrupadas en microservicios, estos tengan un bajo acoplamiento y una alta cohesión. No se establece a priori un número predefinido de microservicios o grupos. No es conveniente determinar de antemano cuántos microservicios debe tener la aplicación. Para resolver este problema y considerando las afirmaciones anteriores, se propone el diseño de nuestro algoritmo de agrupación, el cual se puede apreciar en la figura 43.

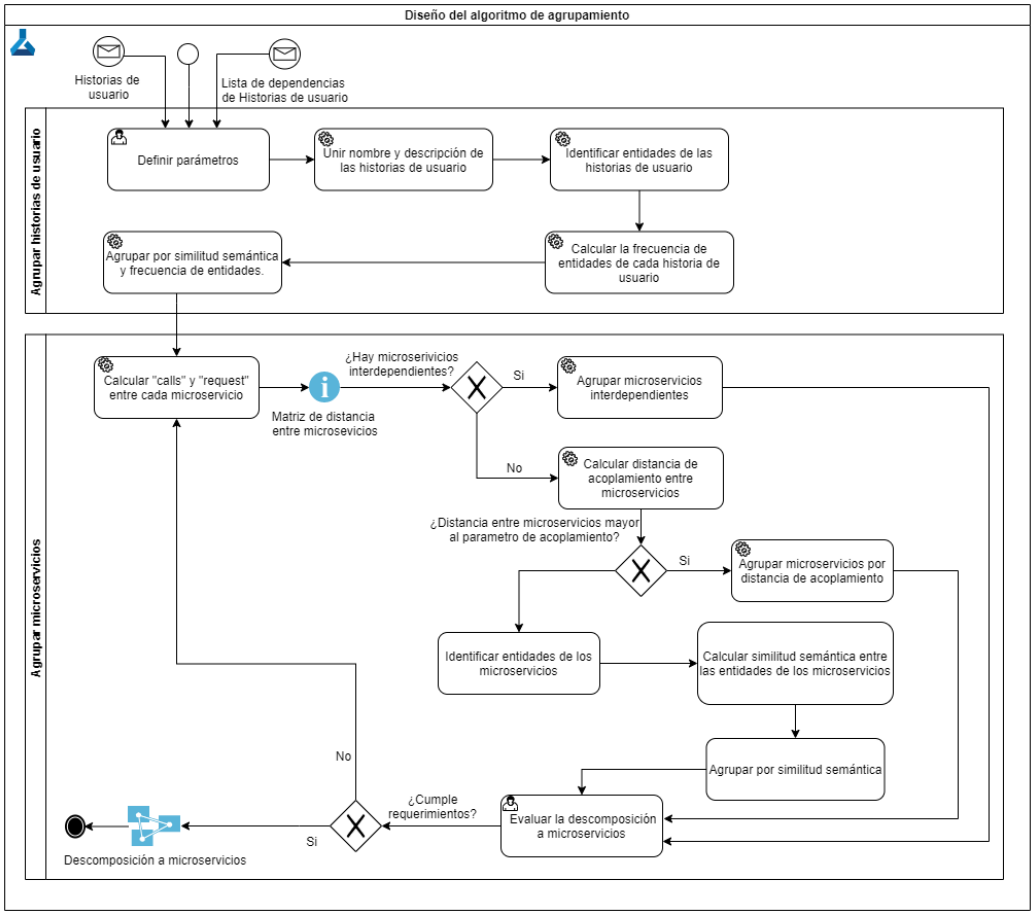


Figura 43. Diseño del algoritmo de agrupamiento

Definir parámetros.

El usuario define los parámetros del algoritmo o datos de entrada, estos son:

- **Parámetro de similitud de agrupamiento:** El cálculo de la similitud semántica devuelve un valor entre 0 y 1, siendo 1 cuando los dos textos son iguales o muy similares. Este parámetro define el valor mínimo de similitud entre los textos de las historias de usuarios que se van a agrupar. Su valor está predefinido en 0,85, por lo que se agruparán las historias que tengan una similitud superior a 0,85. Se hicieron pruebas y se determinó que este valor presenta agrupaciones semánticas coherentes y suficientes.
- **Parámetro de distancia de agrupamiento:** De la misma manera que ocurre con el parámetro anterior, el parámetro de distancia de agrupamiento indica el valor mínimo de la distancia de acoplamiento entre los microservicios a agrupar.
- **Lenguaje:** Este parámetro define el idioma en el que están escritas las historias de los usuarios (español o inglés).
- **Similitudes semánticas sobre Lematización de entidades o Texto completo de entidades:** Este parámetro indica si el texto completo o los lemas de las entidades que forman parte del nombre y la descripción de las historias de usuario se tienen en cuenta al calcular la similitud semántica.

Unir nombre y descripción de las historias

Las historias de usuario se escriben en prosa y establece el requerimiento funcional que la aplicación debe implementar, su descripción detalla los usuarios o perfiles que van a utilizar la historia de usuario y las acciones que debe realizar el sistema sobre la lógica del negocio de la aplicación.

Para el análisis semántico que se realiza para agrupar las historias de usuario en microservicios se unió el nombre de la historia y su descripción, los demás datos de la historia de usuario no fueron tenidos en cuenta para el análisis semántico.

Identificar las entidades de la historia de usuario.

Luego de unir el nombre y la descripción, se eliminan los verbos, artículos, adjetivos y preposiciones del texto obtenido, se dejan sólo los sustantivos porque los sustantivos corresponden a las entidades que están implicadas en la historia de usuario. Las entidades son los objetos sobre los cuales recae la acción que debe hacer la aplicación.

Calcular la frecuencia que se repite cada entidad en la historia de usuario

Se identificaron los tópicos del texto obtenido de la historia de usuario contando el número de veces que se repite una entidad. Se seleccionaron las dos palabras que más se repiten. Estas palabras se guardan en una cadena para hacer los cálculos de la similitud semántica.

Agrupar por similitud semántica.

Para agrupar las historias de usuario en microservicios se implementó un algoritmo de agrupamiento similar al algoritmo de agrupamiento jerárquico con algunas variaciones, a partir del listado de historias de historia de usuario, se comparan las similitudes entre cada historia de usuario y si la similitud supera al parámetro de agrupamiento las historias se incluyen en el mismo

microservicio, en caso de que las historias no sean similares se agrupan en un microservicio diferente. Es un proceso iterativo donde cada historia se compara con cada microservicio candidato y se agrupa donde la similitud semántica sea más alta, si la historia no es similar a ninguno de los microservicios candidatos se agrega un nuevo microservicio incluyendo esta historia.

Una vez obtenida la distribución de las historias de usuario en microservicio y a partir del diccionario de similitud semántica se calcula la similitud semántica de cada microservicio usando la ecuación (16) y la similitud semántica de la aplicación basada en microservicios usando la ecuación (17).

Agrupar por interdependencia entre microservicios

Para agrupar los microservicios interdependientes, el algoritmo recorre uno a uno los microservicios, se calcula en número de veces que el i-ésimo microservicio llama al j-ésimo microservicio (*calls$_i$*), y también se calcula el número de veces que el j-esimo microservicio llama al i-ésimo microservicio (*reques$_i$*), si ambos valores son mayores a cero es porque el i-ésimo microservicio es interdependiente con el j-ésimo microservicio y por lo tanto deben ser unidos. Este proceso se realiza de forma iterativa para cada microservicio, uniendo aquellos que resulten interdependientes.

Calcular "Calls" y "Request" entre cada microservicio.

El sistema calcula el número de llamadas y peticiones entre cada par de microservicios, se obtiene una matriz donde cada fila incluye los microservicios comparados *(ms$_i$, ms$_j$)*, la cantidad de veces que *ms$_i$* llama *(Calls)* a *ms$_j$* y la cantidad de peticiones *(requests)* que *ms$_j$* hace a ms$_i$.

Agrupar microservicios interdependientes.

Se realiza un proceso iterativo donde se identifican los microservicios interdependientes para ser agrupados, se recorre la matriz de "calls" y "request", si la cantidad de llamadas de *MS$_i$* a *MS$_j$* es mayor a cero y la cantidad de peticiones de *MS$_j$* a *MS$_i$* es también mayor a cero, entonces los microservicios son interdependientes y deben ser agrupados. Se busca que la solución o descomposición no tenga microservicios interdependientes, por lo tanto, se reduce el acoplamiento de la aplicación.

Calcular distancia de acoplamiento entre los microservicios.

El sistema calcula el acoplamiento de cada microservicio y la aplicación. Este proceso obtiene una matriz que relaciona la distancia de acoplamiento *(CpD)* entre cada microservicio, la distancia de acoplamiento se basa en las llamadas y solicitudes entre dos microservicios, *CpD* se definió de la siguiente manera.

$$CpD_{i-j} = (calls_{i-j} + Reques_{i-j}) / total\_calls \qquad (26)$$

Donde:
CpD$_{i-j}$: Distancia de acoplamiento entre el i-ésimo microservicio y el j-esimo microservicio.
Calls$_{i-j}$: número de veces que el i-ésimo microservicio llama al j-ésimo microservicio.
Request$_{i-j}$: número de veces que el j-esimo microservicio llama al i-ésimo microservicio.
Total_calls: corresponde al número total de llamadas entre los microservicios de *MSBA.*

Agrupar por distancia de acoplamiento.

Se agrupan los microservicios cuya distancia sea mayor al parámetro de agrupamiento de acoplamiento (Este parámetro puede ser variado por el usuario, inicialmente y según las pruebas realizadas su valor es de 0.5). Se busca reducir la alta comunicación entre los microservicios de la aplicación. Si dos microservicios tienen muchas dependencias deben ser unidos, reduciendo así el acoplamiento.

Identificar entidades del microservicio.

Luego de reducir los microservicios interdependientes y reducir los microservicios que concentran mayor acoplamiento, se verifica la similitud semántica de los microservicios de la aplicación. Si se encuentran microservicios que son muy similares semánticamente deben ser agrupados, aumentado así la cohesión de los microservicios.

Para identificar las entidades del microservicio se unió en una cadena de texto el nombre y la descripción de todas las historias que conforman el microservicio, luego de esa cadena se eliminaron los verbos, artículos, adjetivos y preposiciones, dejando sólo los sustantivos o entidades.

Calcular la frecuencia de las entidades del microservicio.

Con la cadena de texto que contiene solo las entidades que representan al microservicio, se cuenta la frecuencia que aparece cada entidad y se seleccionan las dos mayores para hacer las comparaciones semánticas de entre los microservicios.

Calcular la similitud semántica entre las entidades que más se repiten de los microservicios.

Con las entidades que representan cada microservicio se calcula de forma iterativa las similitudes entre cada microservicio, obteniendo una matriz de similitud semántica.

Agrupar por similitud semántica.

Con el mismo algoritmo utilizado para agrupar las historias de usuario se agrupan los microservicios, comparando ahora la similitud semántica entre los textos de las entidades que representan a cada microservicio. Se compara de forma iterativa en la matriz de similitud semántica, la similitud entre cada par de microservicios, si esta similitud es mayor al parámetro de agrupamiento por similitud semántica entonces los microservicios deben ser unidos. Este parámetro puede ser variado por el usuario, inicialmente y según las pruebas realizadas su valor es de 0.85, por lo tanto, se agrupan los microservicios cuya similitud semántica sea mayor a 0.85.

Evaluar la descomposición a microservicios.

El usuario evalúa lo solución obtenida, sino cumple con los requerimientos deseados, si tiene aún alto acoplamiento, baja cohesión y alta complejidad, puede repetir el proceso de agrupamiento de microservicios las veces que desee hasta encontrar una solución adecuada.

La evaluación del algoritmo de agrupamiento propuesto se presenta en el capítulo 4, aplicado en los casos de estudio seleccionados para validar el Microservices Backlog.

4.5. COMPONENTE CALCULADOR DE MÉTRICAS

El sistema a través del componente calculador de métricas obtiene las métricas de acoplamiento, cohesión, complejidad, granularidad, solicitudes - llamadas, y tiempo estimado de desarrollo. Con estas métricas, se puede evaluar y comparar las descomposiciones de cada proyecto para tomar decisiones en tiempo de diseño.

Utilizamos métricas para medir el acoplamiento, la cohesión, la granularidad (tamaño), el rendimiento (solicitudes y llamadas), la similitud semántica, la complejidad y el tiempo estimado de desarrollo, en aplicaciones basadas en microservicios, obtenidas a partir de las historias de usuarios en tiempo de diseño. Estas métricas se adaptaron a partir de propuestas del estado del arte [101], [102], y [103]. La figura 44 muestra el diseño del componente calculador de métricas.

Estas métricas se calculan automáticamente para cada descomposición obtenida por el "Microservices Backlog" o introducida por el usuario. Se presentan tablas comparativas de estas métricas para que el arquitecto o los desarrolladores puedan tomar decisiones.

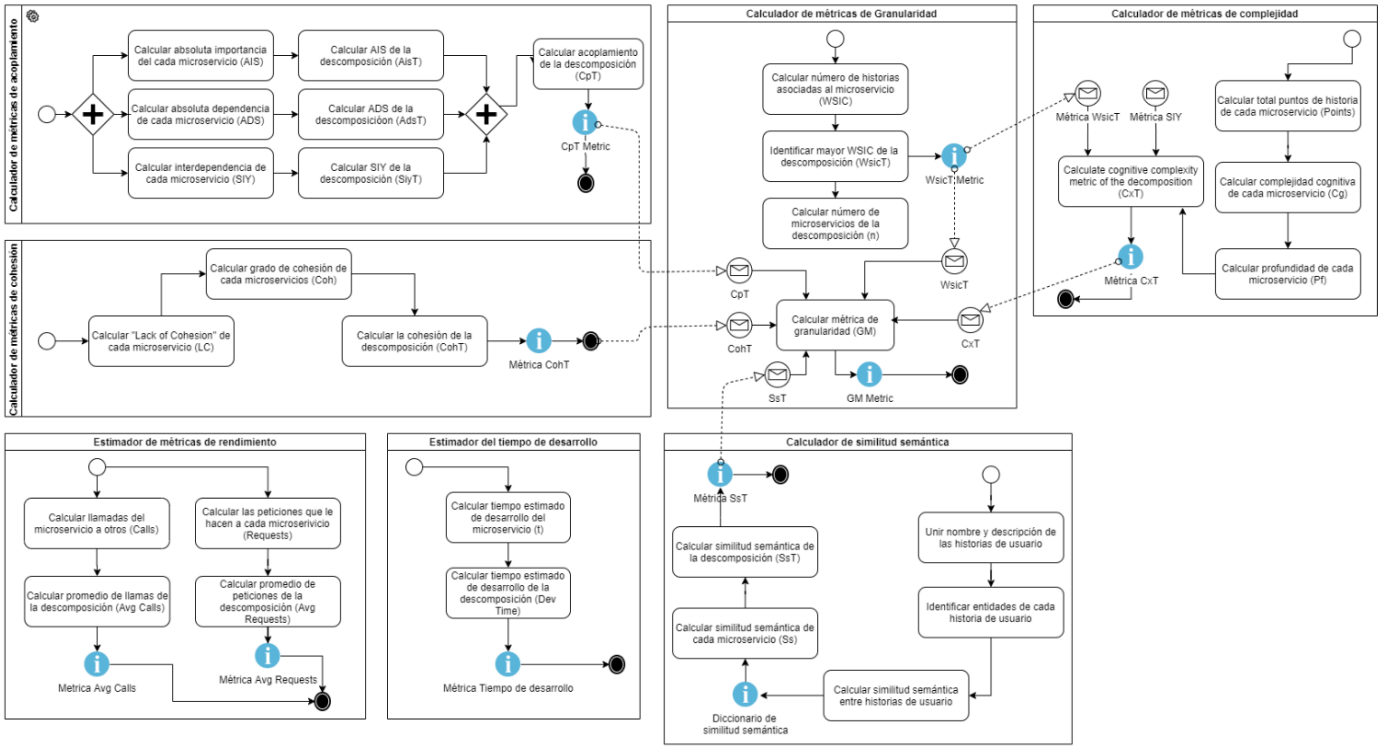


Figura 44. Diseño del componente calculador de métricas

La salida del algoritmo calculador de métricas se puede apreciar en la figura 45. En la parte superior de la figura aparecen las métricas a nivel de sistema y en la parte inferior dentro del cuadro azul aparecen las métricas de cada microservicio.

El usuario o arquitecto puede analizar los datos y determinar si la solución obtenida se adapta a sus requerimientos o necesidades. Después de calcular las métricas el usuario puede observar el diagrama del Microservices Backlog o compararla con otras soluciones obtenidas o ingresadas al sistema.

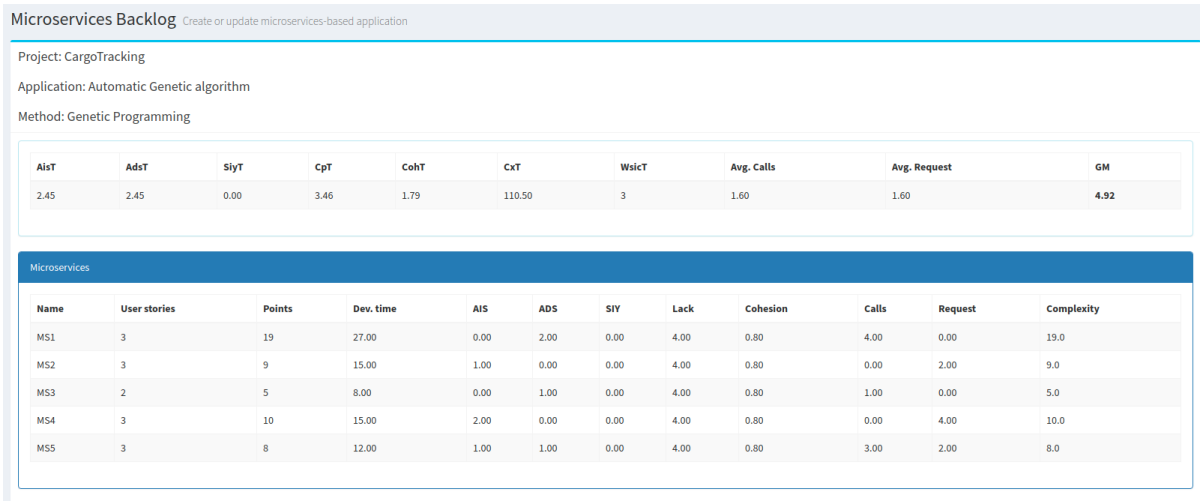


Figura 45. Salida del componente calculador de métricas

4.6. DIAGRAMA MICROSERVICE BACKLOG

Los resultados del modelo son las métricas calculadas y el diagrama del Microservice Backlog. La figura 46 muestra el Microservice Backlog para la aplicación “Cargo Tracking”.

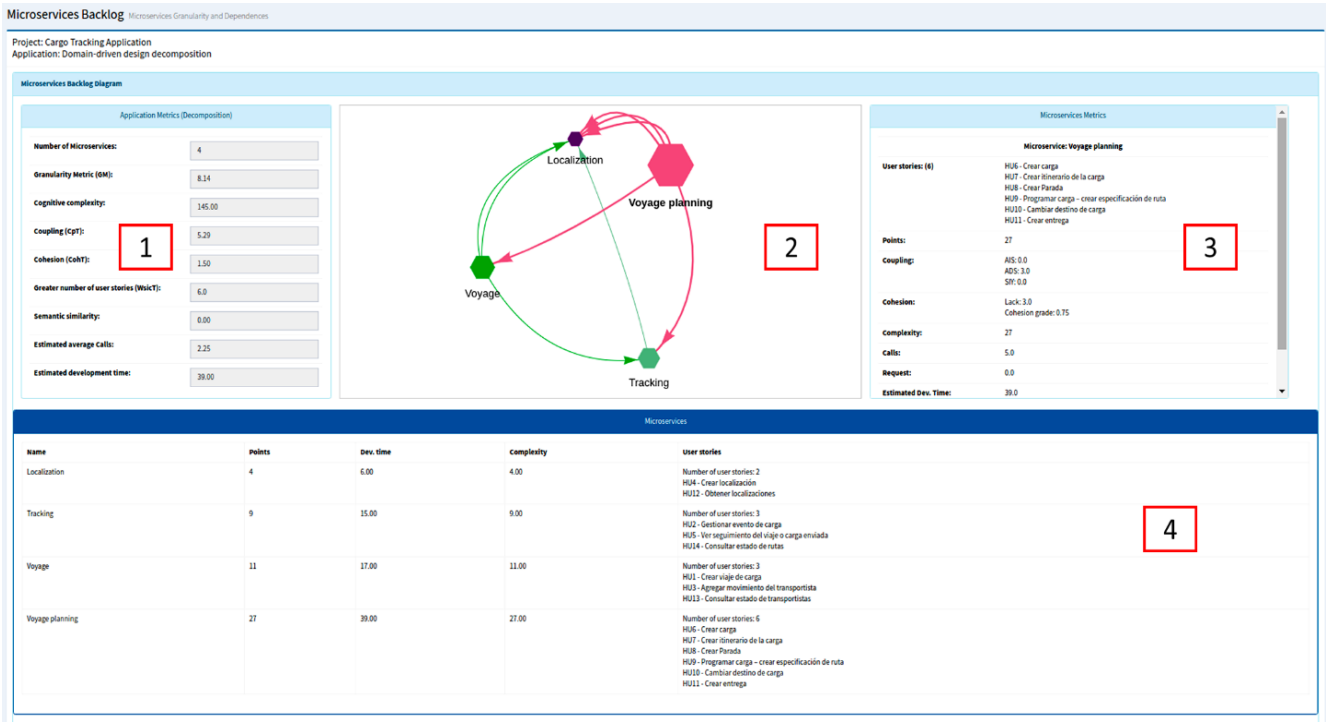


Figura 46. Microservices backlog para la aplicación Cargo Tracking, los microservicios fueron identificados mediante un diseño basado en el dominio (DDD).

En 1) Se muestran las métricas de MSBA; 2) Gráfico de dependencias de MSBA; 3) Métrica de cada microservicio; 4) Detalles de los microservicios.

A partir del modelo, el diseñador puede ver el tamaño de cada microservicio, así como su complejidad, dependencias, acoplamiento, cohesión y tiempo de desarrollo. El arquitecto puede

notar a primera vista que el microservicio púrpura (Localización) es un punto crítico del sistema si este microservicio falla, entonces todo el sistema puede fallar porque es usado por todos los demás. El arquitecto en tiempo de diseño ya puede pensar en mecanismos de tolerancia a fallas en ese microservicio, en el balanceo de carga y en la supervisión de ese microservicio crítico; también puede tener una visión del sistema global en tiempo de diseño. El diagrama propuesto de los microservicios en la figura 42 se obtuvo usando el diseño impulsado por el dominio (DDD) realizando los siguientes pasos:

1. Se identificaron las historias de los usuarios de Cargo Tracking.
2. Se definieron las dependencias entre las historias.
3. Se identificaron las entidades.
4. Se definieron los agregados.
5. Se establecieron los contextos delimitados; se asociaron las entidades y sus respectivas historias de usuario.
6. Se calcularon las métricas; métricas específicas para cada microservicio y para toda la aplicación.

Se puede resaltar que el componente agrupador de nuestro modelo identifica automáticamente los microservicios candidatos cuando se utiliza el algoritmo genético o el algoritmo de agrupamiento, entonces los pasos 3 a 5 son automáticos. Las métricas fueron calculadas por el componente calculador de métricas. Luego de obtener e ingresar las descomposiciones, se puede realizar operaciones de unión o de división de los microservicios, realizar un análisis comparativo de las descomposiciones y seleccionar la mejor. La figura 47 presenta la tabla comparativa de un proyecto registrado en el sistema. Se presentan las métricas para cada descomposición, las cuales pueden ser ordenadas automáticamente para analizar y comparar.

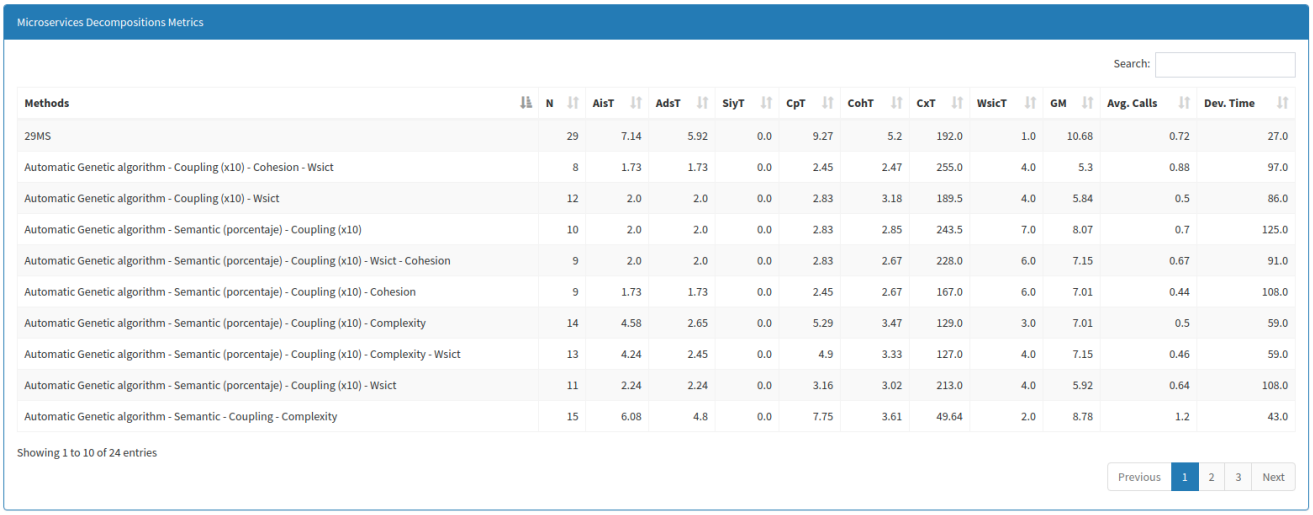


Figura 47. Evaluar descomposiciones en el Microservices Backlog.

5. EVALUACIÓN

5.1. MODELO DE EVALUACIÓN

Se utilizó la investigación de la ciencia del diseño (Design Science Research), siguiendo el paradigma propuesto por Hevner y otros [45], el cual busca extender las fronteras de las capacidades humanas y organizacionales creando nuevos e innovadores artefactos (véase la figura 48). Se propuso una nueva práctica ágil llamada Microservices Backlog.

El proceso de investigación se inició con el diseño y desarrollo del Microservices Backlog, el cual fue evaluado iterativamente en un estudio de campo mediante un análisis estático y dinámico; con cada evaluación se mejoró y corrigió hasta obtener una propuesta óptima. La construcción del Microservicies Backlog se basa en los siguientes fundamentos teóricos: ingeniería de software, inteligencia artificial, computación en la nube, computación de servicios y desarrollo ágil de software.

Los interesados corresponden a arquitectos de software, equipos de desarrollo de software y jefes de proyecto, que requieran desarrollar o migrar una aplicación basada en microservicios; el Microservices Backlog les permite definir la granularidad del microservicio y evaluar la arquitectura de la aplicación.

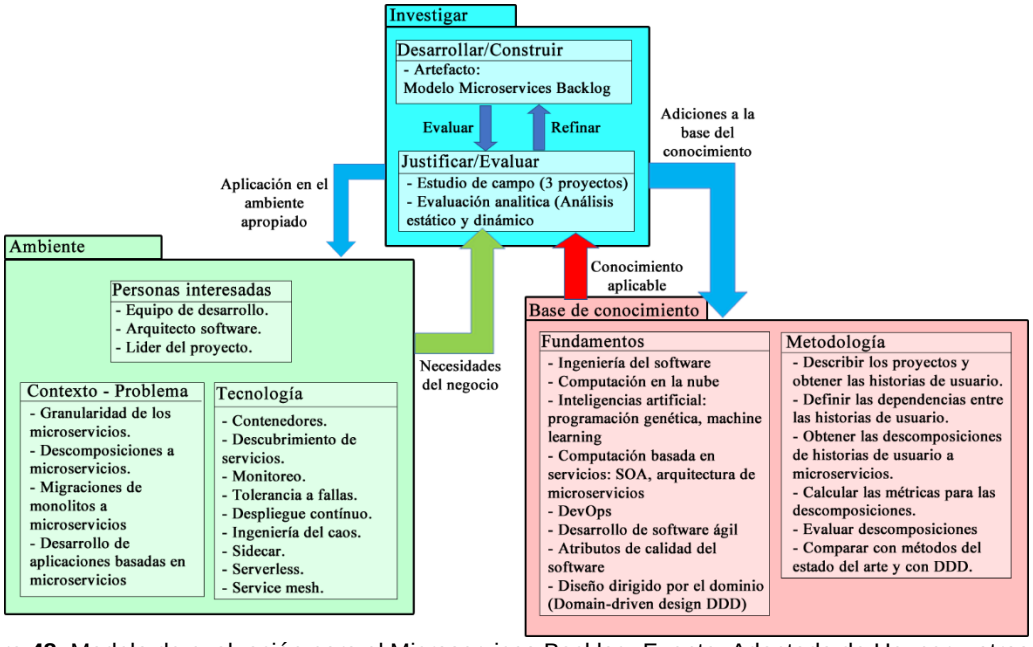


Figura 48. Modelo de evaluación para el Microservices Backlog. Fuente: Adaptado de Hevner y otros [45].

El proceso de investigación se detalla a continuación:

1. **Definición del contexto del problema:** Se definió el contexto del problema: granularidad de los microservicios, descomposición de microservicios, migraciones de monolito a microservicios y desarrollo de aplicaciones basadas en microservicios.
2. **Fundamentación teórica y definición del estado del arte:** se realizó una revisión sistemática de la literatura, se identificaron, adaptaron y propusieron métricas para definir la granularidad de los microservicios; y se identificaron los trabajos relacionados.

3. **Diseño del Microservices Backlog:** se diseñó el Microservices Backlog y se propuso una especificación formal del modelo de granularidad.
4. **Desarrollo del Microservices Backlog:** se construyó un modelo inteligente de granularidad, implementado como un algoritmo genético y un algoritmo de agrupamiento no supervisado, para descomponer el "product backlog" en microservicios. Se implementó un algoritmo para evaluar las métricas de las descomposiciones de microservicios o aplicaciones basadas en microservicios.
5. **Evaluación del Microservices Backlog:** se evaluó el modelo utilizando ejemplos obtenidos de la revisión del estado del arte (Cargo Tracking y JPet Store) y dos proyectos de la vida real (Foristom Conferences y Sinplafut). La evaluación comparó el rendimiento de la descomposición obtenida por el Microservices Backlog con las descomposiciones realizadas por otros métodos: Diseño basado en el dominio (DDD) [123], Service Cuttuer [124], Identificación de Microservicios a través de Análisis de Interfaz (MITIA) [83], e Identificación de Candidatos a Servicios de Sistemas Monolíticos basados en Trazas de Ejecución ("Execution Traces") [61]. Se tomaron las descomposiciones propuestas por MITIA y las trazas de ejecución sobre los ejemplos del estado del arte, luego se identificaron las operaciones asociadas a cada microservicio, y luego se asociaron las operaciones a las historias de usuarios. Tradicionalmente, las historias de usuario especifican los requisitos funcionales de la aplicación y son implementadas como operaciones. Existe dificultad para encontrar proyectos open source que tengan disponibles las historias de usuario para poder hacer las pruebas y comparaciones entre los métodos del estado del arte.
   Dado que la DDD es el método más utilizado para la identificación de microservicios, para la evaluación de los proyectos reales se verificó que la descomposición obtenida era consistente y cercana a la DDD.
6. **Proponer el Microservices Backlog:** se propone el modelo de granularidad Microservice Backlog. De acuerdo con las métricas y a la evaluación analítica, incluidos los ajustes correspondientes en cada evaluación, a través del proceso de investigación, al finalizar la investigación se propone el Microservice Backlog como un modelo inteligente de evaluación y especificación de la granularidad.

5.2. MÉTODOS DE EVALUACIÓN

Siguiendo la recomendación de Hevner y otros [45], se utilizaron métodos de evaluación observacional y analíticos para evaluar la técnica. El método de observación fue un estudio de campo, en el que utilizamos y supervisamos el modelo Microservices Backlog en 4 proyectos (dos hipotéticos y dos reales).

Los métodos analíticos fueron tanto estáticos como dinámicos. Se calcularon métricas de complejidad, acoplamiento, cohesión, dependencias, rendimiento y tamaño de la descomposición propuesta (o de aplicación basada en microservicios), y luego se compararon con otros enfoques del estado del arte. Las métricas se calcularon a partir de los datos de las historias de los usuarios y las dependencias entre ellas en tiempo de diseño, las mismas historias de usuario, las mismas dependencias y el mismo algoritmo que las calcula se utilizaron en todas las pruebas. El proceso de evaluación se llevó a cabo de la siguiente manera:

1. Se analizaron y describieron los ejemplos del estado del arte y los proyectos.
2. Se especificaron las historias de usuario de cada ejemplo y proyectos, obteniendo el "Product Backlog".
3. Se definieron las dependencias de las historias de usuarios. Las cuales fueron identificadas de acuerdo con la lógica de negocio, el flujo de datos, las invocaciones o llamadas entre historias de usuario.

4. Se obtiene la descomposición a través del Microservices Backlog (algoritmo genético y algoritmo de agrupamiento, usando una combinación de las variables de *Gm* ecuación (21)) y se ingresan las descomposiciones de los métodos del estado del arte.
5. Para cada descomposición, el algoritmo calculador de métricas obtiene las métricas y el diagrama de dependencias (grafo) o el Microservices Backlog de los microservicios candidatos.
6. Se evaluaron las descomposiciones y se compararon las métricas.

Comprobamos que la práctica propuesta permite definir la granularidad apropiada de los microservicios y evaluar las métricas de las aplicaciones basadas en los microservicios.

Hipótesis.

Se planteó la siguiente hipótesis de investigación H_0: La solución propuesta por el Microservices backlog no presenta bajo acoplamiento, alta cohesión, baja complejidad, menor comunicación, menos dependencias ni alta similitud semántica, comparado con la solución propuesta por métodos del estado del arte y por DDD.

5.3. MÉTRICAS DE EVALUACIÓN

Utilizamos métricas para medir el acoplamiento, la cohesión, la granularidad (tamaño), el rendimiento (solicitud y llamadas), la similitud semántica, la complejidad y la estimación del tiempo de desarrollo, en aplicaciones basadas en microservicios. Estas métricas se adaptaron a partir de los enfoques del estado del arte [101], [102], y [103]. La formulación matemática de las métricas utilizadas se presenta en la sección 4.1.

El algoritmo genético en su función objetivo utiliza una combinación de 5 métricas (Acoplamiento *CpT*, cohesión *CohT*, granularidad *WsicT*, similitud semántica *SsT* y complejidad cognitiva *CxT*), para obtener los microservicios candidatos se probaron diferentes combinaciones de esas métricas, las funciones objetivo utilizadas se detallaron en la sección 4.3.2, las cuales son diferentes a la métrica de granularidad *Gm* usada en la evaluación.

El algoritmo de agrupamiento utiliza para agrupar las historias de usuario en microservicios la similitud semántica (*SS* y *SsT*) usando las ecuaciones (18) y (19) respectivamente, junto con la distancia de acoplamiento usando la ecuación (26).

La evaluación utiliza otras métricas diferentes, fueron planteadas en la formulación matemática propuesta en la sección 4.1. Estas métricas son:

1. **Granularidad:** se usa la métrica de granularidad *(Gm)* propuesta en la ecuación (3) y el número de microservicios de la aplicación (N).
2. **Complejidad:** los puntos de historia del microservicio *(P_i).*
3. **Rendimiento:** llamadas entre los microservicios *(Calls)*, y el promedio de llamadas entre microservicios de la aplicación *(Avg. Calls)*.
4. **Tiempo estimado de desarrollo:** el mayor tiempo estimado de desarrollo (T) de la aplicación basada en microservicios.

Las mismas métricas, el mismo proceso para calcularlas y el mismo algoritmo fueron usados para comparar las descomposiciones obtenidas por los métodos evaluados, de la misma forma se

usan para todos los casos las mismas historias de usuario y la misma definición de dependencias entre las historias de usuario.

5.4. CASOS DE ESTUDIO

Se evaluó y demostró el modelo del Microservices Backlog usando cuatro aplicaciones, dos ejemplos del estado del arte: Cargo Tracking y JPet-Store; y dos proyectos de la vida real: Foristom Conferencias y Sinplafut. Comparamos “Cargo Tracking” y “Jpet-Store” con las descomposiciones obtenidas con DDD, enfoques del estado del arte y nuestro modelo. Mientras que los proyectos de la vida real se compararon con DDD, la solución del arquitecto de software o del equipo de desarrollo y las descomposiciones obtenidas por el Microservices Backlog.

Se ejecutó el algoritmo genético para todas las pruebas con una población de 1000 individuos, convergencia 10%, número máximo de iteraciones 400 en algunos casos se amplió hasta 1000 iteraciones para buscar la convergencia; con 500 hijos y 500 mutaciones en cada iteración. También, obtuvimos las descomposiciones a los ejemplos y proyectos usando el algoritmo de agrupamiento, los resultados fueron comparados y analizados.

A continuación, se detalla cada caso de estudio y se realiza el análisis comparativo. Para cada caso de estudio se llevó a cabo el siguiente proceso:

1. Se describe el estudio de caso, a partir de la información publicada.
2. Se identifican las historias de los usuarios y se define el “product backlog”.
3. Se identifican las dependencias entre las historias de usuarios.
4. Las descomposiciones se obtienen para cada método de comparación.
5. Las métricas se calculan mediante el componente calculador de métricas.
6. Las soluciones propuestas por cada método se presentan gráficamente.
7. La tabla de datos de métricas se presenta para análisis comparativo.
8. Se presentan gráficos comparativos de las métricas de cada método.
9. Se compara el valor obtenido en la similitud semántica (SsT), la complejidad cognitiva (*CxT*) y en la métrica de granularidad *Gm*.
10. Los mejores resultados del algoritmo genético y del algoritmo de agrupamiento se comparan gráficamente con DDD.

5.4.1. Cargo Tracking Application

Baresi y otros [83] describen la aplicación “Cargo Tracking” como sigue: "el objetivo de la aplicación es mover una carga (identificada por un TrackingId) entre dos ubicaciones a través de una especificación de ruta. Una vez que una Carga está disponible, se asocia con uno de los Itinerarios (listas de Movimientos de Transportistas), seleccionados de los Viajes existentes. HandlingEvents entonces rastrea el progreso de la Carga en el Itinerario. La entrega de una carga informa sobre su estado, la hora estimada de llegada y si está en camino". Del modelo de dominio propuesto por Baresi y otros, se extrajeron y plantearon las historias de usuarios y el “product backlog” se detalla en la tabla 15. Los puntos y tiempos son datos de entrada a nuestro modelo.

Un punto crítico del método propuesto en este trabajo doctoral son las dependencias entre las historias de usuarios. Deben ser identificadas y proporcionadas como datos de entrada al método. El componente parametrizador ofrece una funcionalidad para definir las dependencias entre las historias de usuarios. Se define una dependencia entre HU_i y HU_j cuando HU_i llama o ejecuta HU_j. Por ejemplo, para crear un viaje (HU_1) se debe obtener las ubicaciones (HU_{12}), esto implica

que la HU_1 tiene una dependencia con la HU_{12}. La tabla 16 presenta las dependencias que se identificaron entre las historias de usuarios. Las dependencias se calcularon de acuerdo con la lógica de la aplicación propuesta en los trabajos relacionados.

Tabla 15. Product backlog para la aplicación Cargo Tracking

ID	Nombre	Puntos	Tiempo Des. (hours)
HU_1	Crear viaje de carga	3	5
HU_2	Gestionar evento de carga	3	5
HU_3	Agregar movimiento del transportista	5	7
HU_4	Crear localización	2	3
HU_5	Ver seguimiento del viaje	3	5
HU_6	Crear carga	7	10
HU_7	Crear itinerario de la carga	5	7
HU_8	Crear Parada	2	3
HU_9	Programar carga – crear especificación de ruta	5	7
HU_{10}	Cambiar destino de carga	1	2
HU_{11}	Crear entrega	7	10
HU_{12}	Obtener localizaciones	2	3
HU_{13}	Consultar estado de transportistas	3	5
HU_{14}	Consultar estado de rutas	3	5
	Total:	51	77

ID: Identificador de la historia de usuario. Puntos: Puntos de historia estimados. Tiempo De. Tiempo de Desarrollo estimado.

Tabla 16. Dependencias entre las historias de usuarios para la aplicación Cargo Tracking

Historia de usuario	Dependencias	Historias de usuario	Dependencias
HU_1	$\{HU_{12}, HU_3\}$	HU_8	$\{HU_{12}\}$
HU_2	$\{HU_{12}\}$	HU_9	$\{HU_{12}\}$
HU_3	$\{HU_{12}\}$	HU_{10}	$\{HU_{12}\}$
HU_4	{}	HU_{11}	$\{HU_6, HU_{13}, H_{U14}\}$
HU_5	{}	HU_{12}	{}
HU_6	$\{HU_7, HU_9, HU_{11}\}$	HU_{13}	$\{HU_5\}$
HU_7	$\{HU_8\}$	HU_{14}	$\{HU_5\}$

Las dependencias se utilizan para calcular las métricas, por ejemplo, para calcular la métrica *AIS* de la descomposición obtenida con DDD para el microservicio llamado Localización (ver figura 39). MS_1*(Viaje) = {*HU_1*,* HU_3*,* HU_{13}*}*, MS_2 *(Seguimiento) = {*HU_2*,* HU_5*,* HU_{14}*}*, MS_3 *(Localización) = {*HU_4*,* HU_{12}*}*, MS_4 *(Planificación del viaje) = {*HU_6*,* HU_7*,* HU_8*,* HU_9*,* HU_{10}*,* HU_{11}*}*. La métrica *AIS* es el número de microservicios que invocan al menos una operación de la interfaz del microservicio. Luego se contaron el número de microservicios que llaman o usan HU_4 o HU_{12} de las dependencias. La HU_4 no es utilizada por ninguna otra historia de usuario, no aparece en ninguna dependencia (Ver tabla 12), mientras que la HU_{12} es utilizada por las historias HU_1*,* HU_2*,* HU_3*.* HU_8*,* HU_9 y HU_{10} corresponden a 3 microservicios, por lo tanto, *AIS = 3*. De manera similar, se calculan las otras métricas. El cálculo lo realiza el componente calculador de métricas del Microservices Backlog.

La Figura 49 presenta el diagrama de dependencias para las descomposiciones generadas por el Microservices Backlog en comparación con DDD, MITIA, y Service Cutter para la aplicación Cargo Tracking.

Utilizamos diferentes combinaciones de *CpT, CohT, WsicT, CxT y SsT*. Los mejores resultados fueron usando la función objetivo F1 (*CpT, CxT, WsicT y SsT)*, que obtiene tres microservicios. La tabla 17 muestra la descomposición obtenida por cada método y las historias de usuarios que comprenden los microservicios. Todos los métodos evaluados convergieron en casi el mismo número de microservicios (3 o 4 microservicios).

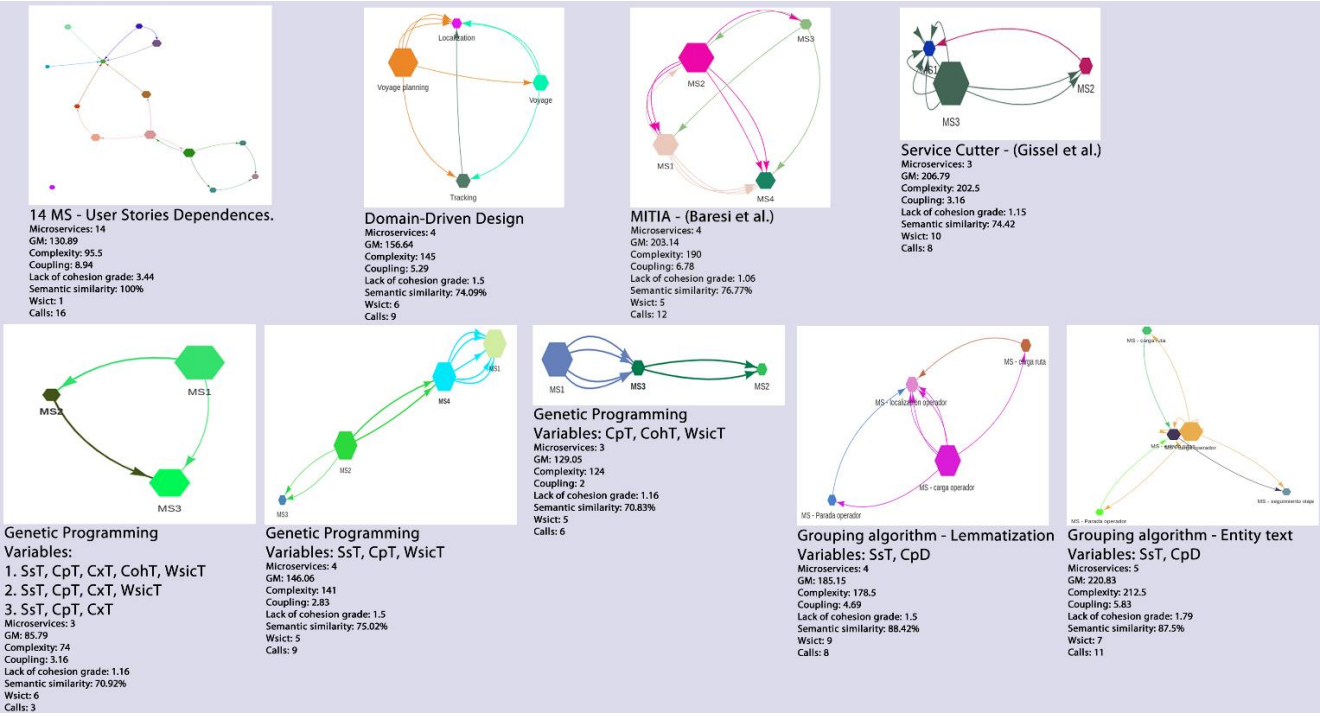


Figura 49. El modelo Microservices Backlog comparado con DDD, MITIA y Service Cutter para la aplicación Cargo Tracking.

La distribución de las historias de usuarios en los microservicios fue diferente. El Microservices Backlog obtuvo descomposiciones coherentes desde el punto de vista semántico.

El Microservices Backlog obtuvo un bajo acoplamiento, menos microservicios, menos llamadas, menos complejidad y una cohesión similar en comparación con los otros métodos del estado del arte (ver tabla 17). La descomposición realizada por nuestro modelo fue diferente de la DDD, nuestro modelo no agrupa las entidades y sus historias que componen un agregado de DDD, ni considera las transacciones entre las historias de los usuarios o la lógica del negocio de la aplicación. Estos temas se considerarán en un trabajo futuro.

Para este caso de estudio no se realizaron operaciones adicionales de unión o desunión de los microservicios obtenidos automáticamente.

En la descomposición obtenida con el algoritmo genético del Microservice Backlog, se elimina el punto crítico de falla de la solución propuesta por DDD, en la cual el microservicio llamado Localización es utilizado por todos los otros microservicios. Se reduce el número de llamadas entre los microservicios, mejorando así el rendimiento. También se reduce el número máximo de operaciones asociadas a un microservicio, así como la complejidad cognitiva y el tiempo estimado de desarrollo. En la descomposición generada por el algoritmo genético del Microservices Backlog, más microservicios pueden funcionar de forma independiente sin usar otros microservicios. Mientras que en la solución propuesta por DDD, sólo un microservicio puede funcionar independientemente. En la descomposición propuesta por DDD, hay más dependencias. Por lo tanto, el modelo propuesto y el Microservices Backlog mejora la descomposición y la identificación de los microservicios. En el algoritmo de agrupamiento también se presenta el punto crítico de falla en al microservicio Localización.

Tabla 17. Comparación de las descomposiciones obtenidas por los métodos evaluados

Método	N	Descomposición a microservicios
Genetic algorithm F1: CpT, CxT, WsicT, SsT F7: CpT, CxT, SsT	3	$MS_1 = \{HU_5, HU_6, HU_{11}, HU_{13}, HU_{14}\}$ $MS_2 = \{HU_4, HU_7, HU_8\}$ $MS_3 = \{HU_1, HU_2, HU_3, HU_9, HU_{10}, HU_{12}\}$
Genetic algorithm F2: CpT, WsicT, SsT	4	$MS_1 = \{HU_2, HU_3, HU_4, HU_8, HU_{12}\}$ $MS_2 = \{HU_6, HU_{11}\}$ $MS_3 = \{HU_5, HU_{13}, HU_{14}\}$ $MS_4 = \{HU_1, HU_7, HU_9, HU_{10}\}$
Genetic algorithm F8: CpT, CohT, WsicT	3	$MS_1 = \{HU_4, HU_6, HU_{11}, HU_{13}, HU_{14}\}$ $MS_2 = \{HU_5, HU_7, HU_8, HU_9\}$ $MS_3 = \{HU_1, HU_2, HU_3, HU_{10}, HU_{12}\}$
Grouping Algorithm – Lemmatization	4	$MS_1 = \{HU_1, HU_2, HU_5, HU_6, HU_7, HU_{10}, HU_{11}, HU_{13}, HU_{14}\}$ $MS_2 = \{HU_9\}$ $MS_3 = \{HU_3, HU_4, HU_{12}\}$ $MS_4 = \{HU_8\}$
Grouping Algorithm – Text	5	$MS_1 = \{HU_1, HU_2, HU_6, HU_7, HU_{10}, HU_{11}, HU_{13}\}$ $MS_2 = \{HU_9\}$ $MS_3 = \{HU_3, HU_4, HU_{12}, HU_{14}\}$ $MS_4 = \{HU_8\}$ $MS_4 = \{HU_5\}$
DDD	4	$MS_1 = \{HU_1, HU_3, HU_{13}\}$ $MS_2 = \{HU_2, HU_5, HU_{14}\}$ $MS_3 = \{HU_4, HU_{12}\}$ $MS_4 = \{HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}\}$
Service Cutter	3	$MS_1 = \{HU_4, HU_{12}\}$ $MS_2 = \{HU_2, HU_5\}$ $MS_3 = \{HU_1, HU_3, HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}, HU_{13}, HU_{14}\}$
MITIA	4	$MS_1 = \{HU_3, HU_9, HU_{10}, HU_{13}\}$ $MS_2 = \{HU_1, HU_2, HU_5, HU_{11}, HU_{14}\}$ $MS_3 = \{HU_6\}$ $MS_4 = \{HU_4, HU_7, HU_8, HU_{12}\}$

Método: Método evaluado. N: Número de microservicios identificados. MS_i: i-ésimo microservicio. HU_k: k-ésima historia de usuario asociada a MS_i.

Tabla 18. Análisis comparativo para la aplicación Cargo Tracking

Método / métricas	Acop.	Cohesión		Granularidad		Rend.		Complejidad		T	Gm
	Cpt	Coht	Sst (%)	N	Wsict	Calls	Avg. Calls	Max. Pi	Cxt		
Genetic algorithm F1: Cpt, cxt, wsict, sst F7: cpt, cxt, sst	3.16	1.16	70.9	3	6	3	1.00	23	74.0	35	85.8
Genetic algorithm F2: cpt, wsict, sst	2.83	1.50	75.0	4	5	9	2.25	14	141.0	21	146.1
Genetic algorithm F8: cpt, coht, wsict	2.00	1.16	70.8	3	5	6	2.00	22	124.0	33	129.1
Grouping algorithm – lemmatization	4.69	1.50	88.4	4	9	8	2.00	35	178.5	54	185.2
Grouping algorithm – text	5.83	1.79	87.5	5	7	11	2.20	29	212.5	44	220.8
DDD	5.29	1.50	74.1	4	6	9	2.25	27	145.0	39	156.6
Service cutter	3.16	1.15	74.4	3	10	8	2.66	41	202.5	61	206.8
MITIA	6.78	1.06	76.8	4	5	12	3.00	19	190.0	30	203.1
14ms (finer granularity)	8.94	3.44	100.0	14	1	16	1.14	7	95.5	10	130.9
Monolith (greater granularity)	0	0	69.2	1	14	0	0.00	51	32.5	77	46.9

Acop: Acoplamiento, Rend: Rendimiento, N: Número de microservicios, Max. Puntos: Máximos puntos de historia asociados a un microservicio, Tiempo Des: Mayor tiempo de desarrollo estimado de los microservicios.

Al distribuir las historias de usuario de manera diferente, se pueden obtener tiempos de desarrollo más cortos del sistema. Teniendo en cuenta que cada microservicio es desarrollado por un equipo independiente en paralelo. El menor tiempo de desarrollo se obtuvo para la descomposición generada por el algoritmo genético.

En la figura 50 se muestra gráficamente el análisis comparativo de las métricas de evaluación. La figura 51 muestra específicamente la complejidad cognitiva obtenida por los métodos estudiados.

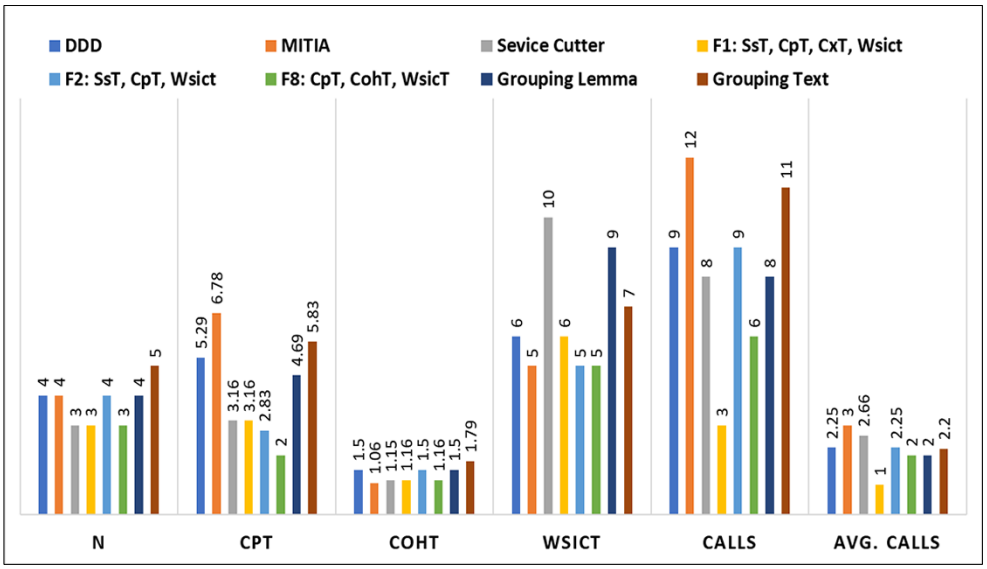


Figura 50. Análisis comparativo de las métricas de evaluación de la aplicación Cargo Tracking.

Donde N: Número de microservicios (MS), CPT: Acoplamiento de MSBA, COHT: Grado de falta cohesión de MSBA, WSICT: Máximo WSIC de MSBA, WSIC es el número de historias de usuarios de MS. CALLS: número de llamadas entre microservicios. Las tres primeras columnas en cada variable corresponden a los métodos de estado del arte, las siguientes tres al algoritmo genético y las dos últimas al algoritmo de agrupamiento.

Según las métricas comparadas, el algoritmo genético obtuvo menor número de microservicios *(N)*, menor acoplamiento *(CpT)*, igual o menor cohesión *(CohT)*, menor número de historias de usuario asociadas a un microservicio (WsicT) y el menor número de llamadas (*calls*) entre microservicios y el menor promedio de llamadas entre los microservicios (Avg. Calls), teniendo así menor comunicación y dependencias.

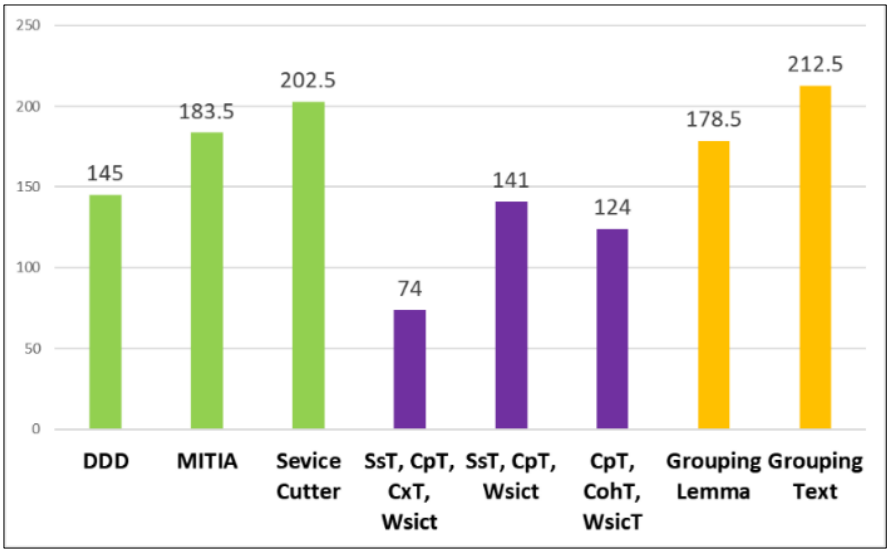


Figura 51. Análisis comparativo de los puntos de complejidad cognitiva de la aplicación Cargo Tracking.

La métrica de la complejidad cognitiva estima la dificultad de entender, implementar y mantener la aplicación basada en los microservicios, dependiendo de la complejidad de cada microservicio, de las interacciones entre ellos y su número. El algoritmo genético del Microservices Backlog propuesto obtuvo un menor número de puntos de complejidad cognitiva que DDD, MITIA, Service Cutter y que el algoritmo de agrupamiento; por lo tanto, con el modelo propuesto se pueden obtener descomposiciones de menor complejidad cognitiva que DDD.

En color verde aparecen los métodos del estado del arte, en color morado los resultados obtenidos por el algoritmo genético y en anaranjado los resultados obtenidos con el algoritmo de agrupamiento. El algoritmo de agrupamiento propuesto usando la lematización de las entidades obtiene una complejidad mayor que DDD y menor que los otros dos métodos del estado del arte. El algoritmo de agrupamiento usando el texto completo de las entidades obtiene mayor complejidad que todos los métodos, el algoritmo de agrupamiento hace más énfasis en la similitud semántica que en otras características.

La figura 52 presenta el gráfico que compara la similitud semántica para las descomposiciones obtenidas con los métodos comparados. Los resultados obtenidos para la similitud semántica son muy parecidos en todas las descomposiciones propuestas. Siendo la mayor la obtenida con el algoritmo de agrupamiento del Microservices Backlog. Se puede resaltar que la similitud semántica para todos los casos supera el 70%, por lo tanto, las descomposiciones son coherentes desde el punto de vista semántico.

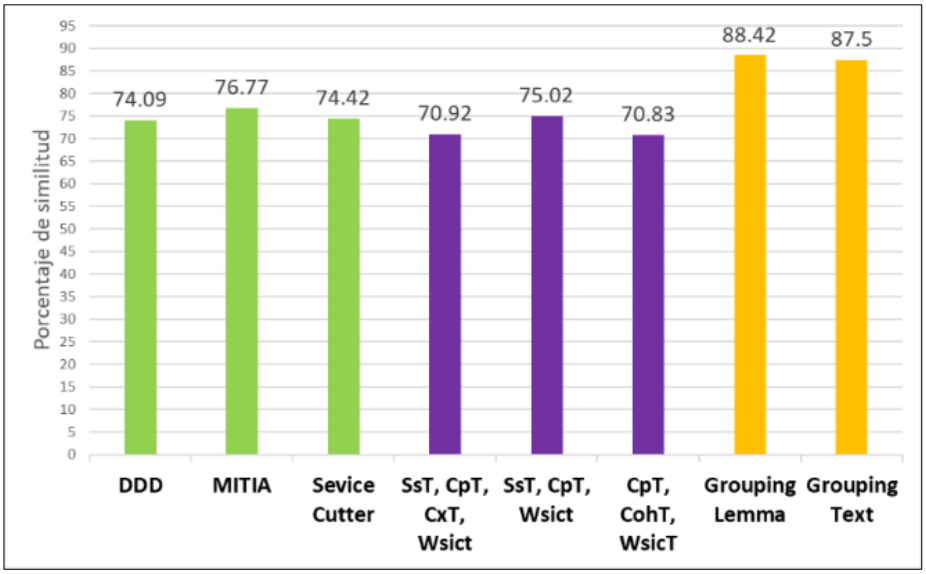


Figura 52. Análisis comparativo de la similitud semántica de la aplicación Cargo Tracking.

La figura 53 presenta la gráfica comparativa de los resultados relacionados con la métrica de granularidad *Gm.* Los valores de menor *Gm* corresponden al algoritmo genético del Microservice Backlog. Por lo tanto, corresponden a la mejor solución al problema propuesto. Se pudo observar que la solución propuesta con menor complejidad, menor acoplamiento y menor número de llamadas corresponde también a la de menor valor de la métrica *GM,* y corresponden al algoritmo genético.

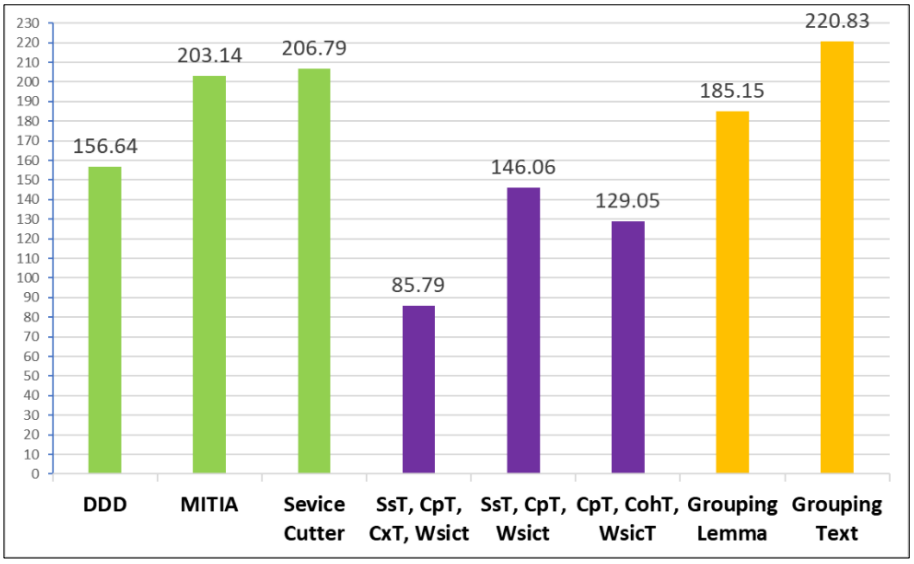


Figura 53. Análisis comparativo de la métrica de granularidad GM para la aplicación Cargo Tracking.

La figura 54 muestra el comparativo entre DDD, el algoritmo genético y el algoritmo de agrupamiento. Se puede apreciar que el algoritmo genético obtiene menor tiempo estimado de desarrollo, menor puntos de historia asociados a un microservicio *(P_i)*, menor llamadas entre microservicios, menor complejidad, mayor cohesión, menor acoplamiento y menor GM que DDD. El algoritmo de agrupamiento obtiene valores cercanos a DDD, por lo tanto, puede usarse también para obtener descomposiciones del "Product Backlog" a microservicios.

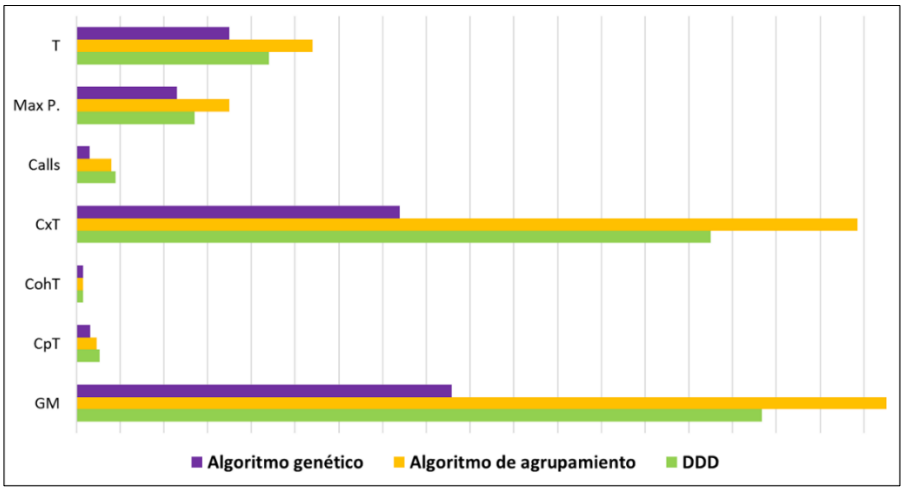


Figura 54. Análisis comparativo de los resultados obtenidos con el algoritmo genético, el algoritmo de agrupamiento para Cargo Tracking.

Donde T es el tiempo estimado de desarrollo de la descomposición obtenida por cada algoritmo y método.

5.4.2. Aplicación JPet Store

La aplicación JPet-Store corresponde a una tienda de mascotas en línea. En la cual se puede navegar y buscar en el catálogo de productos, elegir los artículos para añadirlos al carrito de compras, modificar el carrito de compras y pedir los artículos del carrito de compras. Se puede realizar muchas de estas acciones sin necesidad de registrarte ni de iniciar sesión en la

aplicación. Sin embargo, antes de poder pedir los artículos, debe iniciar sesión (log in) en la aplicación. Para iniciar sesión, debe tener una cuenta en la aplicación, que se crea cuando se registra (se inscribe) en la aplicación [125]. El resumen de las funcionalidades de la Jpet Store se enumeran a continuación.

- Inscripción
- Inicio de sesión
- Trabajando con el Catálogo de Productos: Navegando por el catálogo, buscando en el catálogo
- Trabajar con el carrito de compras: Añadir y quitar artículos, actualizar la cantidad de un artículo, pedir artículos, revisar un pedido

Esta aplicación ha sido utilizada para validar los métodos de descomposición y las migraciones de las aplicaciones monolíticas a microservicios [61], [72], [78]. A partir de la descripción, la definición y el código fuente de la aplicación, se identificaron las historias de usuarios y el "Product Backlog". Se identificaron primero las operaciones; luego a partir de ellas se especificaron y estimaron las historias de usuario. Ver tabla 19.

Tabla 19. Product backlog for Jpet-Store application

Id.	Nombre	Puntos	Tiemp des (hours)
HU_1	View category	3	5
HU_2	List categories	1	3
HU_3	Search products	5	7
HU_4	View product	3	5
HU_5	View item (get item)	3	5
HU_6	Add item to cart	5	7
HU_7	Remove item from cart	3	5
HU_8	Update cart quantities	3	5
HU_9	Get cart	3	5
HU_{10}	New order	7	10
HU_{11}	Get order	3	5
HU_{12}	Set order id	5	7
HU_{13}	List orders	3	5
HU_{14}	Is authenticated	3	5
HU_{15}	New account (sign up)	5	7
HU_{16}	Get account	3	5
HU_{17}	Sign off	2	3
HU_{18}	Update account	3	5
HU_{19}	Getcategory	2	3
HU_{20}	Getproduct	2	3
HU_{21}	Is item in stock	3	5
HU_{22}	Sign in	3	5
Total		73	115

Los puntos y tiempos se estimaron de acuerdo con nuestra experiencia y corresponden al esfuerzo y tiempo que se llevaría en desarrollar cada historia de usuario; el tiempo total corresponde a un orden secuencial de desarrollo de las historias de usuario. En un desarrollo real de software utilizando metodologías ágiles, esta estimación la haría el equipo de desarrollo en la planificación de la entrega considerando sus propias características y velocidad del equipo de desarrollo. El texto de las historias de usuario para este caso se usó en inglés.

Se define una dependencia cuando una historia de usuario utiliza o llama a otra historia de usuario. Este ejemplo puede considerarse como una migración de monolito a microservicios, en este caso las historias de usuario pueden ser sustituidas por las operaciones/métodos o servicios de la aplicación; en este caso una dependencia corresponde a una dependencia de ejecución, en la que una operación llama a otra operación para cumplir su propósito. En este ejemplo, el código fuente de la aplicación monolítica estaba disponible. Para definir las dependencias entre historias de usuarios, se analizó el código fuente para identificar las dependencias de invocación

entre historias de usuarios y/o operaciones (OrderService, CatalogService, AcountService, entidad Cart, y otras entidades). El proceso se detalla a continuación:

HU_1 - Ver categoría- View category
ViewCategory → List Categories – Corresponde a otra historia de usuario. *(HU_2)*
ViewCategory → catalogService.getProductListByCategory – Es la misma historia de usuario, es su implementación.
ViewCategory → catalogService.getCategory(id) – Corresponde a otra historia de usuario. *(HU_{19})*
Dependencias: *HU_1 = {HU_2, HU_{19}}*

HU_2 - Listar categorías - List categories
ListCategories → accountAction.getCategories() – Implementación de la historia de usuario
No tiene dependencias

HU6 - Agregar item o artículo al carrito de compras - Add item to cart
AddItemtoCart → cart.incrementQuantityByItemId(workingItemId); - Corresponde a la implementación de esta historia
AddItemtoCart → catalogService.isItemInStock(workingItemId); - Corresponde a otra historia de usuario. *(HU_{21})*
AddItemtoCart → catalogService.getItem(workingItemId); - Corresponde a otra historia de Usuario: ver item *(HU_5)*
Dependencias: *HU6 = {HU_5, HU_{21}}*

Este proceso se realizó para todas las historias de usuario, la tabla 20 muestra el resultado obtenido.

Tabla 20. User stories dependencies for JPet Store application

Id.	Dependencias	Id.	Dependencias
HU_1	{HU_2, HU_{19}}	HU_{12}	{}
HU_2	{}	HU_{13}	{HU_{16}}
HU_3	{}	HU_{14}	{}
HU_4	{HU_{20}}	HU_{15}	{HU_{16}, HU_1}
HU_5	{}	HU_{16}	{}
HU_6	{HU_5, HU_{21}}	HU_{17}	{}
HU_7	{}	HU_{18}	{HU_{16}, HU_1}
HU_8	{}	HU_{19}	{}
HU_9	{}	HU_{20}	{}
HU_{10}	{HU_9, HU_{16}}	HU_{21}	{}
HU_{11}	{HU_{16}}	HU_{22}	{HU_1}

Se probaron diferentes funciones objetivo en el algoritmo genético, también se usó el algoritmo de agrupamiento (Lemmatización y texto completo). Se compararon las descomposiciones obtenidas por el Microservices Backlog con DDD, y el método “Execution Traces”. Ver figura 55.

Los resultados obtenidos por DDD y “Execution Traces” fueron los mismos, sólo una historia de usuario estaba en un microservicio diferente, pero las métricas y la comparación fueron iguales. La tabla 21 muestra las descomposiciones obtenidas. Los resultados obtenidos automáticamente del Microservices Backlog obtuvo más microservicios candidatos que DDD y “Execution Traces”, el usuario hizo algunas operaciones de unión de microservicios y obtuvo cinco microservicios, esta solución se acercó a DDD y “Execution Traces”.

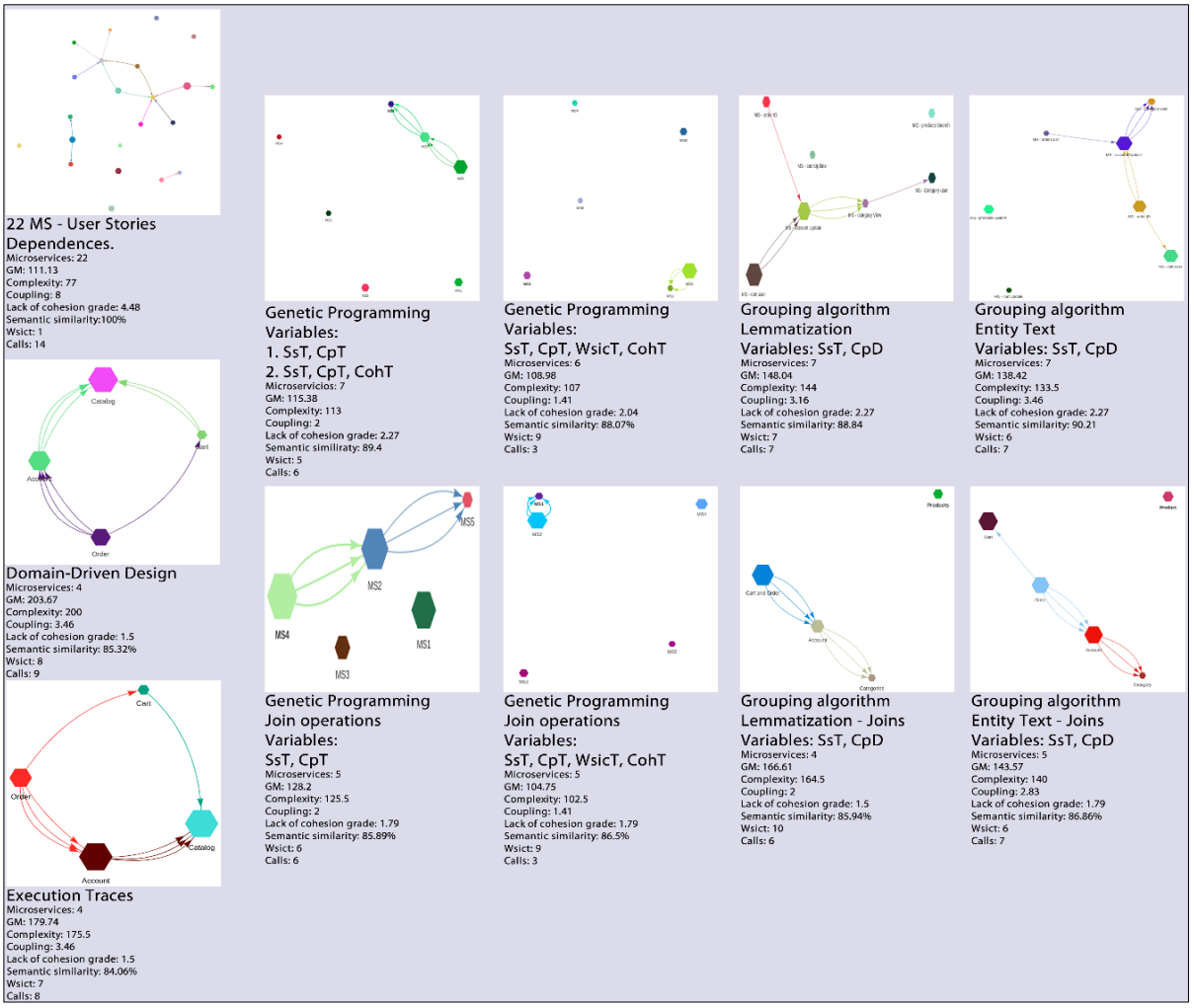


Figura 55. Microservices Backlog comparado con DDD y "Execution Traces" para la aplicación JPet Store.

Los resultados obtenidos por DDD y "Execution Traces" fueron los mismos, sólo una historia de usuario estaba en un microservicio diferente, pero las métricas y la comparación fueron iguales. La tabla 21 muestra las descomposiciones obtenidas. Los resultados obtenidos automáticamente del Microservices Backlog obtuvo más microservicios candidatos que DDD y "Execution Traces", el usuario hizo algunas operaciones de unión de microservicios y obtuvo cinco microservicios, esta solución se acercó a DDD y "Execution Traces".

Las descomposiciones propuestas por el Microservice backlog tuvo coherencia semántica y son una buena solución al problema. El análisis comparativo de las métricas se detalla en la tabla 22. La descomposición que tuvo más acoplamiento fue 22MS (la granularidad más fina), si se sigue el principio de responsabilidad única, se debe asociar una historia de usuario con un solo microservicio; con los casos evaluados se evidencia que el acoplamiento de la aplicación global aumenta al tener más microservicios y más dependencias; por lo tanto, el principio de responsabilidad única se puede interpretar como "agrupar las cosas que se refieren a las mismas cosas", por lo tanto, la similitud semántica es fundamental para agrupar cosas similares. El Microservices Backlog agrupó las historias de usuarios referidas a la misma entidad manteniendo un bajo acoplamiento y una alta cohesión.

Tabla 21. Comparación de las descomposiciones obtenidas por métodos evaluados

Método	N	Descomposiciones a Microservicios
Genetic algorithm F5: CpT, SsT	7	MS_1 = {HU_5, HU_6, HU_{21}}, MS_2 = {HU_7, HU_8}, MS_3 = {HU_3, HU_4, HU_{20}}, MS_4 = {HU_{14}, HU_{17}}, MS_5 = {HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{13}}, MS_6 = {HU_1, HU_2, HU_{19}}, MS_7 = {HU_{15}, HU_{16}, HU_{18}, HU_{22}}
Genetic algorithm and Joins F5: CpT, SsT	5	MS_1 = {HU_5, HU_6, HU_7, HU_8, HU_{21}}, MS_2 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}} MS_3 = {HU_3, HU_4, HU_{20}}, MS_4 = {HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{13}} MS_5 = {HU_1, HU_2, HU_{19}}
Genetic algorithm F6: CpT, CohT, WsicT, SsT	6	MS_1 = {HU_1, HU_2, HU_{19}}, MS_2 = {HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{13}, HU_{15}, HU_{16}, HU_{18}, HU_{22}}, MS_3 = {HU_3, HU_4, HU_{20}}, MS_4 = {HU_7, HU_8}, MS_5 = {HU_5, HU_6, HU_{21}}, MS_6 = {HU_{14}, HU_{17}}
Genetic algorithm and Joins F6: CpT, CohT, WsicT, SsT	5	MS_1 = {HU_1, HU_2, HU_{19}}, MS_2 = {HU_{14}, HU_{17}}, MS_3 = {HU_3, HU_4, HU_{20}} MS_4 = {HU_5, HU_6, HU_7, HU_8, HU_{21}} MS_5 = {HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{13}, HU_{15}, HU_{16}, HU_{18}, HU_{22}}
Grouping algorithm Lemmatization	7	MS_1 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}}, MS_2 = {HU_8} MS_3 = {HU_5, HU_6, HU_7, HU_9, HU_{10}, HU_{11}, HU_{21}}, MS_4 = {HU_4, HU_{19}, HU_{20}} MS_5 = {HU_1, HU_2}, MS_6 = {HU_{12}, HU_{13}}, MS_7 = {HU_3}
Grouping algorithm Lemmatization and Joins	4	MS_1 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}} MS_2 = {HU_5, HU_6, HU_7, HU_8, HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{13}, HU_{21}}, MS_3 = {HU_1, HU_2, HU_{19}} MS_4 = {HU_3, HU_4, HU_{20}}
Grouping algorithm Text	7	MS_1 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}}, MS_2 = {HU_8}, MS_3 = {HU_5, HU_6, HU_7, HU_9, HU_{21}}, MS_4 = {HU_1, HU_2, HU_{19}}, MS_5 = {HU_{10}, HU_{11}, HU_{12}}, MS_6 = {HU_{13}}, MS_7 = {HU_3, HU_4, HU_{20}}
Grouping algorithm Text and Joins	5	MS_1 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}}, MS_2 = {HU_5, HU_6, HU_7, HU_8, HU_9, HU_{21}} MS_3 = {HU_1, HU_2, HU_{19}}, MS_4 = {HU_{10}, HU_{11}, HU_{12}, HU_{13}} MS_5 = {HU_3, HU_4, HU_{20}}
DDD	4	MS_1 = {HU_1, HU_2, HU_3, HU_4, HU_5, HU_{19}, HU_{20}, HU_{21}} MS_2 = {HU_6, HU_7, HU_8, HU_9}, MS_3 = {HU_{10}, HU_{11}, HU_{12}, HU_{13}} MS_4 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}}
Execution Traces	4	MS_1 = {HU_1, HU_2, HU_3, HU_4, HU_5, HU_{19}, HU_{20}} MS_2 = {HU_6, HU_7, HU_8, HU_9, HU_{21}} MS_3 = {HU_{10}, HU_{11}, HU_{12}, HU_{13}} MS_4 = {HU_{14}, HU_{15}, HU_{16}, HU_{17}, HU_{18}, HU_{22}}

Método: Método evaluado. N: número de microservicios. MS_i: i-ésimo microservicio. HU_k: k-ésima historia de Usuario asociada a MSi.

Table 22. Análisis comparativo de las descomposiciones para aplicación Jpet-Store

Método / métricas	Acop.	Cohesión		Granularidad		Rendimiento		Complejidad		T	Gm
	CpT	CohT	SsT (%)	N	WsicT	Calls	Avg. Calls	Max. Pi	CxT		
Genetic algorithm F4: cpt, coht, sst F5: cpt, sst	2.00	2.27	89.4	7	5	6	0.86	21	113.0	32	115.4
Genetic algorithm and joins. Cpt, sst	2.00	1.79	85.9	5	6	6	1.20	21	125.5	32	128.2
Genetic algorithm F1: cpt, coht, wsict, sst	1.41	2.04	88.1	6	9	3	0.50	35	107.0	54	109.0
Genetic algorithm and joins. Cpt, coht, wsict, sst	1.41	1.79	86.5	5	9	3	0.60	35	102.5	54	104.7
Grouping algorithm - lemma	3.16	2.27	88.8	7	7	7	1.00	27	144	42	148.0
Grouping algorithm – lemma and joins	2.00	1.50	85.9	4	10	6	1.50	38	164.5	59	166.6
Grouping algorithm – Text	3.46	2.27	90.2	7	6	7	1.00	19	133.5	30	138.4
Grouping algorithm – text and joins	2.83	1.79	86.6	5	6	7	1.40	20	140	32	143.6
DDD	3.46	1.50	85.3	4	8	9	2.25	22	200.0	36	203.7
Execution traces	3.46	1.50	84.1	4	7	8	2.00	19	175.5	31	179.7
22ms (finer granularity)	8.00	4.48	100.0	22	1	14	0.63	7	77	10	111.1
Monolith (greater granularity)	0.00	0.00	70.8	1	22	0	0.00	73	47.5	115	22

El Microservices Backlog obtuvo el menor acoplamiento para esta aplicación, la falta de cohesión fue muy similar en todos los casos, también obtuvo el menor *WsicT* y la menor complejidad en comparación con DDD y trazas de ejecución. El tiempo de desarrollo estimado de la solución propuesta por el algoritmo genético fue menor que el DDD y cercano a las trazas de ejecución; el algoritmo de agrupamiento obtuvo el menor tiempo estimado de desarrollo, así como el máximo

número de puntos de historia de usuario asociados a un microservicio entre los métodos evaluados.

La solución propuesta por el algoritmo genético obtuvo el menor acoplamiento, la menor complejidad, el menor número de llamadas entre microservicios (“calls”); también obtuvo valores muy similares para la cohesión, la similitud semántica, en número máximos de historias de usuario, puntos y tiempo estimado de desarrollo comparado con los otros métodos. Por lo tanto, el Microservice Backlog propone descomposiciones a microservicios válidas y coherentes para la aplicación Jpet-Store.

En las figuras 56 y 57 se muestra gráficamente el análisis comparativo de las métricas y la complejidad cognitiva de las descomposiciones obtenidas para la aplicación JPet-Store. Las dos primeras barras corresponden a DDD y a “Execution Traces”, las otras cuatro corresponden al algoritmo genético y las cuatro últimas al algoritmo de agrupamiento.

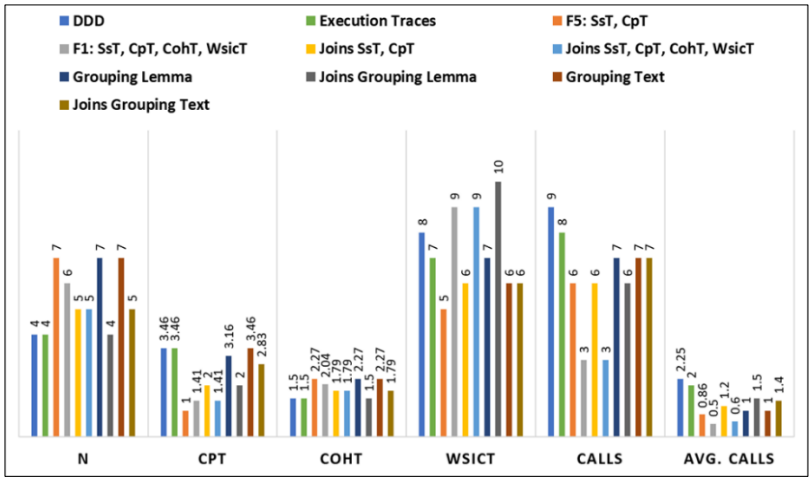


Figura 56. Análisis comparativo de las métricas de evaluación de la aplicación Jpet-Store.

Dónde MS: Número de microservicios, CPT: Acoplamiento de MSBA, COHT: Falta de grado de cohesión de MSBA, WSICT: Máximo WSIC de MSBA, WSIC es el número de historias de usuarios de MS. CALLS: número de llamadas entre microservicios. SsT: Similitud semántica. AVG Calls: Es el promedio de llamadas entre los microservicios de la aplicación.

Los resultados de Microservices Backlog fueron similares en ambas aplicaciones JPet-Store y Cargo-Tracking, nuestro modelo obtuvo un bajo acoplamiento, una baja o similar falta de cohesión, una alta cohesión semántica, menor llamadas entre microservicios y una baja complejidad que los métodos propuesto en el estado del arte.

El cambio de una historia de usuario u operación de un microservicio a otro puede tener consecuencias en el rendimiento, la complejidad y el acoplamiento de la aplicación; por lo tanto, es un punto importante a considerar cuando se diseñan aplicaciones basadas en microservicios, y debe hacerse en base a métricas como las propuestas en este trabajo de investigación, donde se puede analizar el impacto de esos cambios y las diferentes distribuciones de las historias de usuario u operaciones en los microservicios, todo ello en tiempo de diseño.

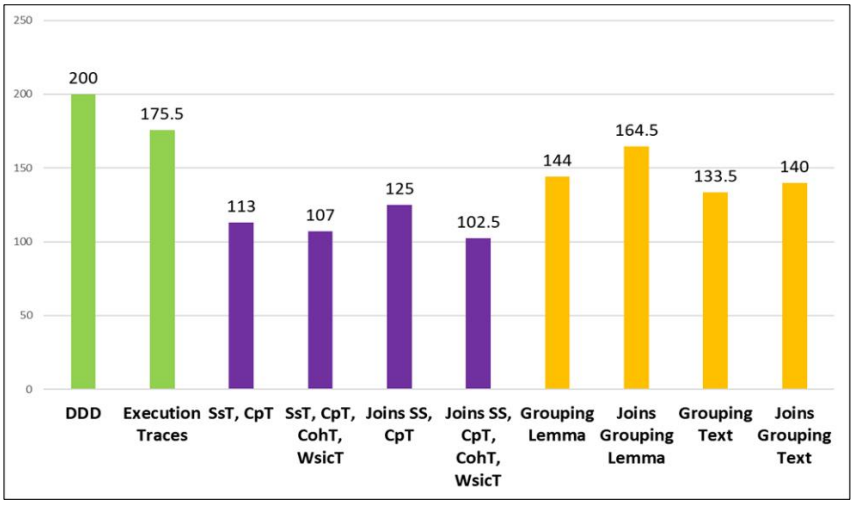


Figure 57. Análisis comparativo de los puntos de complejidad cognitiva de la aplicación Jpet-Store.

Los valores de menor complejidad corresponden al algoritmo genético, seguido por el algoritmo de agrupamiento, entonces el Microservice Backlog pudo obtener descomposiciones de baja complejidad comparada con DDD y con Execution Traces. La figura 58 muestra el comparativo de la similitud semántica.

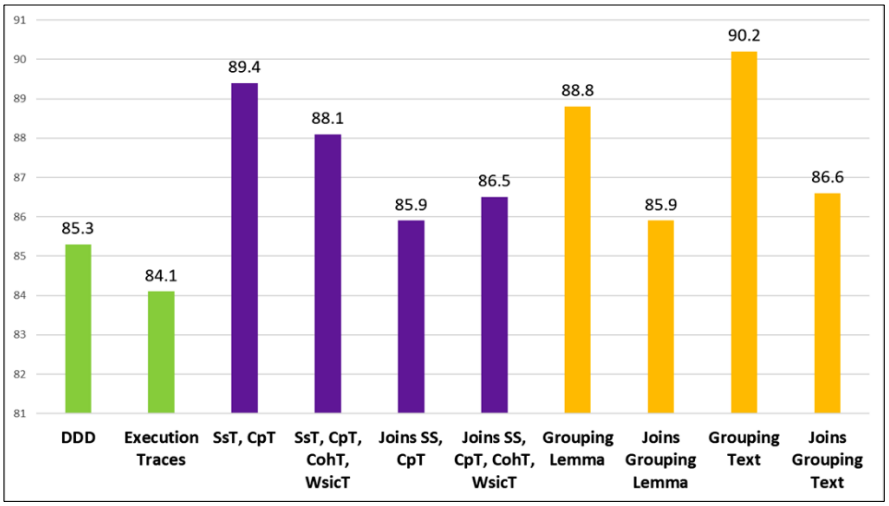


Figure 58. Análisis comparativo de la similitud semántica de la aplicación Jpet-Store.

Tanto el algoritmo genético como el algoritmo de agrupamiento del Microservice Backlog obtuvieron mayor similitud semántica que DDD y “Execution Traces”; adicionalmente obtuvieron un valor alto de similitud siendo mayor al 85% para todos los casos; entonces el Microservices Backlog obtuvo descomposiciones coherentes desde el punto de vista semántico, indicando una alta cohesión semántica. La figura 59 muestra la gráfica que compara el valor de la métrica de granularidad *Gm* obtenida por los casos evaluados.

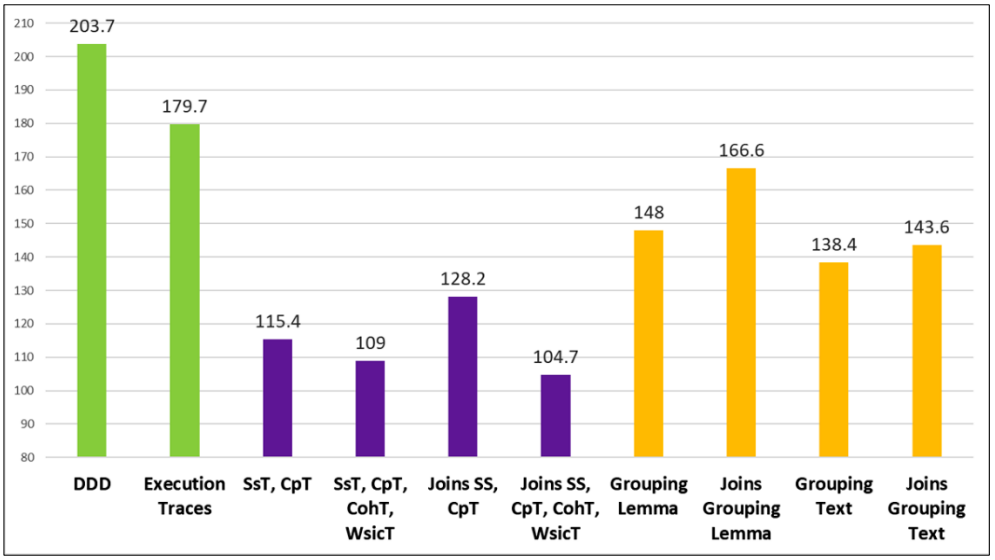


Figure 59. Análisis comparativo de la métrica de granularidad *Gm* de la aplicación Jpet-Store.

Se puede apreciar que las soluciones propuestas por el algoritmo genético obtuvieron menor *Gm* que DDD, "Execution Traces" y el algoritmo de agrupamiento. En esta aplicación el algoritmo de agrupamiento tuvo un valor menor de GM comparado con DDD y "Execution Traces". La figura 60 muestra el resultado del comparativo de este caso de estudio.

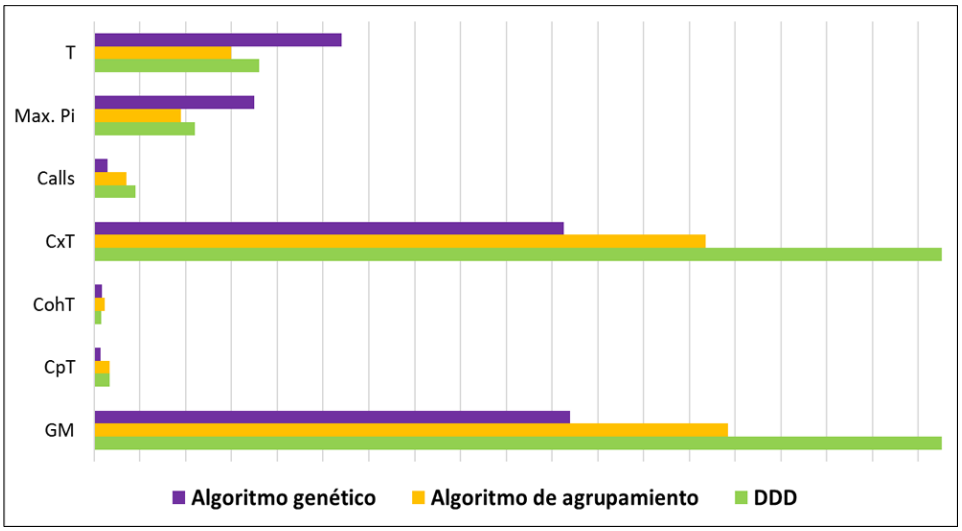


Figure 60. Análisis comparativo de los resultados obtenidos con el algoritmo genético, el algoritmo de agrupamiento y DDD para la aplicación Jpet-Store.

El algoritmo genético obtuvo menor acoplamiento, mayor cohesión, menor complejidad, una alta cohesión semántica, menos llamadas entre los microservicios y menor *Gm* comparado con DDD. DDD obtuvo mayor menor tiempo estimado de desarrollo y menor puntos de historia asociados con un microservicio.

El Microservices Backlog obtuvo para los proyectos hipotéticos Cargo Tracking y Jpet-Store un diseño para la aplicación basada en microservicios (descomposición) con menor acoplamiento, menor complejidad, menos comunicación entre los microservicios (menos "calls"), mayor cohesión, menor valor *Gm* y menor tiempo de desarrollo comparado con DDD. Este es un

resultado prometedor e importante porque DDD es uno de los métodos más utilizados para definir la granularidad de los microservicios.

Los dos ejemplos estudiados hasta este momento son casos hipotéticos con pocas historias de usuario, en los cuales los resultados fueron satisfactorios; luego se analizó el Microservices Backlog en dos proyectos reales, el primero corresponde al sistema de gestión de conferencias virtuales llamado “Foristom Conferences” de la fundación Foristom**1**. El otro corresponde a un proyecto de los grupos de investigación Grinder y GEDI de la Universidad del Valle y al grupo de investigación GIA de la Universidad Francisco de Paula Santander, en el cual se está implementando un sistema de información para la planificación del entrenamiento deportivo llamado Sinplafut, el cual fue introducido en la sección 3.1 y se desarrolló un prototipo inicial de la aplicación usando la arquitectura de microservicios.

5.4.3. Aplicación Foristom Conferencias

Foristom Conferencias es una aplicación web que permite gestionar la información y organización de conferencias virtuales patrocinadas por la fundación Foristom. Este sistema permite gestionar desde la creación y divulgación de la conferencia hasta la publicación y presentación de los artículos enviados. Actualmente se encuentra en fase de desarrollo. Foristom Conferencias es un sistema de información que permite gestionar, organizar y coordinar la logística de una conferencia virtual de difusión del conocimiento académico-científico derivado de las investigaciones, desarrollos tecnológicos e innovaciones realizadas por estudiantes, profesores, e investigadores.

Tabla 23. Product Backlog para la aplicación Foristom Conferencias.

Id.	Nombre	Puntos	Tiempo des. (hours)
HU_1	Create conference.	13	27
HU_2	Get conference by id.	2	3
HU_3	Edit conference	5	9
HU_4	Delete conference	5	9
HU_5	Upload conference’s support files	8	16
HU_6	Get conference’s support files	2	3
HU_7	Generate conference landing page.	8	16
HU_8	Generate call for papers.	13	27
HU_9	Apply filters and specialized search on conferences.	8	16
HU_{10}	Generate conference history.	8	16
HU_{11}	Register on the conferences system.	8	16
HU_{12}	Log in the conferences system.	8	16
HU_{13}	List submitted papers by state.	2	3
HU_{14}	Submit a paper to the conference.	13	27
HU_{15}	Register the paper evaluation.	8	16
HU_{16}	Generate paper evaluation board.	5	9
HU_{17}	Register for a conference as an attendee or author	13	27
HU_{18}	Activate user (attendee or author) registration	8	16
HU_{19}	Get the list of registrations by state.	2	3
HU_{20}	Pay the conference registration fee online.	13	27
HU_{21}	Get the conferences that are taking place "on-line"	13	27
HU_{22}	Generate the home page of the conference.	13	27
HU_{23}	List conference program	5	9
HU_{24}	Create conference program, keynotes, and sessions.	13	27
HU_{25}	Enter to the meeting room as a speaker or assistant.	8	16
HU_{26}	Upload files of the presentation or session (video, photo, presentation)	5	9
HU_{27}	Generate abstract book.	13	27
HU_{28}	Download certificate of attendees and speakers.	8	16
HU_{29}	Register the presentation of a paper.	5	9
	Total	235	469

1 www.foristom.org

La aplicación web permite gestionar la información, así como generar visibilidad y divulgar el conocimiento académico y científico. Asimismo, esta aplicación web permite a los participantes de la conferencia seleccionar el workshop que desean presenciar según el tema propuesto en la conferencia. Los participantes tendrán la opción de registrarse como asistentes o ponentes, así como de enviar el resumen del trabajo de investigación que será incluido en la programación de la conferencia. Además, se podrá descargar de la página web de la conferencia y de forma digital el programa, el libro de resúmenes, así como el certificado de ponente o asistente. El "product backlog" de Foristom Conferences se puede apreciar en la tabla 23.

En este caso, las dependencias se definieron de acuerdo con la lógica del negocio de la aplicación y el flujo de datos entre las diferentes historias de usuarios. Ver tabla 24.

Tabla 24. Dependencias entre historias de usuario para Foristom Conferences

Id.	Dependencias	Id.	Dependencias
HU_1	{}	HU_{16}	{HU_{13}, HU_{15}}
HU_2	{}	HU_{17}	{HU_{20}}
HU_3	{HU_2}	HU_{18}	{HU_{19}}
HU_4	{HU_2}	HU_{19}	{}
HU_5	{HU_2, HU_6}	HU_{20}	{}
HU_6	{}	HU_{21}	{HU_9}
HU_7	{HU_2}	HU_{22}	{HU_{23}, HU_{13}, HU_{28}}
HU_8	{HU_9}	HU_{23}	{}
HU_9	{}	HU_{24}	{HU_{13}}
HU_{10}	{HU_9}	HU_{25}	{HU_{26}, HU_{29}}
HU_{11}	{}	HU_{26}	{}
HU_{12}	{HU_{11}, HU_8}	HU_{27}	{HU_{13}}
HU_{13}	{}	HU_{28}	{}
HU_{14}	{}	HU_{29}	{}
HU_{15}	{}		

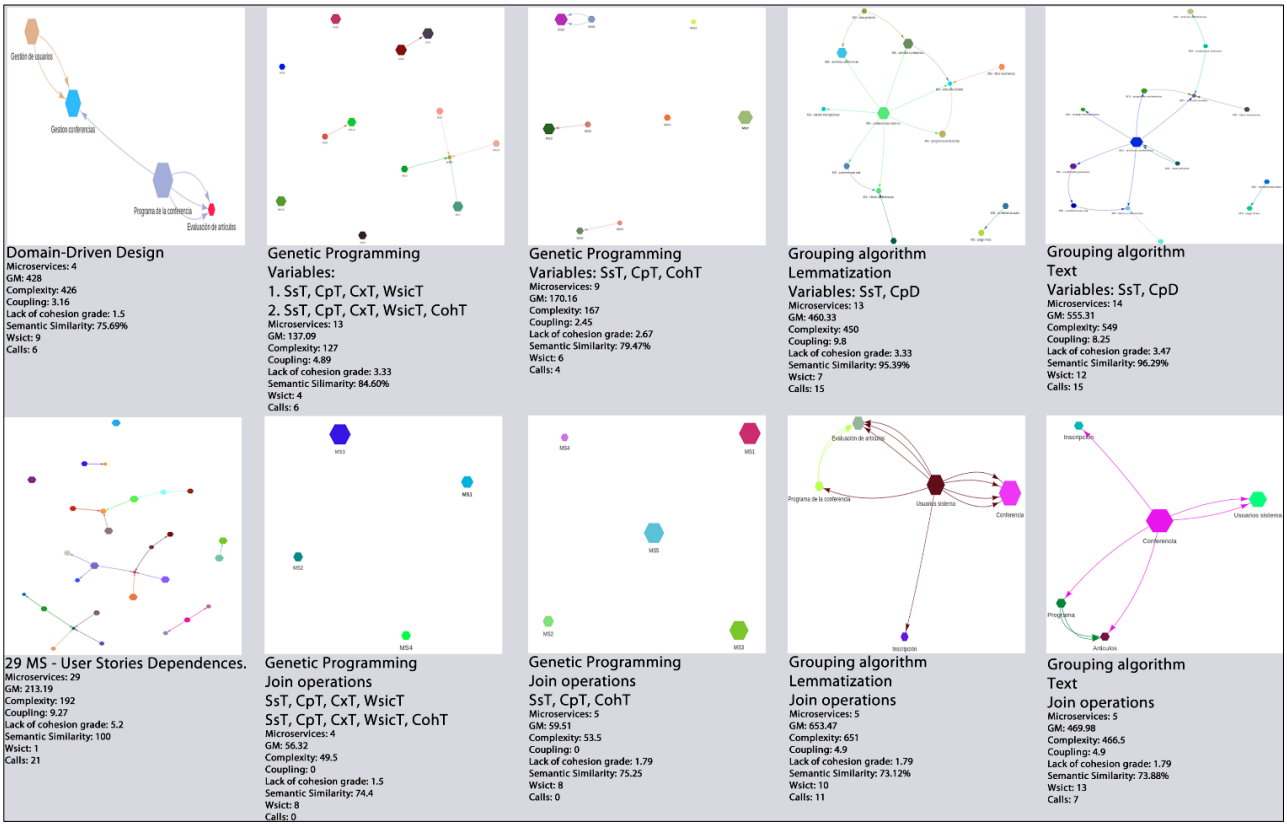


Figura 61. El modelo Microservices Backlog comparado con DDD para la aplicación Foristom Conferencias.

El objetivo fue comparar el diseño propuesto por el arquitecto o equipo de desarrollo usando DDD contra el diseño obtenido con nuestro modelo el Microservice Backlog, usando el algoritmo genético y el algoritmo de agrupamiento. Se usó DDD siguiendo los enfoques propuestos por Evan [122],[24], quien establece que primero se deben identificar las entidades, luego los objetos que aportan valor, y los contextos delimitados para proponer los microservicios que van a formar parte de la aplicación.

La figura 61 muestra las propuestas de descomposición para el sistema de conferencias de Foristom. El número de microservicios y sus detalles se presentan en la Tabla 25.

Tabla 25. Comparación de las descomposiciones obtenidas por los métodos evaluados

Método	N	Descomposición a Microservicios
Genetic algorithm F1: CpT, CxT, WsicT, SsT	13	$MS_1 = \{HU_9, HU_{10}, HU_{21}\}$, $MS_2 = \{HU_{24}\}$, $MS_3 = \{HU_{27}\}$ $MS_4 = \{HU_5, HU_6\}$, $MS_5 = \{HU_{25}, HU_{26}, HU_{29}\}$, $MS_6 = \{HU_{18}, HU_{19}\}$ $MS_7 = \{HU_{22}, HU_{23}, HU_{28}\}$, $MS_8 = \{HU_1, HU_{14}\}$, $MS_9 = \{HU_8, HU_{11}, HU_{12}\}$, $MS_{10} = \{HU_{17}, HU_{20}\}$, $MS_{11} = \{HU_{13}\}$ $MS_{12} = \{HU_{15}, HU_{16}\}$, $MS_{13} = \{HU_2, HU_3, HU_4, HU_7\}$
Genetic algorithm and Joins F1: CpT, CxT, WsicT, SsT	4	$MS_1 = \{HU_8, HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{21}\}$ $MS_2 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_7, HU_{14}\}$ $MS_3 = \{HU_{13}, HU_{15}, HU_{16}, HU_{22}, HU_{23}, HU_{24}, HU_{27}, HU_{28}\}$ $MS_4 = \{HU_{17}, HU_{18}, HU_{19}, HU_{20}, HU_{25}, HU_{26}, HU_{29}\}$
Genetic algorithm F4: CpT, CohT, SsT	9	$MS_1 = \{HU_{18}, HU_{19}\}$, $MS_2 = \{HU_{13}, HU_{15}, HU_{16}, HU_{23}, HU_{24}, HU_{27}\}$ $MS_3 = \{HU_8, HU_9, HU_{10}, HU_{21}\}$, $MS_4 = \{HU_{25}, HU_{26}, HU_{29}\}$ $MS_5 = \{HU_{11}, HU_{12}\}$, $MS_6 = \{HU_2, HU_3, HU_4, HU_7\}$ $MS_7 = \{HU_1, HU_{14}, HU_{17}, HU_{20}\}$, $MS_8 = \{HU_{22}, HU_{28}\}$ $MS_9 = \{HU_5, HU_6\}$
Genetic algorithm and Joins CpT, CohT, SsT	5	$MS_1 = \{HU_{13}, HU_{15}, HU_{16}, HU_{22}, HU_{23}, HU_{24}, HU_{27}, HU_{28}\}$ $MS_2 = \{HU_2, HU_3, HU_4, HU_5, HU_6, HU_7\}$ $MS_3 = \{HU_8, HU_9, HU_{10}, HU_{11}, HU_{12}, HU_{21}\}$ $MS_4 = \{HU_{25}, HU_{26}, HU_{29}\}$ $MS_5 = \{HU_1, HU_{14}, HU_{17}, HU_{18}, HU_{19}, HU_{20}\}$
Grouping algorithm lemmatization	13	$MS_1 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_{26}\}$ $MS_2 = \{HU_{14}, HU_{15}, HU_{16}, HU_{28}, HU_{29}\}$ $MS_3 = \{HU_{13}\}$, $MS_4 = \{HU_{17}\}$, $MS_5 = \{HU_{21}\}$, $MS_6 = \{HU_8\}$, $MS_7 = \{HU_7, HU_{10}, HU_{11}, HU_{12}, HU_{18}, HU_{22}\}$ $MS_8 = \{HU_9\}$, $MS_9 = \{HU_{27}\}$, $MS_{10} = \{HU_{19}\}$, $MS_{11} = \{HU_{20}\}$ $MS_{12} = \{HU_{23}, HU_{24}\}$, $MS_{13} = \{HU_{25}\}$
Grouping algorithm and Joins - lemmatization	5	$MS_1 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_8, HU_9, HU_{21}, HU_{26}\}$ $MS_2 = \{HU_{13}, HU_{14}, HU_{15}, HU_{16}, HU_{28}, HU_{29}\}$ $MS_3 = \{HU_{17}, HU_{19}, HU_{20}\}$ $MS_4 = \{HU_{23}, HU_{24}, HU_{27}\}$ $MS_5 = \{HU_7, HU_{10}, HU_{11}, HU_{12}, HU_{18}, HU_{22}, HU_{25}\}$
Grouping algorithm text	14	$MS_1 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_7, HU_{10}, HU_{18}, HU_{22}, HU_{26}, HU_{29}\}$ $MS_2 = \{HU_{14}, HU_{15}\}$, $MS_3 = \{HU_{13}\}$, $MS_4 = \{HU_{11}, HU_{12}, HU_{28}\}$ $MS_5 = \{HU_{17}\}$, $MS_6 = \{HU_{21}\}$, $MS_7 = \{HU_8\}$, $MS_8 = \{HU_{16}\}$ $MS_9 = \{HU_9\}$, $MS_{10} = \{HU_{27}\}$, $MS_{11} = \{HU_{19}\}$, $MS_{12} = \{HU_{20}\}$ $MS_{13} = \{HU_{23}, HU_{24}\}$, $MS_{14} = \{HU_{25}\}$
Grouping algorithm and Joins text	5	$MS_1 = \{HU_{13}, HU_{14}, HU_{15}, HU_{16}\}$ $MS_2 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_7, HU_{10}, HU_{18}, HU_{22}, HU_{25}, HU_{26}, HU_{29}\}$ $MS_3 = \{HU_{17}, HU_{19}, HU_{20}\}$ $MS_4 = \{HU_{23}, HU_{24}, HU_{27}\}$ $MS_5 = \{HU_8, HU_9, HU_{11}, HU_{12}, HU_{21}, HU_{28}\}$
DDD	4	$MS_1 = \{HU_1, HU_2, HU_3, HU_4, HU_5, HU_6, HU_8, HU_9, HU_{10}\}$ $MS_2 = \{HU_{13}, HU_{14}, HU_{15}, HU_{16}\}$ $MS_3 = \{HU_{21}, HU_{22}, HU_{23}, HU_{24}, HU_{25}, HU_{26}, HU_{27}, HU_{28}, HU_{29}\}$ $MS_4 = \{HU_7, HU_{11}, HU_{12}, HU_{17}, HU_{18}, HU_{19}, HU_{20}\}$

Método: método evaluado. N: número de microservicios. MS_i: i-ésimo microservicio. HU_k: k-ésima historia de Usuario asociada MS_i.

Para los microservicios identificados, se calcularon las métricas de complejidad, acoplamiento, granularidad, llamadas entre los microservicios y se estimó el tiempo de desarrollo. De esta manera, el arquitecto puede observar gráficamente varias propuestas de solución, compararlas,

evaluarlas y seleccionar la que quiere implementar de acuerdo con sus necesidades y requerimientos.

Igualmente, que en los casos de estudio anteriores se probaron diferentes variables en la función objetivo del algoritmo genético.

El mayor acoplamiento lo obtuvo la descomposición 29MS (la granularidad más fina), el acoplamiento de la descomposición obtenido por nuestro modelo fue cero, esto significa que los microservicios identificados no tienen dependencias y pueden funcionar por sí solos. La solución propuesta por el Microservices Backlog tiene menos *WsicT*, menor complejidad cognitiva, menos tiempo estimado de desarrollo y menos puntos de historia de usuario que DDD; se puede resaltar que la comunicación entre los microservicios fue cero. Se mantiene la tendencia encontrada en los casos de estudio Cargo-Tracking y JPet-Store.

El algoritmo de agrupamiento en primera instancia obtiene varios microservicios con sólo una historia de usuario, debido a que no encuentra similitud semántica entre la historia de usuario con las demás, y la distancia de acoplamiento no supera el parámetro. Por este motivo es necesario realizar las operaciones de unión o desunión de microservicios; también a veces el algoritmo de similitud puede confundir el significado de alguna entidad y relacionarla donde no debe ir, para corregir este problema es necesario cambiar de microservicio a la historia de usuario.

La solución obtenida automáticamente tanto por al algoritmo genético y por el algoritmo de agrupamiento presenta una diferencia significativa con relación a DDD, después de realizar las operaciones de unión y desunión de microservicios la solución obtenida es muy cercana o mejor que DDD. Estas operaciones se realizan a través del componente agrupador del Microservice Backlog con intervención del usuario. La tabla 26 presenta el resultado de las métricas

Tabla 26. Análisis comparativo para la aplicación Foristom Conferencias

Método / métricas	Acop.	Cohesión		Granularidad		Rendimiento		Complejidad		T	Gm
	Cpt	Coht	Sst (%)	N	Wsict	Calls	Avg. Calls	Max. Pi	Cxt		
Genetic Algorithm F1: Cpt, Cxt, Wsict, Sst	4.90	3.33	84.6	13	4	6	0.46	29	127	59	137.1
Genetic Algorithm and Joins F1: Cpt, Cxt, Wsict, Sst	0.00	1.50	74.4	4	8	0	0.00	67	49.5	134	56.3
Genetic Algorithm F4: Cpt, Coht, Sst	2.45	2.67	79.5	9	6	4	0.44	52	167	108	170.2
Genetic Algorithm and Joins F4: Cpt, Coht, Sst	0.00	1.79	75.3	5	8	0	0.00	67	53.5	134	59.5
Grouping Algorithm Lemmatization	9.80	3.33	95.4	13	7	15	1.15	53	450	107	460.3
Grouping Algorithm and Joins - Lemmatization	4.90	1.79	73.1	5	10	11	2.20	74	651	146	653.5
Grouping Algorithm - Text	8.25	3.47	96.3	14	12	15	1.07	82	549	160	555.3
Grouping Algorithm and Joins – Text	4.90	1.79	73.9	5	13	7	1.40	90	466	176	470.0
DDD	3.16	1.50	75.7	4	9	6	1.50	83	426	167	428.0
29ms (Finer Granularity)	9.27	5.20	100.0	29	1	21	0.72	13	192	27	213.2
Monolith (Greater Granularity)	0.00	0.00	72.0	1	29	0	0.00	235	132	469	138.0

Estos resultados se pueden ver gráficamente en la figura 62. Se puede apreciar el número de microservicios (*N*), el acoplamiento *(CpT)*, la cohesión *(CohT)*, el máximo número de historias asociadas a un microservicio (*WsicT*), el total de llamadas entre los microservicios (*Calls*) y el promedio de llamadas entre los microservicios de la aplicación *(Avg. Calls)*.

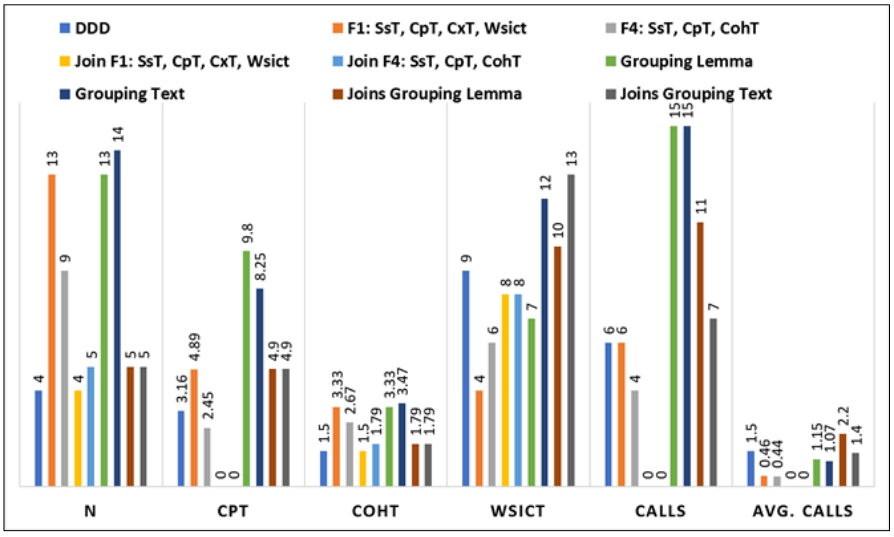


Figura 62. Análisis comparativo para la aplicación Foristom Conferencias.

Dónde MS: Número de microservicios, CPT: Acoplamiento de MSBA, COHT: Falta de grado de cohesión de MSBA, WSICT: Máximo WSIC de MSBA, WSIC es el número de historias de usuarios de MS. CALLS: número de llamadas entre microservicios. SS: Similitud semántica.

La figura 63 detalla la complejidad cognitiva obtenida por cada descomposición. La primera columna corresponde a DDD (en verde), seguido por el algoritmo genético (4 columnas siguientes en morado) y finaliza el algoritmo de agrupamiento (4 columnas siguientes en anaranjado).

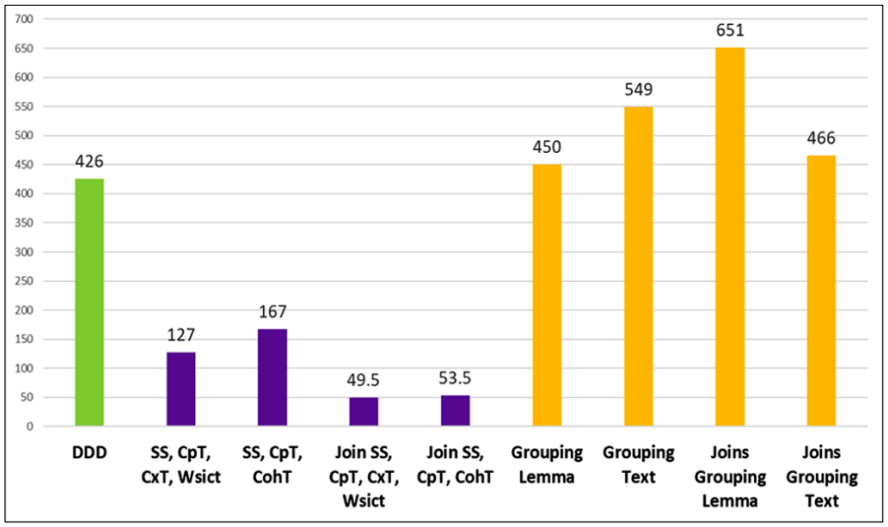


Figura 63. Análisis comparativo de los puntos de complejidad cognitiva para la aplicación Foristom Conferencias.

La complejidad cognitiva obtenida por el algoritmo genético fue considerablemente menor que la complejidad de la descomposición propuesta por DDD. Se repitió el mismo resultado que en los casos anteriores, por lo que se puede afirmar que nuestro modelo obtiene soluciones de menor complejidad, siendo así menos difícil de implementar y mantener. El algoritmo de agrupamiento tuvo valores cercanos a DDD pero siempre más grandes, por lo tanto la descomposición obtenida presenta mayor complejidad que la descomposición obtenida por DDD y por el algoritmo genético.

Las descomposiciones propuestas por nuestro modelo fueron semántica y funcionalmente coherentes; pudimos obtener microservicios completamente independientes, siendo ésta una característica importante para implementar, mantener y desplegar una aplicación basada en microservicios. La figura 64 muestra la similitud semántica obtenida en cada descomposición propuesta por los métodos.

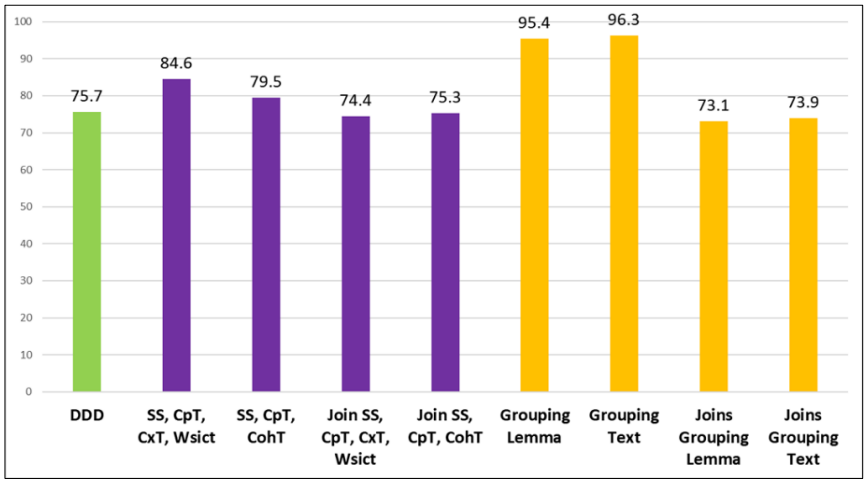


Figura 64. Análisis comparativo de la similitud semántica para la aplicación Foristom Conferencias.

Si las soluciones propuestas presentan una alta similitud semántica (mayor al 70%) sugieren que los microservicios agrupan las historias que se refieren a la misma entidad, por lo tanto, su cohesión es alta. En todas las soluciones obtenidas por el algoritmo genético, por el algoritmo de agrupamiento y por DDD, presentan una alta similitud semántica; siendo las propuestas por el algoritmo de agrupamiento las de mayor similitud. Se pudo observar que no es suficiente tener sólo alta similitud semántica para considerar una buena distribución de las historias de usuario en los microservicios, se deben analizar otros factores como el acoplamiento, las dependencias y comunicación entre los microservicios de la aplicación.

La figura 65 detalla los resultados de la métrica de granularidad *Gm* para las descomposiciones obtenidas por DDD, el algoritmo genético y el algoritmo de agrupamiento.

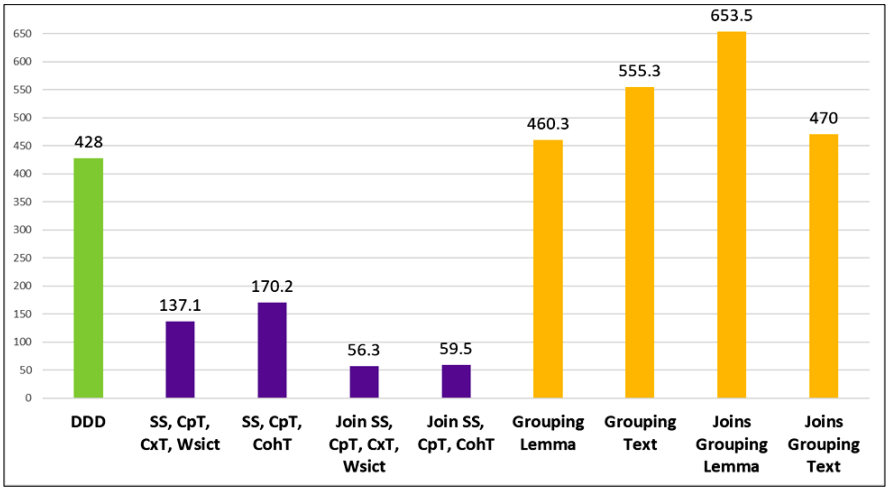


Figura 65. Análisis comparativo de la métrica de granularidad *Gm* la aplicación Foristom Conferencias.

Se puede apreciar que los resultados fueron considerablemente más bajos para el algoritmo genético, y más altos para el algoritmo de agrupamiento. El algoritmo de agrupamiento usando la lematización de las entidades de las historias de usuario tuvo un valor cercado a DDD, sus resultados son similares a DDD. En la figura 66 se presenta el análisis comparativo para las mejores soluciones (aquellas que tiene menor *Gm*) propuestas por el algoritmo genético y el algoritmo de agrupamiento comparados con DDD.

El algoritmo genético obtuvo mejores resultados en todas las métricas evaluadas. Obtuvo menor tiempo estimado de desarrollo, cero acoplamiento entre los microservicios, cero comunicación entre los microservicios, igual cohesión, menor complejidad y menor valor en GM comparado con DDD. Por lo tanto, el Microservice Backlog se puede considerar un método adecuado para definir y evaluar la granularidad de los microservicios. El algoritmo de agrupamiento obtuvo valores mayores en todas las métricas, aunque la diferencia no es muy grande.

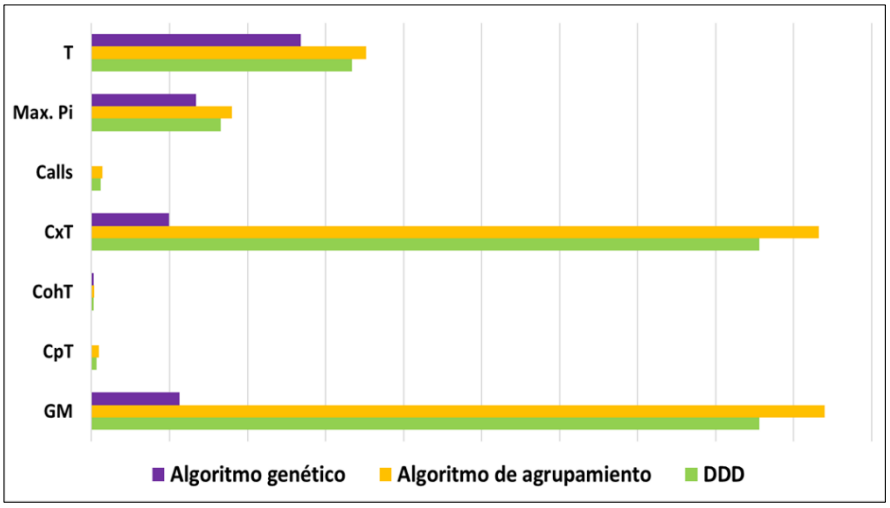


Figura 66. Análisis comparativo de los resultados obtenidos con el algoritmo genético, el algoritmo de agrupamiento y DDD para Foristom Conferencias.

5.4.4. Sinplafut

Sinplafut es un sistema de información que permite mejorar y agilizar la planificación del entrenamiento de fútbol utilizando técnicas modernas de entrenamiento deportivo. Sinplafut puede ser utilizado por equipos de fútbol, clubes, preparadores físicos, entrenadores y directores técnicos, tanto a nivel aficionado como profesional, como una aplicación de software como servicio - SaaS. Con Sinplafut se puede gestionar el perfil y la ficha de cada jugador, la información fisiológica y las lesiones. Permite crear y configurar los mesociclos, microciclos y sesiones de entrenamiento, utilizando métodos y pruebas preestablecidas, gestionadas y controladas por Sinplafut [116].

A partir de la descripción y definición del estudio de caso junto con su lógica de negocio, se identificaron las historias de usuarios y sus dependencias, para resumir, se agruparon en historias épicas. Ver tabla 27.

Sinplafut es un proyecto real, contiene 92 historias de usuarios con 302 puntos de historia estimados y 604 horas de tiempo de desarrollo estimado (secuencial). Debido a las limitaciones de espacio no se incluyeron todos los detalles de las historias de usuarios, las dependencias se

pueden apreciar en la tabla 28, estas fueron calculadas de acuerdo con la lógica del negocio de la aplicación y el flujo de datos.

Tabla 27. Product backlog de Sinplafut

Nombre de la épica	Historias de usuario	Puntos	Tiempo des.
Gestión del club deportivo	4	7	14
Gestión de equipo del club deportivo	6	16	32
Gestión de jugadores del equipo	11	33	66
Gestión del cuerpo técnico	6	9	18
Gestión del plan de entrenamiento	5	39	78
Gestión de mesociclos	7	15	30
Gestión de microciclos	6	13	26
Gestión de sesiones de entrenamiento	6	20	40
Gestión de métodos del entrenamiento deportivo	8	30	60
Generar cronograma de las sesiones de entrenamiento	1	8	16
Generar landing page de Sinplafut	1	10	20
Gestión de test deportivos	14	52	104
Gestión de usuario y seguridad	17	50	100
Total	92	302	604

Tabla 28. Dependencias entre las historias de usuario de Sinplafut

Id	Dependencia	Id.	Dependencia	Id.	Dependencia
HU1	{}	HU32	{}	HU63	{HU71}
HU2	{HU4}	HU33	{}	HU64	{HU88}
HU3	{HU4}	HU34	{HU78}	HU65	{HU89}
HU4	{HU71, HU8}	HU35	{}	HU66	{HU89}
HU5	{HU9}	HU36	{HU79, HU42, HU82, HU80}	HU67	{HU88}
HU6	{HU9, HU73}	HU37	{HU83, HU42, HU82, HU80}	HU68	{HU75, HU88}
HU7	{HU73}	HU38	{HU83}	HU69	{HU75, HU88, HU67}
HU8	{HU4, HU18, HU13}	HU39	{HU79}	HU70	{HU73, HU88, HU90}
HU9	{}	HU40	{}	HU71	{}
HU10	{HU14}	HU41	{HU84}	HU73	{}
HU11	{HU14, HU74}	HU42	{}	HU74	{}
HU12	{HU74}	HU43	{HU84}	HU75	{}
HU13	{HU73}	HU44	{}	HU76	{}
HU14	{}	HU45	{HU77}	HU77	{HU31}
HU15	{HU14}	HU46	{HU47}	HU78	{}
HU16	{HU75}	HU47	{HU48, HU58}	HU79	{}
HU17	{HU75}	HU48	{HU52}	HU80	{}
HU18	{HU73, HU23}	HU49	{}	HU81	{HU84}
HU19	{HU75}	HU50	{HU86}	HU82	{}
HU20	{HU75}	HU51	{HU86}	HU83	{}
HU21	{HU76}	HU52	{}	HU84	{}
HU22	{HU76}	HU53	{}	HU85	{HU84}
HU23	{HU75}	HU54	{HU87}	HU86	{}
HU24	{HU73, HU28, HU92, HU36}	HU55	{HU87}	HU87	{}
HU25	{HU77, HU29}	HU56	{}	HU88	{}
HU26	{HU77}	HU57	{HU86, HU56}	HU89	{}
HU27	{HU73}	HU58	{HU86}	HU90	{HU73, HU88}
HU28	{HU33, HU32, HU34}	HU59	{HU71, HU87}	HU92	{HU35, HU39}
HU29	{HU78, HU33, HU32, HU34}	HU60	{}	HU93	{HU79, HU35, HU34}
HU30	{HU78}	HU61	{HU88}	HU94	{HU79}
HU31	{HU77}	HU62	{HU88}		

Luego de definir las historias de usuario y sus dependencias se usa DDD siguiendo el procedimiento establecido por Evans para determinar los microservicios que van a hacer parte de la aplicación. Se deben identificar las entidades, los objetos de valor y los contextos delimitados para así poder identificar los microservicios que van a ser parte de la aplicación.

Entidades:

- Club deportivo
- Equipo
- Jugador
- Ficha del jugador

- Tipo de cuerpo técnico
- Cuerpo técnico
- Integrante cuerpo técnico
- Categoría
- Funcionalidad
- Lesión del jugador
- Mesociclo
- Tipo mesociclo
- Estado mesociclo
- Microciclo
- Estado microciclo
- Plan de entrenamiento
- Calendario de sesiones de entrenamiento
- Método de entrenamiento
- Archivo explicativo del método de entrenamiento
- Sesiones de entrenamiento
- Tipo de sesión de entrenamiento
- Test deportivo
- Tabla de valoración del test
- Valoración test Jugador
- Valoración test equipo
- Estadísticas test del equipo
- Perfil de usuario
- Permiso funcionalidad usuario
- Usuario
- Registro a Sinplafut

Agregados:

Se relacionan los nombres de los agregados y las entidades que hacen parte de cada uno de ellos:

1. Club deportivo
2. Equipo, categoría
3. Cuerpo técnico, integrante cuerpo técnico
4. Jugador: Ficha del jugador, lesión del jugador
5. Plan de entrenamiento, mesociclo, tipo de mesociclo, estado mesociclo, microciclo, estado microciclo, calendario de sesiones de entrenamiento
6. Sesiones de entrenamiento, Tipo de sesión de entrenamiento
7. Método de entrenamiento, Archivo explicativo del método de entrenamiento
8. Test deportivo, Tabla de valoración del test deportivo, Valoración test Jugador, Valoración test equipo, Estadísticas test del equipo.
9. Funcionalidad, perfiles de usuario, usuario, permiso funcionalidad usuario, registro a Sinplafut.

Contextos delimitados:

Se relacionan los nombres de los contextos delimitados, las entidades asociadas y las historias de usuario relacionadas) Se identificaron 9 contextos delimitados los cuales corresponden a los agregados identificados. Los contextos delimitados definen los microservicios que harán parte de la aplicación. El detalle de las entidades e historias de usuario asignadas a cada microservicio se puede apreciar en la tabla 29.

En total se identificaron 9 microservicios, los cuales fueron creados e ingresados al Microservices Backlog de forma manual y para poder ser comparados con las descomposiciones propuestas por el algoritmo genético y el algoritmo de agrupamiento. La figura 67 muestra las descomposiciones obtenidas para Sinplafut y las métricas calculadas.

En la figura 67 se puede apreciar en la parte inferior el grafo con las dependencias entre las historias de usuario, se aprecia que es complejo, tiene muchas relaciones y comunicación entre ellas. La tabla 30 presenta el detalle de las métricas calculadas para para las descomposiciones

obtenidas por el algoritmo genético, por el algoritmo de agrupamiento, por el equipo de desarrollo o arquitecto de software y por DDD.

Tabla 29. Microservicios identificados para Sinplafut usando DDD.

Microservicio	Entidades	Número de historias
1. Club deportivo	Club deportivo	4
2. Equipo	Equipo categoría	6
3. Jugador	Ficha del jugador Jugador Lesión de un jugador	11
4. Cuerpo técnico	Cuerpo técnico Integrante del cuerpo técnico	6
5. Plan de entrenamiento	Plan de entrenamiento del equipo. Mesociclo. Microciclo.	19
6. Sesión de entrenamiento	Sesión de entrenamiento Tipo de sesión de entrenamiento	7
7. Método de entrenamiento	Métodos de entrenamiento Archivos explicativos del método de entrenamiento	8
8. Test deportivo	Test deportivo Tabla de valoración del test. Valoración del test para el jugador. Tabla de valoración de test Estadísticas del test deportivo a nivel de equipo	14
9. Usuarios y permisos	Landing page Sesión de usuario Usuario Perfil de usuario Funcionalidad permiso funcionalidad usuario	17
	Total:	92

Tabla 30. Análisis comparativo de las descomposiciones obtenidas para Sinplafut.

Método / métricas	**Acop.**	**Cohesión**		**Granularidad**		**Rendimiento**		**Complejidad**		**T**	**Gm**
	Cpt	**Coht**	**Sst**	**N**	**Wsict**	**Calls**	**Avg. Calls**	**Max. Pi**	**Cxt**		
Genetic Algorithm F6: Cpt, Coht, Wsict, Sst	8.60	5.00	84.8	27	7	49	1.81	32	857.5	64	862.0
Genetic Algorithm and Joins F6: Cpt, Coht, Wsict, Sst	7.35	3.33	86.6	13	13	24	1.84	49	788.5	98	792.1
Genetic Algorithm F4: Cpt, Coht, Sst	8.94	5.20	86.1	29	6	55	1.90	35	881.0	70	885.7
Genetic Algorithm and Joins F4: Cpt, Coht, Sst	6.32	3.02	83.0	11	18	24	2.18	57	838.0	114	840.8
Grouping Algorithm Lemmatization	17.32	5.48	95.1	32	9	67	2.09	41	1123.5	82	1136.8
Grouping Algorithm Text	16.31	5.39	94.7	31	7	67	2.16	41	1097.5	82	1109.6
Grouping Algorithm Lemmatization – Joins	9.59	3.02	86.8	11	16	24	2.18	58	814.0	116	819.9
Grouping Algorithm Text - Joins	8.72	3.18	86.8	12	14	28	2.33	52	889.0	104	893.5
Architec - Development Team	3.74	1.79	82.8	5	34	9	1.80	127	721.0	254	723.0
DDD	10.58	2.31	84.4	9	19	23	2.55	75	920.5	150	926.9
92ms (Finer Granularity)	24.00	9.49	100.0	92	1	98	1.07	15	480.5	30	537.2
Monolith (Greater Granularity)	0	0	61.0	1	92	0	0.00	304	198.0	608	221.8

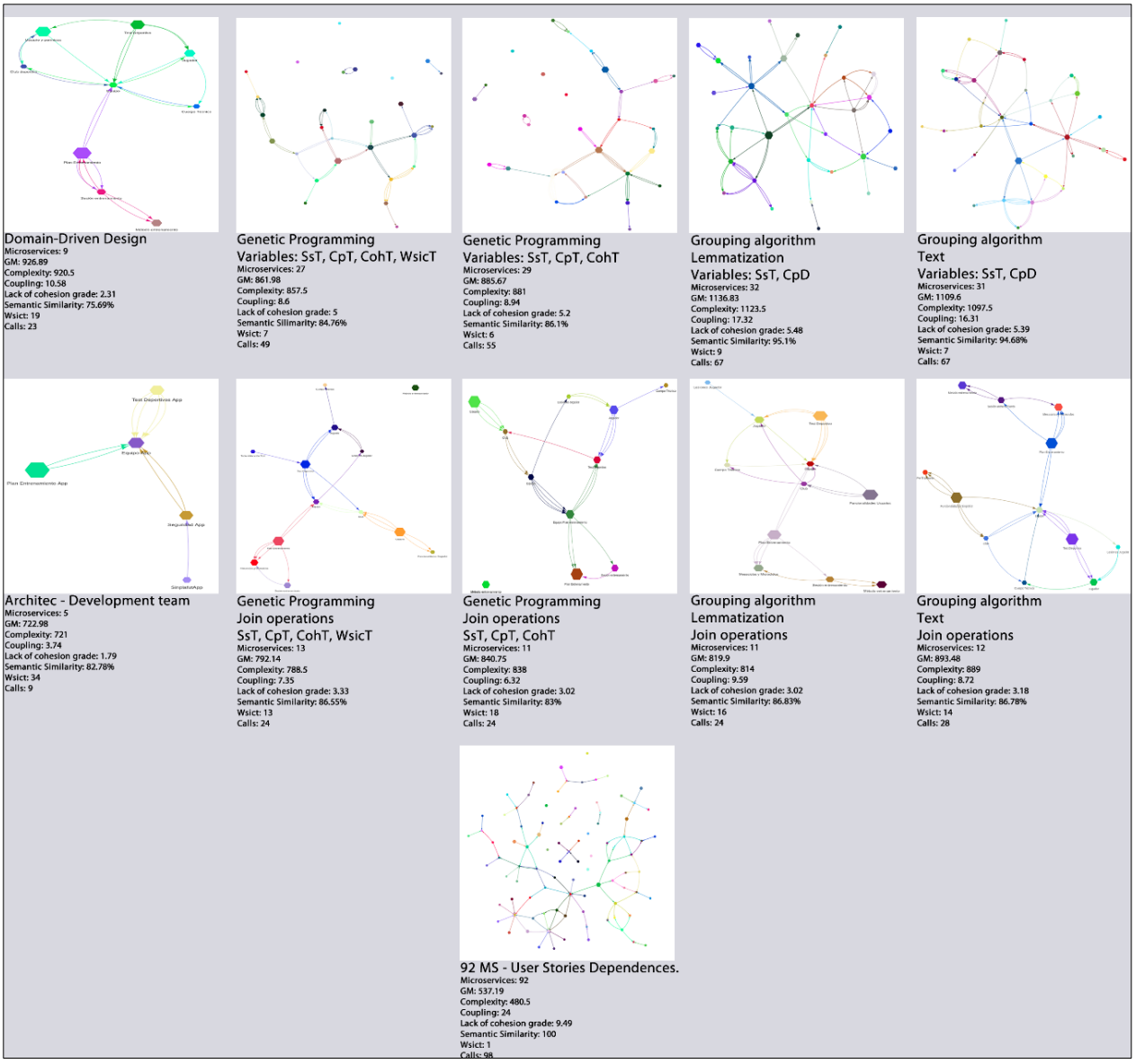


Figura 67. Microservices Backlog comparado con DDD y con la propuesta del equipo de Desarrollo para la aplicación Sinplafut.

La descomposición propuesta por el equipo de desarrollo resulto ser la de menor valor en la métrica *Gm*, aunque la solución consta de 5 microservicios grandes y complejos; presenta mayor *WsicT*, mayor tiempo estimado de desarrollo y mayor puntos de historias asociados a un microservicio; esta solución se hizo bajo la experiencia del equipo de desarrollo de Sinplafut, sin considerar algún método o técnica para definir la granularidad de los microservicios. La figura 68 muestra una gráfica comparativa de las métricas evaluadas en las descomposiciones obtenidas para Sinplafut.

El número de microservicios obtenidos automáticamente por el modelo es significativamente más grande que lo obtenido por DDD y la solución propuesta por el arquitecto. El algoritmo genético obtiene menor acoplamiento que DDD pero mayor que la solución propuesta por el arquitecto del proyecto; mientras que el algoritmo de agrupamiento sólo obtiene menor acoplamiento que DDD después de realizar las operaciones de unión y desunión de microservicios.

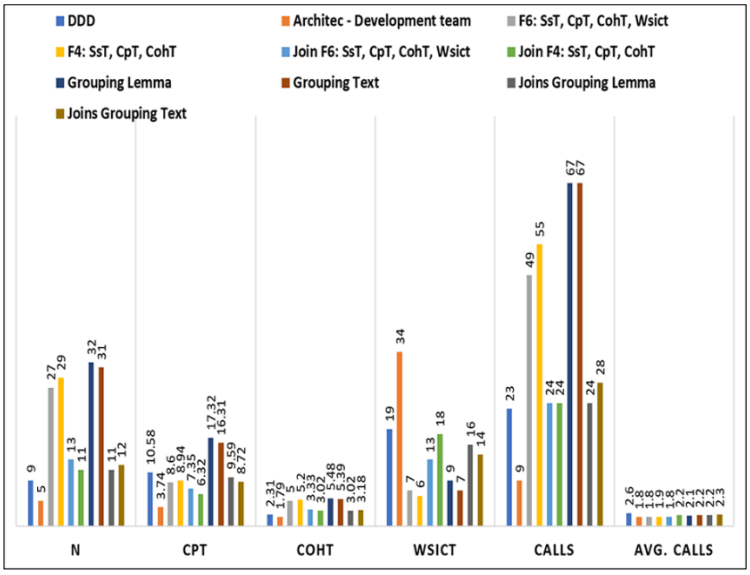


Figura 68. Análisis comparativo de las métricas calculadas para la aplicación Sinplafut.

El menor acoplamiento fue para la solución propuesta por el arquitecto. La falta de cohesión fue muy similar para casi todas las descomposiciones propuestas, para el algoritmo genético y el algoritmo de agrupamiento siempre fueron mayores que DDD y que la solución propuesta por el arquitecto. En cuanto al mayor número de historias asociadas a un microservicio *(WsicT)* los valores más grandes corresponden a DDD y a la propuesta del arquitecto; el menor valor corresponde al algoritmo genético, pero se incrementa al realizar las operaciones de unión y desunión de microservicios.

El algoritmo de agrupamiento obtiene valores muy similares al algoritmo genético. El número de llamadas entre microservicios es significativamente mayor para las descomposiciones obtenidas por el algoritmo genético y de agrupamiento automáticamente, al realizar las operaciones de unión y desunión se obtienen valores muy similares a DDD y a la propuesta del arquitecto.

La figura 69 presenta el análisis comparativo de los puntos de complejidad cognitiva calculados para las descomposiciones propuestas. El algoritmo genético obtuvo menor complejidad que DDD y que el algoritmo de agrupamiento y un valor mayor pero muy cercano a la propuesta por el arquitecto, siendo esta la de menor complejidad. La mayor complejidad la obtiene el algoritmo de agrupamiento, aunque al realizar las operaciones de unión y desunión se obtienen valores menores a DDD y que a algunas descomposiciones obtenidas con al algoritmo genético.

La figura 70 presenta el grafico que compara la similitud semántica obtenida por las soluciones propuestas. Al tener una alta similitud semántica indica que la aplicación tiene una alta cohesión y las historias que se agrupan en un microservicio se refieren al mismo tema (misma entidad). Las soluciones propuestas todas tienen un alto acoplamiento semántico superior al 80%, los valores son muy similares, siendo el de mayor similitud semántica el algoritmo de agrupamiento.

La figura 71 presenta los resultados obtenidos de las descomposiciones propuestas en cuanto a la métrica de granularidad *Gm*; se mantiene el mismo comportamiento encontrado en los casos de estudio anteriores, siendo el algoritmo genético el de menor valor comparado con DDD y el algoritmo de agrupamiento. La solución propuesta por el arquitecto o equipo de desarrollo fue la de menor *Gm.*

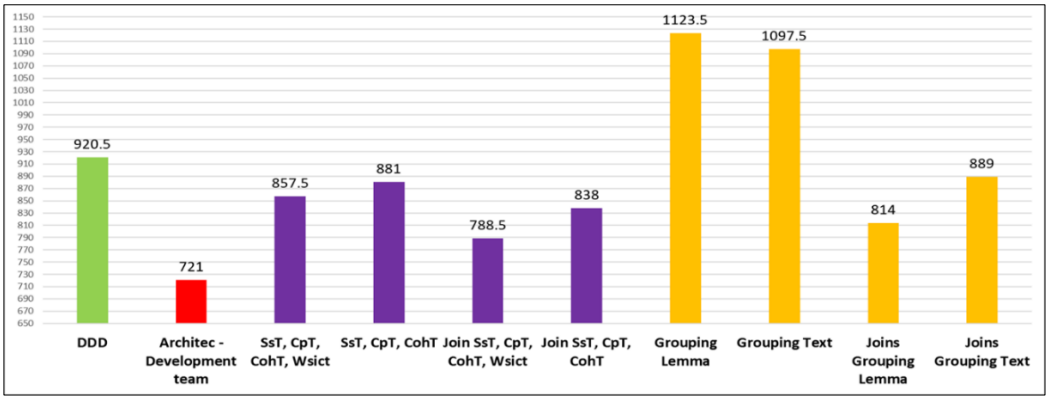


Figura 69. Análisis comparativo de la complejidad cognitiva para la aplicación Sinplafut.

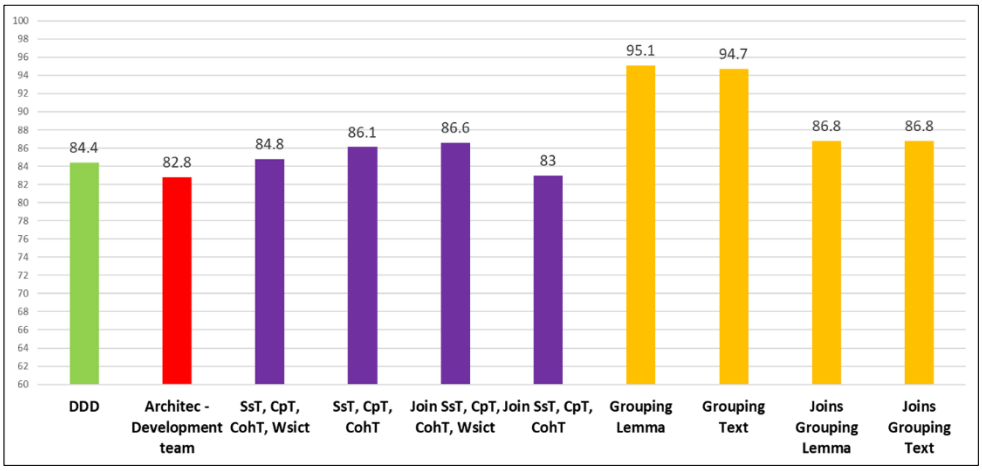


Figura 70. Análisis comparativo de la similitud semántica para la aplicación Sinplafut.

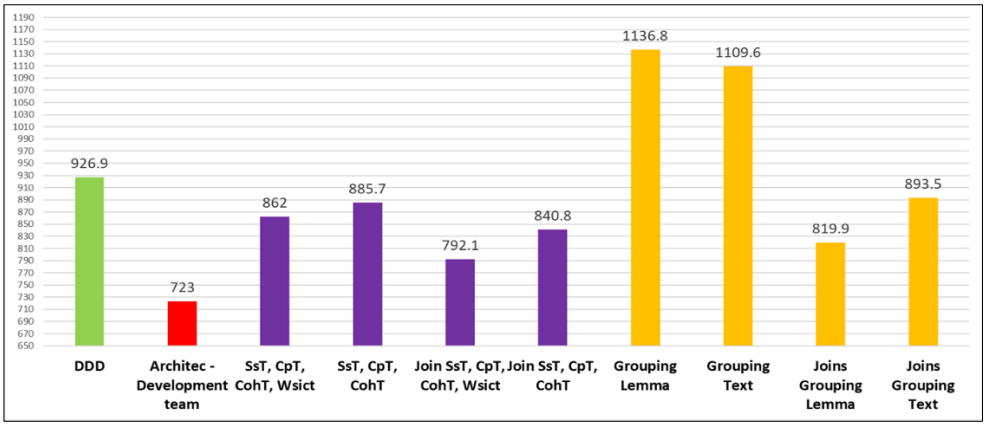


Figura 71. Análisis comparativo de la métrica de granularidad *Gm* para la aplicación Sinplafut.

En la figura 72 se presenta el análisis comparativo final de las métricas evaluadas (acoplamiento, cohesión, rendimiento, complejidad, mayor puntos de historia de los microservios y tiempo estimado de desarrollo) para las descomposiciones que tuvieron un menor *Gm* tanto en el algoritmo genético como en el algoritmo de agrupamiento; comparados con DDD. Se mantiene la misma tendencia que en los casos de estudio anteriores, el algoritmo genético obtiene menor tiempo estimado de desarrollo, casi igual número de llamadas entre microservicios, menor

complejidad, similar cohesión, menor acoplamiento y menor valor en *Gm*; el algoritmo de agrupamiento obtuvo casi el mismo comportamiento.

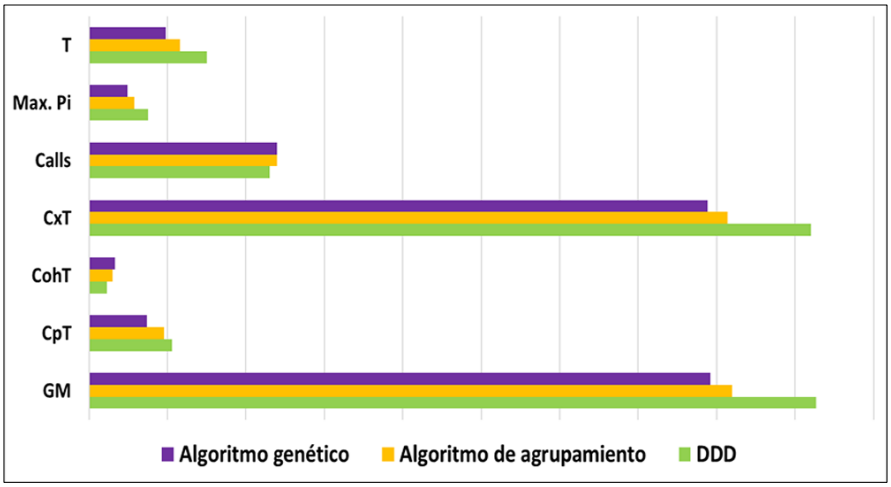


Figura 72. Análisis comparativo de los métodos para la aplicación Sinplafut.

Las operaciones de unión y desunión fueron fundamentales para obtener resultados mejores que DDD, automáticamente en algunos casos se obtienen resultados similares o mejores que DDD, pero siempre se debe revisar que las historias de usuario queden asociadas en el lugar que les corresponde, por ejemplo, el algoritmo de similitud semántica asume que la sesión de entrenamiento es muy similar semánticamente a la sesión del usuario, siendo dos cosas diferentes, por este motivo es muy importante la intervención del usuario para analizar y evaluar lo que se obtiene automáticamente, para así proponer mejoras y obtener mejores resultados.

En este caso de estudio se pudo demostrar que el Microservices Backlog obtiene descomposiciones de historias de usuario a microservicios con bajo acoplamiento, alta cohesión (desde el punto de vista semántico), baja complejidad, baja comunicación entre los microservicios y menor tiempo de desarrollo estimado; por tanto, el Microservices Backlog es una opción viable para el diseño y evaluación de aplicaciones basadas en microservicios.

Una diferencia importante entre el algoritmo genético y el algoritmo de agrupamiento en este caso de estudio fue el tiempo de ejecución, la solución al problema es compleja (“NP-Hard”) al aumentar las historias de usuario se aumenta considerablemente el tiempo de ejecución. El algoritmo genético en promedio duró aproximadamente 11 horas y media en Sinplafut, en los otros casos de estudio el promedio de ejecución fue de 9.7 minutos; mientras que el algoritmo de agrupamiento duro aproximadamente 1.5 minutos en Sinplafut y casi inmediato en los otros casos de estudio. Se pudo identificar que el cálculo de las métricas y el análisis semántico son los que representan el mayor costo computacional del Microservices Backlog. El problema del costo computacional será abordado como trabajo futuro usando computación en paralelo, la cual fue usada y probada para generar la población del algoritmo genético mejorando el rendimiento, se pretende paralelizar el cálculo de las métricas y el cálculo de la similitud semántica.

En resumen, el análisis de los resultados obtenidos se presenta en la tabla 31, comparando los mejores resultados obtenidos con cada método.

Al realizar el análisis de los resultados, se rechaza la hipótesis nula planteada: La solución propuesta por el Microservices backlog presentó bajo acoplamiento, alta cohesión, baja complejidad, menor comunicación y menos dependencias comparado con la solución propuesta

por métodos del estado del arte y por DDD; adicionalmente la solución propuesta es coherente (alta similitud semántica, *SsT* mayor al 70%) desde el punto de vista semántico.

Tabla 31. Resultados obtenidos en el proceso de evaluación del Microservices Backlog

Caso de estudio	Método	Métricas										
		N	GM	CpT	CohT	SsT	WsicT	CxT	T	Max Pi	Calls	Avg. Calls
Cargo-Tracking	Algoritmo genético	3	85.8	3.16	1.16	70.92	6	74.0	35	23	3	1.0
	Algoritmo de agrupamiento	4	185.2	4.69	1.50	88.42	9	178.5	54	35	8	2.0
	DDD	4	156.6	5.29	1.50	74.09	6	145.0	39	27	9	2.3
	Service Cutter	3	206.8	3.16	1.15	74.42	10	202.5	61	41	8	2.7
	MITIA	4	203.1	6.78	1.06	76.77	5	190.0	30	19	12	3.0
Jpet-Store	Algoritmo genético	5	104.7	1.41	1.79	86.50	9	102.5	54	35	3	0.6
	Algoritmo de agrupamiento	5	143.6	2.83	1.79	86.60	6	140.5	32	20	7	1.4
	DDD	4	203.7	3.46	1.50	85.30	8	200.0	36	22	9	2.3
	Execution Traces	4	179.7	3.46	1.50	84.06	7	175.5	31	19	8	2.0
Foristom Conferencias	Algoritmo genético	4	56.3	0.00	1.50	74.40	8	49.5	134	67	0	0.0
	Algoritmo de agrupamiento	5	470.0	4.90	1.79	73.88	13	466.5	176	90	7	1.4
	DDD	4	428.0	3.16	1.50	75.69	9	426.0	167	83	6	1.5
Sinplafut	Algoritmo genético	13	792.1	7.35	3.33	86.55	13	788.5	98	49	24	1.8
	Algoritmo de agrupamiento	11	819.9	9.59	3.02	86.83	16	814.0	116	58	24	2.2
	DDD	9	926.9	10.58	2.31	84.4	19	920.5	150	75	23	2.6
	Arquitecto o equipo de desarrollo	5	723.0	3.74	1.79	82.78	34	721.0	254	127	9	1.8

6. CONCLUSIONES

6.1. CONCLUSIONES Y CONTRIBUCIONES

En esta tesis doctoral se propuso un modelo denominado Microservices Backlog que permite a los arquitectos, diseñadores o desarrolladores razonar sobre la granularidad de los microservicios en tiempo de diseño, pueden analizar métricas, diagramas y dependencias de los microservicios; pueden notar puntos críticos, tiempo estimado de desarrollo de la aplicación; pueden probar diferentes soluciones o descomposiciones, analizarlas y seleccionar las mejores para ser implementadas.

El Microservices Backlog obtuvo un bajo acoplamiento, una alta cohesión, baja comunicación entre los microservicios y una baja complejidad cognitiva en las descomposiciones obtenidas automática o semiautomáticamente, comparado con los métodos del estado del arte. Usando nuestro modelo, el arquitecto de software o el equipo de desarrollo puede obtener aplicaciones basadas en microservicios con bajo acoplamiento, baja complejidad y menos llamadas entre microservicios, siendo este un aporte fundamental al diseño y desarrollo de aplicaciones basadas en microservicios.

La distribución de las historias de usuario en microservicios afecta el acoplamiento, la cohesión, la complejidad impactando al rendimiento, la modularidad y la mantenibilidad de la aplicación. Asociar una historia de usuario a un solo microservicio (granularidad más fina), siguiendo el principio de responsabilidad simple, implica más acoplamiento para la aplicación, de igual forma tener una aplicación monolítica no es la mejor opción en cuanto a mantenibilidad, escalabilidad, pruebas y despliegue de la aplicación. La solución óptima se encuentra en un punto intermedio entre estas dos, y depende de los requerimientos no funcionales y características propias de la aplicación, del equipo de desarrollo y de los requerimientos no funcionales que se deban abordar.

Con el Microservices Backlog el equipo de desarrollo puede evaluar diferentes formas de distribuir las historias de usuario en los microservicios y tomar decisiones basados en métricas, gráficos y análisis comparativos, esto en tiempo de diseño. Por lo tanto, con nuestro modelo podemos razonar acerca de la granularidad de los microservicios en tiempo de diseño, cubriendo así uno de los vacíos de investigación propuesto en la revisión de literatura.

Al comparar nuestra propuesta el Microservices Backlog con los trabajos relacionados y con los resultados de la revisión sistemática de literatura, ninguno de los trabajos identificados usó las historias de usuario como datos de entrada, ninguno usó datos de las prácticas ágiles o del desarrollo ágil de software. Sólo un trabajo usó atributos de calidad con características en tiempo de ejecución y al mismo tiempo características del software como artefacto (P16), de la misma forma el Microservices Backlog propuesto usa características en tiempo de ejecución y características del software como artefacto. Ninguno de los trabajos identificados usó el rendimiento, la funcionalidad, la mantenibilidad y la modularidad al mismo tiempo para evaluar la granularidad de los microservicios.

Sólo un trabajo (P24) utilizó métricas de acoplamiento, cohesión y complejidad al mismo tiempo, de la misma forma el Microservice Backlog usa métricas de acoplamiento, cohesión y complejidad para evaluar los microservicios candidatos de la aplicación. Por lo tanto, con este trabajo de investigación doctoral se cubren vacíos de investigación propuestos en el estado del arte y representa una innovación en el desarrollo de aplicaciones basadas en microservicios.

Las contribuciones más importantes de este trabajo de investigación doctoral corresponden a:

1. Se propuso una formulación matemática del modelo de granularidad de aplicaciones basadas en microservicios.
2. Se implementó el Microservices Backlog como un sistema de apoyo a la toma de decisiones para arquitectos y equipos de desarrollo de software, donde ellos pueden evaluar la granularidad de las aplicaciones basadas en microservicios.
3. El arquitecto o equipo de desarrollo puede con el Microservices Backlog razonar acerca de la granularidad de los microservicios en tiempo de diseño.
4. Se propusieron dos métodos de agrupamiento de las historias de usuario en microservicios, el primero un algoritmo genético y el segundo un algoritmo de aprendizaje automático no supervisado basado en un agrupamiento jerárquico sin determinar a priori el número de grupos a obtener.
5. En el sistema Microservices Backlog se plantea la base para implementar otros métodos de agrupamiento (por ejemplo, colonia de hormigas, algoritmos genético multiobjetivo, algoritmo genético NSGAII, método multicriterio, entre otros) como trabajo futuro y evaluarlos contra los propuestos y en los casos de estudio utilizados en esta tesis doctoral.
6. Se identificaron, adaptaron y propusieron métricas de acoplamiento, cohesión, complejidad (complejidad por puntos cognitivos métrica nueva propuesta en esta tesis), comunicación entre los microservicios, granularidad y tiempo estimado de desarrollo.
7. Se implemento un algoritmo que permite calcular las métricas y hacer análisis comparativos.
8. Se propone el diagrama de dependencias Microservices Backlog como un grafo bidireccional, donde se puede apreciar las dependencias de los microservicios, la complejidad de la aplicación, el acoplamiento, la cohesión, la granularidad y los detalles de cada microservicio. Todo esto a partir de la información contenida en las historias de usuario y sus dependencias.
9. Se caracterizó el proceso de desarrollo de aplicaciones basadas en microservicios y se implementó un prototipo de una aplicación siguiendo este proceso de desarrollo.

Por último, se abre una línea de investigación interesante para la Universidad del Valle y la Universidad Francisco de Paula Santander, línea de investigación vigente y de mucho interés a nivel nacional e internacional, a nivel académico e industrial, se plantean las bases para realizar futuras investigaciones para seguir haciendo aportes al desarrollo de aplicaciones basadas en microservicios.

6.2. LIMITACIONES Y TRABAJO FUTURO

Para determinar las historias de los usuarios de los estudios de casos, utilizamos la información reportada en los artículos publicados, estudiamos su lógica del negocio, e hicimos nuestro mejor esfuerzo para no sesgar esta definición. La definición y descripción de las historias de usuarios fue revisada por el director de tesis y asesores de forma independiente, las contradicciones se resolvieron de común acuerdo entre todos.

La definición de las dependencias entre las historias de usuarios es un punto crítico de nuestro modelo. Se definieron a partir de la información contenida en la historia de usuario, de la lógica de negocio de la aplicación, del código fuente, del flujo de datos y de las dependencias en el modelo de datos. Para aplicaciones más grandes que tienen muchas historias de usuario puede ser una tarea compleja determinar estas dependencias. Como trabajo futuro se pretende crear un método automático o semiautomático que permita definir las dependencias entre las historias

de usuario teniendo en cuenta, el flujo de datos, la lógica del negocio, o un análisis transaccional, también a partir del modelo de datos. Este es un tema interesante de investigación.

El problema de la asignación de historias de usuario a los microservicios tiene una complejidad alta de resolver (tipo NP-Hard) cuando el número de historias de usuario aumenta el tiempo de ejecución del algoritmo genético aumenta considerablemente. El tiempo medio de ejecución en las pruebas realizadas (sin considerar Sinplafut) no superó los 10 minutos, utilizando un ordenador core-i7, con 16 gigabytes de Ram (una población de 1000 individuos, convergencia 10%, número máximo de iteraciones 400 con 500 hijos y 500 mutaciones en cada iteración). Como trabajo futuro tenemos la intención de implementar el algoritmo genético usando programación paralela para mejorar el tiempo de ejecución. En el caso de estudio Sinplafut con 92 historias de usuario el tiempo de ejecución fue muy alto (aproximadamente 12 horas), pero fue reducido considerablemente a unos cuantos minutos al implementar el algoritmo de agrupamiento.

El algoritmo genético no es determinista, en cada ejecución puede dar resultados diferentes, para reducir este problema, ejecutamos el algoritmo varias veces, seleccionamos para cada caso la mejor solución y verificamos que se repitiera la mayoría de las veces.

Se implementó un algoritmo que calcula las métricas de evaluación; se utilizó el mismo algoritmo para todas las descomposiciones utilizadas para la comparación.

La métrica “Lack of cohesion” se calcula a partir de la interdependencia de los microservicios, depende mucho del número de microservicios, como trabajo futuro se quiere revisar y proponer otra métrica de cohesión, incluirla en el modelo y evaluar los resultados. Se concluyó que el Microservices Backlog propone soluciones de alta cohesión porque su similitud semántica es alta, entonces los microservicios se refieren al mismo tópico, tema o entidad; la cohesión semántica fue propuesta y trabajada por otros autores [70], [107].

También como trabajo futuro generaremos código fuente o plantillas para la solución seleccionada, así como estimaremos los recursos computacionales y las opciones de despliegue. Así como incorporar DevOps (integración continua, despliegue continuo), pruebas automatizadas, tolerancia a fallas, entre otras cosas que se pueden hacer a partir de este trabajo de investigación doctoral.

6.3. AMENAZAS A LA VALIDEZ

6.3.1. Validez interna

El planteamiento de las métricas se basó principalmente en la revisión del estado del arte, estas métricas han sido probadas y usadas en otros proyectos; para poder ser usadas en el Microservices Backlog propuesto se hicieron unas adaptaciones en su planteamiento y formulación matemática sin afectar su fundamento teórico base. Estas métricas se calculan sobre el objeto de estudio, sin intervención humana por medio del algoritmo calculador de métricas, evitando así variaciones en cálculos sucesivos debido a factores humanos; usando el algoritmo se garantiza su replicación obteniendo el mismo resultado al ser calculadas una y otra vez sobre el mismo objeto de estudio. Los cálculos de las métricas fueron revisados contra los cálculos realizados manualmente, siendo siempre los mismos y garantizando que son correctos, y miden lo que se quiere medir.

Los objetos de estudio en los análisis estático y dinámico realizados en la validación de los resultados de esta tesis corresponden a las descomposiciones (distribución de las historias de usuario en microservicios) obtenidas por los métodos evaluados para los 4 casos de estudio, para obtener la descomposición de los métodos del estado del arte se usó la descripción y detalles publicados por los autores en los trabajos relacionados, a partir de esa información se definieron los microservicios y las respectivas historias de usuario, este punto puede ser una amenaza a la validez, para reducir el sesgo los asesores de la tesis revisaron de forma independiente el planteamiento, discutiendo las diferencia y llegando a las soluciones o descomposiciones presentadas en la tesis.

Para reducir el sesgo usamos el algoritmo que calcula las métricas, el cual es diferente al algoritmo genético, este mismo algoritmo es usado para calcular las métricas en los microservicios candidatos obtenidos por los métodos del estado del arte (DDD, Service Cutter, MITIA, Execution Traces, Solución propuesta por el arquitecto). Las mismas historias de usuario, las mismas dependencias y la misma información fue usada por todos los casos evaluados. La definición de las historias de usuario fue revisada por dos expertos (asesores de la tesis) y por el director de la tesis para garantizar que son correctas y completas para los casos de estudio usados.

El algoritmo genético usa una expresión matemática diferente a la métrica de granularidad *Gm* usada para la validación y evaluación, de la misma forma se usaron otras métricas diferentes a las usadas en la función objetivo (tiempo estimado de desarrollo, puntos estimados, llamadas entre microservicios, el promedio de llamadas, y el número de microservicios) reduciendo el sesgo y siendo una evaluación correcta y completa.

6.3.2. Validez externa

Se presenta una amenaza a la validez externa debido a la generalización de los resultados y que se garantice que en otros casos los resultados obtenidos en nuestra evaluación sean los mismos. Para poder generalizar los resultados se requiere una evaluación estadística más completa, con más casos y objetos de estudio, buscamos datos (datasets) de proyectos ágiles que tengan definidas las historias de usuario o requerimientos, el “producto backlog”, la planificación de las iteraciones, y los microservicios implementados, no se encontraron proyectos con estas características. Lo mas cercano fue el dataset propuesto por Rahman y otros en [91] que contiene 20 proyectos implementados con microservicios, para usar estos datos se requiere un proceso de extracción y definición de las historias de usuario y su respectiva asociación a los microservicios implementados, esta es una tarea tediosa de realizar y estuvo por fuera del alcance de esta tesis doctoral. Esta evaluación será abordada como trabajo futuro.

Para reducir esta amenaza usamos 4 proyectos de estudio, 2 ejemplos típicos del estado del arte, que ha sido usados por otros autores, y dos proyectos de la vida real, usamos e ilustramos el Microservices Backlog en esos 4 casos de estudios diferentes, en cada uno se aumenta su complejidad y tamaño. Los resultados fueron coherentes en los 4 proyectos, se obtuvieron resultados muy similares que permiten concluir que el Microseravices Backlog representa un aporte importante al desarrollo de aplicaciones basadas en microservicios.

6.4. PRODUCTOS OBTENIDOS

Como parte del proceso de evaluación y validación del Microservices Backlog se participó en conferencias regionales, nacionales e internacionales. A medida que se iban cumpliendo los objetivos, teniendo resultados y productos presentables se fueron enviando a conferencias reconocidas y del área de la ingeniería del software, de la computación basada en servicios y en arquitectura de software. La evaluación dada por los pares evaluadores y los comentarios recibidos en las presentaciones ayudaron a mejorar y validar la aceptación del Microservices Backlog. La tabla 28 detalla los productos realizados.

En resumen, se obtuvieron los siguientes productos:
Conferencias internacionales: 6
Conferencias nacionales o regionales: 2
Publicaciones internacionales Scopus Q1: 2 (En evaluación o revisión)
Publicaciones internacionales Scopus Q2: 3 (1 publicado 2 en evaluación)
Publicaciones internacionales Scopus Q3: 3 publicados.
Publicaciones Publindex Categoria A1: 2 (En evaluación o revisión)
Publicaciones Publindex Categoria A2: 3 (1 publicado 2 en evaluación o revisión)
Publicaciones Publindex Categoria B: 4 publicados.

Tabla 32. Publicaciones resultado del proceso de investigación

Producto	Publicado en:	Año	Cuartil scopus / publindex	Citaciones
Plataforma como servicio – PaaS para la gestión de tecnologías en el desarrollo y despliegue de aplicaciones web. In: congreso internacional de ciencias de la computación y sistemas de información – ciccsi 2017. Mendoza, argentina,	Libro de resúmenes conferencia internacional.	2017	No	0
Método de automatización del despliegue continuo en la nube para la implementación de microservicios. In: Avances en Ingenieria de software a nivel iberoamericano, CIBSE 2018. Bogota: ibero-american conference on software engineering, 597–604.	Libro de resúmenes conferencia internacional. Scopus.	2018	No	3
A development process of enterprise applications with microservices. Encuentro internacional de ciencias aplicadas e ingeniería (universidad - industria) – eisi 2018. Journal of physics: conference series 1126:012017. Doi: 10.1088/1742-6596/1126/1/012017.	Revista internacional. Scopus.	2018	Q3 (en 2018) / Categoria B	5
Modelo ágil de especificación de la arquitectura de aplicaciones basadas en microservicios. 13 Congreso colombiano de computación. 13CCC, 2018.	Libro de resúmenes. Conferencia nacional	2018	No	0
Sinplafut: a microservices – based application for soccer training. In: 5th international week of science, technology & innovation. Journal of physics: conference series. 012026. Doi: 10.1088/1742-6596/1388/1/012026.	Revista internacional. Scopus.	2019	Q3 (en 2019) / categoría B	1
Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación. I congreso de ciencia de la computacion, electronica e industrial. Ambato, ecuador. Revista ibérica de sistemas e tecnologias de informação e23:107–120.	Revista internacional. Scopus	2019	Q3 (en 2019) / categoría B	0
Generación automática de la planificación de la entrega en desarrollo de software ágil, asignación de historias de usuario a los desarrolladores usando algoritmos genéticos. Aibi revista de investigación, administración e ingeniería:29–38. Doi: 10.15649/2346030x.735.	Revista nacional. Publindex	2020	No clasificada/ categoria B	0
Microservices backlog - a model of granularity specification and microservice identification. In: international conference on service computing scc 2020. Lecture notes in computer science. Springer science and business media deutschland, 85–102. Doi: 10.1007/978-3-030-59592-0_6.	Libro de resúmenes. Lecture notes in computer science. Springer. Scopus.	2020	Q2 / No homologado	1
Microservice backlog - modelo inteligente para la definición y evaluación de la granularidad de los microservicios de una aplicación. Semana de ingeniería universidad del valle.	Sin publicación.	2020	No	0

Defining and measuring microservice granularity – a literature overview. Journal: Peerj computer science.	Enviado a evaluación. Revista internacional. Scopus	2020	Q1 / Categoria A1	0
Semantic grouping algorithm to identify microservices from user stories in agile application development. Journal: IEEE transactions Latin America.	En revisión para enviar a evaluación. Revista internacional. Scopus	2021*	Q2 / Categoria A2	0
Cognitive complexity points: a metric for evaluating complexity of microservices decompositions and microservices-based applications. Journal: IEEE transactions latin america.	En revisión para enviar a evaluación. Revista internacional. Scopus	2021*	Q2 / Categoria A2	0
Microservices backlog – a model for identification and evaluation of microservices from user stories in agile software development. Journal: IEEE access	Enviado a evaluación. Revista internacional. Scopus	2021*	Q1 / Categoria A1	0
Total	11 productos			10 citaciones

* Año probable de publicación.

7. BIBLIOGRAFIA

[1] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[2] V. Farcic, *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices*. leanpub.com., 2016.

[3] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, 2015, doi: 10.1109/MS.2015.11.

[4] T. Salah, M. Jamal Zemerly, Chan Yeob Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, Dec. 2016, pp. 318–325, doi: 10.1109/ICITST.2016.7856721.

[5] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in Practice, Part 1: Reality Check and Service Design," *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, Jan. 2017, doi: 10.1109/MS.2017.24.

[6] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice Transition and its Granularity Problem: A Systematic Mapping Study," *Softw. Pract. Exp.*, no. February, pp. 1–31, 2020, doi: 10.1002/spe.2869.

[7] N. Kulkarni and V. Dwivedi, "The role of service granularity in a successful soa realization - A case study," in *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*, 2008, vol. PART 1, pp. 423–430, doi: 10.1109/SERVICES-1.2008.86.

[8] A. Homay, M. de Sousa, A. Zoitl, and M. Wollschlaeger, "Service Granularity in Industrial Automation and Control Systems," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2020, pp. 132–139, doi: 10.1109/ETFA46521.2020.9212048.

[9] R. S. Pressman, *Ingeniería del Software un enfoque práctico.*, Septima. Mexico: McGraw Hill Interamericana editores, 2010.

[10] I. Sommerville, *Ingeniería del Software*, Septima. Madrid: Person Education S.A., 2005.

[11] A. Abran and J. W. Moore, *Swebok - Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2004.

[12] K. Beck *et al.*, "Manifiesto por el Desarrollo Ágil de Software," 2001. http://agilemanifesto.org/iso/es/manifesto.html (accessed Apr. 29, 2017).

[13] Versionone Enterprise, I. Digital.ai Software, and I. CollabNet, "14th annual state of agile report," 2019. [Online]. Available: www.stateofagile.com.

[14] T. Sedano, P. Ralph, and C. Peraire, "The Product Backlog," in *Proceedings - International Conference on Software Engineering*, May 2019, vol. 2019-May, pp. 200–211, doi: 10.1109/ICSE.2019.00036.

[15] X. Quesada Allue, "Introducción a la estimación y planificación ágil," *proyectosagiles.org*, 2009. https://proyectosagiles.org/2009/06/08/introduccion-estimacion-planificacion-agil/ (accessed Feb. 14, 2017).

[16] M. Cohn, *User Stories applied for agile software development*. Addison Wesley. Pearson Education Inc., 2004.

[17] M. Cohn, *Agile Estimating and Planning*. New York, NY, USA, 2005.

[18] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2000.

[19] A. S. Tanenbaum and M. Van Steen, *Distributed Systems:Principles and Paradigms*, Second Edi. Pearson Education Inc, Prentice Hall, 2006.

[20] T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, Mar. 2018, pp. 11–20, doi: 10.1109/SOSE.2018.00011.

[21] O. Zimmermann, "Microservices tenets: Agile approach to service development and deployment," *Comput. Sci. - Res. Dev.*, vol. 32, no. 3–4, pp. 301–310, 2017, doi: 10.1007/s00450-016-0337-0.

[22] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in Practice, Part 1: Reality Check and Service Design," *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, 2017, doi: 10.1109/MS.2017.24.

[23] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May 2018, doi: 10.1109/MS.2018.2141039.

[24] E. Evans, *Domain-Driven Design*. Addison Wesley, 2004.

[25] V. Vernon, *Implementing Domain-Driven Design*. Addison Wesley, 2013.

[26] R. Benítez, G. Escudero, S. Kanaan, and D. Masip Rodó, *Inteligencia artificial avanzada*, Editorial UOC. Barcelona, 2013.

[27] L. Álvarez Munárriz, *Fundamentos de inteligencia artificial*. Universidad de Murcia, 1994.

[28] J. Chugh, "Types of Machine Learning and Top 10 Algorithms Everyone Should Know | Artificial Intelligence," *Oracle artificial intelligence blog*, 2018. https://blogs.oracle.com/ai/types-of-machine-learning-and-top-10-algorithms-everyone-should-know (accessed Feb. 01, 2019).

[29] B. Oster, "What are the most known algorithms used in artificial intelligence? - Quora," *Quora*, 2018. https://www.quora.com/What-are-the-most-known-algorithms-used-in-artificial-intelligence (accessed Feb. 05, 2019).

[30] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002, doi: 10.1145/505282.505283.

[31] S. López López, "Complejidad Cognitiva," *enmilocalfunciona.io*, 2018. https://enmilocalfunciona.io/complejidad-cognitiva/ (accessed Feb. 05, 2019).

[32] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, "A Suite of Object Oriented Cognitive

Complexity Metrics," *IEEE Access*, vol. 6, no. c, pp. 8782–8796, 2018, doi: 10.1109/ACCESS.2018.2791344.
[33] Y. Wang, "On the cognitive informatics foundations of software engineering," in *Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004.*, Aug. 2004, pp. 22–31, doi: 10.1109/COGINF.2004.1327456.
[34] R. V. O'Connor, P. Elger, and P. M. Clarke, "Exploring the impact of situational context — A case study of a software development process for a microservices architecture," *2016 IEEE/ACM Int. Conf. Softw. Syst. Process.*, pp. 6–10, 2016, doi: 10.1109/ICSSP.2016.009.
[35] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, and S. Gil, "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud Evaluando el Patrón de Arquitectura Monolítica y de Micro Servicios Para Desplegar Aplicaciones en la Nube," *10th Comput. Colomb. Conf.*, pp. 583–590, 2015, doi: 10.1109/ColumbianCC.2015.7333476.
[36] M. S. Hamzehloui, S. Sahibuddin, and K. Salah, "A Systematic Mapping Study on Microservices Mohammad," in *IRICT: International Conference of Reliable Information and Communication Technology 2018*, 2019, vol. 843, pp. 1079–1090, doi: 10.1007/978-3-319-99007-1.
[37] F. H. Vera-Rivera, C. M. Gaona Cuevas, and H. Astudillo, "Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación," *Rev. Ibérica Sist. e Tecnol. Informação*, vol. E23, pp. 107–120, Jul. 2019.
[38] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," 2016, Accessed: Sep. 13, 2017. [Online]. Available: http://dspace.brunel.ac.uk/bitstream/2438/14968/1/FullText.pdf.
[39] P. Di Francesco, I. Malavolta, and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Apr. 2017, pp. 21–30, doi: 10.1109/ICSA.2017.24.
[40] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The Pains and Gains of Microservices: A Systematic Grey Literature Review," *J. Syst. Softw.*, vol. 146, pp. 215–232, 2018, doi: 10.1016/j.jss.2018.09.082.
[41] J. Bogner, S. Wagner, and A. Zimmermann, "Automatically measuring the maintainability of service- and microservice-based systems," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement on - IWSM Mensura '17*, 2017, pp. 107–115, doi: 10.1145/3143434.3143443.
[42] J. Kobielus, "Building AI Microservices for Cloud-Native Deployments - Wikibon Research," *Wikibon*, 2017. https://wikibon.com/building-ai-microservices-for-cloud-native-deployments/ (accessed Feb. 13, 2019).
[43] B. Kitchenham, "Procedures for performing systematic reviews," 2004. doi: 10.1.1.122.3308.
[44] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, vol. 9783642290. 2012.
[45] 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, Accessed: May 16, 2018. [Online]. Available: https://pdfs.semanticscholar.org/fa72/91f2073cb6fdbdd7c2213bf6d776d0ab411c.pdf.
[46] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requir. Eng.*, vol. 11, no. 1, pp. 102–107, Mar. 2006, doi: 10.1007/s00766-005-0021-6.
[47] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *J. Syst. Softw.*, vol. 150, pp. 77–97, Apr. 2019, doi: 10.1016/j.jss.2019.01.001.
[48] F. Osses, G. Márquez, and H. Astudillo, "An exploratory study of academic architectural tactics and patterns in microservices: A systematic literature review," *Av. en Ing. Softw. a Niv. Iberoam. CIbSE 2018*, no. February, pp. 71–84, 2018, [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85054066307&partnerID=40&md5=e51536f75f66e6e3b6171fe7e3d0f7dd.
[49] M. Mishra, S. Kunde, and M. Nambiar, "Cracking the monolith - Challenges in Data Transitioning to Cloud Native Architectures," in *Proceedings of the 12th European Conference on Software Architecture Companion Proceedings - ECSA '18*, 2018, pp. 1–4, doi: 10.1145/3241403.3241440.
[50] C. Xu, H. Zhu, I. Bayley, D. Lightfoot, M. Green, and P. Marshall, "CAOPLE: A programming language for microservices SaaS," *Proc. - 2016 IEEE Symp. Serv. Syst. Eng. SOSE 2016*, no. 1, pp. 42–52, 2016, doi: 10.1109/SOSE.2016.46.
[51] D. Liu *et al.*, "CIDE: An integrated development environment for microservices," *Proc. - 2016 IEEE Int. Conf. Serv. Comput. SCC 2016*, no. 0, pp. 808–812, 2016, doi: 10.1109/SCC.2016.112.
[52] P. Di Francesco, "Architecting microservices," *Proc. - 2017 IEEE Int. Conf. Softw. Archit. Work. ICSAW 2017 Side Track Proc.*, pp. 224–229, 2017, doi: 10.1109/ICSAW.2017.65.
[53] J. Sorgaila, P. Wizenty, F. Rademacher, S. Sachweh, and A. Zündorf, "AjiL - Enabling Model-driven Microservice Development Jonas," in *Proceedings of the 12th European Conference on Software Architecture Companion Proceedings - ECSA '18*, 2018, pp. 1–4, doi: 10.1145/3241403.3241406.
[54] P. de Lange, P. Nicolaescu, R. Klamma, and M. Jarke, "Engineering Web Applications Using Real-Time Collaborative Modeling," Springer, Cham, 2017, pp. 213–228.
[55] M. Kleehaus, Ö. Uludağ, P. Schäfer, and F. Matthes, "MICROLYZE: A Framework for Recovering the Software Architecture in Microservice-Based Environments," in *CAiSE 2018: Information Systems in the Big Data Era*,

Jun. 2018, pp. 148–162, doi: 10.1007/978-3-319-92901-9_14.
[56] R. Oberhauser, “Microflows: Lightweight Automated Planning and Enactment of Workflows Comprising Semantically-Annotated Microservices,” *Proc. Sixth Int. Symp. Bus. Model. Softw. Des.*, no. Bmsd, pp. 134–143, 2016, doi: 10.5220/0006223001340143.
[57] Cambridge University Press, “GRANULAR | significado, definición en el Cambridge English Dictionary.” https://dictionary.cambridge.org/es-LA/dictionary/english/granular (accessed Oct. 21, 2020).
[58] H. Vural, M. Koyuncu, and S. Guney, “A Systematic Literature Review on Microservices,” Springer, Cham, 2017, pp. 203–217.
[59] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kroger, “Microservice Decomposition via Static and Dynamic Analysis of the Monolith,” *Proc. - 2020 IEEE Int. Conf. Softw. Archit. Companion, ICSA-C 2020*, pp. 9–16, 2020, doi: 10.1109/ICSA-C50368.2020.00011.
[60] O. Al-Debagy and P. Martinek, “Extracting Microservices’ Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach,” in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Jun. 2020, pp. 289–294, doi: 10.1109/SoSE50414.2020.9130466.
[61] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, “Service Candidate Identification from Monolithic Systems based on Execution Traces,” *IEEE Trans. Softw. Eng.*, vol. X, no. X, pp. 1–1, 2019, doi: 10.1109/TSE.2019.2910531.
[62] M. Abdullah, W. Iqbal, and A. Erradi, “Unsupervised learning approach for web application auto-decomposition into microservices,” *J. Syst. Softw.*, vol. 151, pp. 243–257, 2019, doi: 10.1016/j.jss.2019.02.031.
[63] S. Li *et al.*, “A dataflow-driven approach to identifying microservices from monolithic applications,” *J. Syst. Softw.*, vol. 157, 2019, doi: 10.1016/j.jss.2019.07.008.
[64] D. Taibi and K. Syst, “From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining,” *Int. Conf. Cloud Comput. Serv. Sci. - CLOSER 2019*, no. March, 2019.
[65] N. Santos *et al.*, “A logical architecture design method for microservices architectures,” in *ACM International Conference Proceeding Series*, Sep. 2019, vol. 2, pp. 145–151, doi: 10.1145/3344948.3344991.
[66] O. Al-Debagy and P. Martinek, “A new decomposition method for designing microservices,” *Period. Polytech. Electr. Eng. Comput. Sci.*, vol. 63, no. 4, pp. 274–281, 2019, doi: 10.3311/PPee.13925.
[67] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, “Business object centric microservices patterns,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11877 LNCS, pp. 476–495, doi: 10.1007/978-3-030-33246-4_30.
[68] L. Nunes, N. Santos, and A. Rito Silva, “From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts,” in *13th European Conference, ECSA 2019. Lectures Notes in Computer Science 11681*, 2019, pp. 37–52, doi: 10.1007/978-3-030-29983-5_3.
[69] A. Homay, A. Zoitl, M. De Sousa, M. Wollschlaeger, and C. Chrysoulas, “Granularity cost analysis for function block as a service,” *IEEE Int. Conf. Ind. Informatics*, vol. 2019-July, pp. 1199–1204, 2019, doi: 10.1109/INDIN41052.2019.8972205.
[70] M. Cojocaru, A. Uta, and A. M. Oprescu, “MicroValid: A validation framework for automatically decomposed microservices,” in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, Dec. 2019, vol. 2019-Decem, pp. 78–86, doi: 10.1109/CloudCom.2019.00023.
[71] A. Christoforou, L. Odysseos, and A. Andreou, “Migration of Software Components to Microservices: Matching and Synthesis,” in *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2019, pp. 134–146, doi: 10.5220/0007732101340146.
[72] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, “Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search,” in *17th International Conference Service-Oriented Computing. Lectures Notes in computer science 11895*, Oct. 2019, pp. 58–63, doi: 10.1007/978-3-030-33702-5_5.
[73] M. I. Josélyne, D. Tuheirwe-Mukasa, B. Kanagwa, and J. Balikuddembe, “Partitioning microservices - A Domain Engineering Approach,” in *Proceedings of the 2018 International Conference on Software Engineering in Africa - SEiA ’18*, 2018, pp. 43–49, doi: 10.1145/3195528.3195535.
[74] H. Vural, M. Koyuncu, and S. Misra, “A Case Study on Measuring the Size of Microservices,” in *International Conference on Computational Science and Its Applications - ICCSA 2018*, 2018, pp. 454–463, doi: 10.1007/b98054.
[75] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, “Identifying Microservices Using Functional Decomposition,” in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, 2018, vol. 10998, pp. 50–65, doi: 10.1007/978-3-319-99933-3.
[76] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, “Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems,” 2018, pp. 37–53.
[77] M. Tusjunt and W. Vatanawood, “Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern,” in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Dec. 2018, pp. 609–613, doi: 10.1109/CompComm.2018.8781036.
[78] Z. Ren *et al.*, “Migrating web applications from monolithic structure to microservices architecture,” in *ACM International Conference Proceeding Series*, Sep. 2018, pp. 1–10, doi: 10.1145/3275219.3275230.
[79] W. Hasselbring and G. Steinacker, “Microservice architectures for scalability, agility and reliability in e-

commerce," in *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 2017, pp. 243–246, doi: 10.1109/ICSAW.2017.11.
[80] J. P. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture," in *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 2017, pp. 62–65, doi: 10.1109/ICSAW.2017.35.
[81] D. Shadija, M. Rezai, and R. Hill, "Microservices: Granularity vs. Performance," in *UCC '17 Companion Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 215–220, doi: 10.1145/3147234.3148093.
[82] S. Hassan, N. Ali, and R. Bahsoon, "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity," in *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, Apr. 2017, pp. 1–10, doi: 10.1109/ICSA.2017.32.
[83] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *European Conference on Service-Oriented and Cloud Computing - Lecture Notes in Computer Science.*, Sep. 2017, vol. 10465 LNCS, pp. 19–33, doi: 10.1007/978-3-319-67262-5_2.
[84] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 524–531, doi: 10.1109/ICWS.2017.61.
[85] G. Kecskemeti, A. Kertesz, and A. C. Marosi, "Towards a methodology to form microservices from monolithic ones," in *Euro-Par 2016 Workshops - Lecture Notes in Computer Science*, 2017, vol. 10104 LNCS, pp. 284–295, doi: 10.1007/978-3-319-58943-5_23.
[86] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," *Proc. - 2016 IEEE Int. Conf. Serv. Comput. SCC 2016*, pp. 813–818, 2016, doi: 10.1109/SCC.2016.113.
[87] M. Ahmadvand and A. Ibrahim, "Requirements Reconciliation for Scalable and Secure Microservice (De)composition," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Sep. 2016, pp. 68–73, doi: 10.1109/REW.2016.026.
[88] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2017, pp. 466–475, doi: 10.1109/APSEC.2017.53.
[89] G. Kecskemeti, A. C. Marosi, and A. Kertesz, "The ENTICE approach to decompose monolithic services into microservices," *2016 Int. Conf. High Perform. Comput. Simulation, HPCS 2016*, pp. 591–596, 2016, doi: 10.1109/HPCSim.2016.7568389.
[90] P. Cruz, H. Astudillo, R. Hilliard, and M. Collado, "Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM," *2019 IEEE Int. Conf. Softw. Archit. Companion*, pp. 174–181, 2019, doi: 10.1109/icsa-c.2019.00039.
[91] M. I. Rahman, S. Panichella, and D. Taibi, "A Curated Dataset of Microservices-Based Systems," in *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution (CEUR Workshop Proceedings; Vol. 2520). CEUR-WS.*, 2019, vol. 2520, Accessed: Feb. 14, 2020. [Online]. Available: http://research.tuni.fi/clowee.
[92] G. Marquez and H. Astudillo, "Actual Use of Architectural Patterns in Microservices-Based Open Source Projects," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Jul. 2018, vol. 2018-December, pp. 31–40, doi: 10.1109/APSEC.2018.00017.
[93] C. Richardson and microservices.io, "Microservice Architecture pattern." https://microservices.io/patterns/microservices.html (accessed Dec. 12, 2019).
[94] O. Zimmermann, M. Stocker, U. Zdun, D. Lübke, and C. Pautasso, "Microservice API Patterns," 2019. https://www.microservice-api-patterns.org/introduction (accessed Dec. 17, 2019).
[95] F. H. Vera-Rivera and F. A. Rojas Morales, "Propuesta de aplicación de la Ingeniería del Software Basada en Componentes en el desarrollo de software empresarial.," *Rev. Iteckne*, vol. 7, no. 2, pp. 128–135, 2010.
[96] F. H. Vera-Rivera, "A development process of enterprise applications with microservices," *Encuentro Int. Ciencias Apl. e Ing. (Universidad - Ind. – EISI 2018. J. Phys. Conf. Ser.*, vol. 1126, no. 17, p. 012017, Nov. 2018, doi: 10.1088/1742-6596/1126/1/012017.
[97] L. Bass, P. Clemens, and R. Katzman, *Software Architecture in Practice*. Addison Wesley, 1998.
[98] H. Astudillo, "Five ontological levels to describe and evaluate software architectures," *Rev. Fac. Ing. Univ. Tarapacá*, vol. 13, no. 1, pp. 69–73, 2005, [Online]. Available: https://scielo.conicyt.cl/pdf/rfacing/v13n1/art08.pdf.
[99] H. Alipour and Y. Liu, "Online machine learning for cloud resource provisioning of microservice backend systems," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017, pp. 2433–2441, doi: 10.1109/BigData.2017.8258201.
[100] S.-P. Ma, Y. Chuang, C.-W. Lan, H.-M. Chen, C.-Y. Huang, and C.-Y. Li, "Scenario-Based Microservice Retrieval Using Word2Vec," in *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, Oct. 2018, pp. 239–244, doi: 10.1109/ICEBE.2018.00046.
[101] J. Bogner, S. Wagner, and A. Zimmermann, "Towards a practical maintainability quality model for service-and microservice-based systems," in *Proceedings of the 11th European Conference on Software Architecture Companion Proceedings - ECSA '17*, 2017, vol. 3, pp. 195–198, doi: 10.1145/3129790.3129816.

[102] I. Candela, G. Bavota, B. Russo, and R. Oliveto, “Using cohesion and coupling for software remodularization: Is it enough?,” *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, May 2016, doi: 10.1145/2928268.
[103] D. Rud, A. Schmietendorf, and R. R. Dumke, “Product Metrics for Service-Oriented Infrastructures,” in *Conference: Applied Software Measurement. Proceedings of the International Workshop on Software Metrics and DASMA Software Metrik Kongress (IWSM/MetriKon 2006)*, 2006, Accessed: Jun. 18, 2019. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.6887&rep=rep1&type=pdf.
[104] H. Gall, M. Jazayeri, and J. Krajewski, “CVS release history data for detecting logical couplings,” in *International Workshop on Principles of Software Evolution (IWPSE)*, 2003, vol. 2003-January, pp. 13–23, doi: 10.1109/IWPSE.2003.1231205.
[105] R. C. Martin, *Agile Software Development Principles, Patterns, and Practices Alan Apt Series*. Upper Saddle River, New Jersey 07458: Pearson Education, Inc., 2002.
[106] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*, Second edi. 2012.
[107] M. Perepletchikov, C. Ryan, and K. Frampton, “Cohesion Metrics for Predicting Maintainability of Service-Oriented Software,” in *Seventh International Conference on Quality Software (QSIC 2007)*, 2007, pp. 328–335, doi: 10.1109/QSIC.2007.4385516.
[108] Totalmetrics.com, “Total Metrics Approach - Function points,” *www.totalmetrics.com*. https://www.totalmetrics.com/our_approach (accessed Aug. 05, 2020).
[109] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, “Coupling Metrics for Predicting Maintainability in Service-Oriented Designs,” in *2007 Australian Software Engineering Conference (ASWEC’07)*, Apr. 2007, pp. 329–340, doi: 10.1109/ASWEC.2007.17.
[110] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, “A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures,” Springer, Berlin, Heidelberg, 2009, pp. 41–52.
[111] E. Bouwers, J. P. Correia, A. Van Deursen, and J. Visser, “Quantifying the analyzability of software architectures,” *Proc. - 9th Work. IEEE/IFIP Conf. Softw. Archit. WICSA 2011*, pp. 83–92, 2011, doi: 10.1109/WICSA.2011.20.
[112] P. Calçado, “Pattern: Service Mesh,” 2017. http://philcalcado.com/2017/08/03/pattern_service_mesh.html (accessed Nov. 07, 2018).
[113] E. Mueller, J. Wickett, K. Gaekwad, and P. Karayanev, “What Is DevOps? | the agile admin,” 2017. https://theagileadmin.com/what-is-devops/ (accessed Mar. 12, 2018).
[114] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, 2016, doi: 10.1109/MS.2016.68.
[115] K. Bakshi, “Microservices-based software architecture and approaches,” in *2017 IEEE Aerospace Conference*, Mar. 2017, pp. 1–8, doi: 10.1109/AERO.2017.7943959.
[116] F. H. Vera-Rivera, J. L. Vera-Rivera, and C. M. Gaona-Cuevas, “Sinplafut: A microservices – based application for soccer training,” in *5th International Week of Science, Technology & Innovation. Journal of Physics: Conference Series*, Nov. 2019, vol. 1388, no. 2, p. 012026, doi: 10.1088/1742-6596/1388/1/012026.
[117] Y. Jayawardana, R. Fernando, G. Jayawardena, D. Weerasooriya, and I. Perera, “A Full Stack Microservices Framework with Business Modelling,” in *ICTer 2018*, 2018, no. September.
[118] Airbnb.io, “Synapse Airbnb.” https://airbnb.io/projects/synapse/ (accessed Feb. 02, 2021).
[119] Spacy.io, “Models · spaCy Models Documentation,” 2020. https://spacy.io/models (accessed Jun. 12, 2020).
[120] Spacy.io, “Word Vectors and Semantic Similarity · spaCy Usage Documentation.” https://spacy.io/usage/vectors-similarity#basics (accessed Nov. 20, 2020).
[121] J. Holland, *Adaptation in natural and artificial systems*. Michigan: University of Michigan Press, 1975.
[122] F. Herrera, M. Lozano, and J. L. Verdegay, *Algoritmos Genéticos: Fundamentos, Extensiones y Aplicaciones*. ProQuest, 1995.
[123] E. Evans, *Domain-Driven Design Reference - Definitions and Pattern Summaries*. 2015.
[124] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, “Service Cutter: A Systematic Approach to Service Decomposition,” in *IFIP International Federation for Information Processing 2016*, 2016, pp. 185–200, doi: 10.1007/978-3-319-44482-6_12.
[125] mybatis.org, “Mybatis Jpetstore-6: A web application built on top of MyBatis 3, Spring 3 and Stripes.” https://github.com/mybatis/jpetstore-6 (accessed Nov. 22, 2020).

8. ANEXOS

8.1. APLICACIÓN CARGO-TRACKING DETALLADA

En la sección 5.4.1 se describió este caso de estudio, Según lo publicado por Baresi y otros (2017) [83] a partir del modelo del dominio se extrajeron las posibles historias de usuario de esta aplicación de ejemplo, con el fin de evaluar y hacer una prueba del "Microservices Backlog" y luego comparar con lo propuesto por ellos y en los ejemplos de validación de su método. El modelo del domino propuesto por Baresi y otros se puede apreciar en la figura 73.

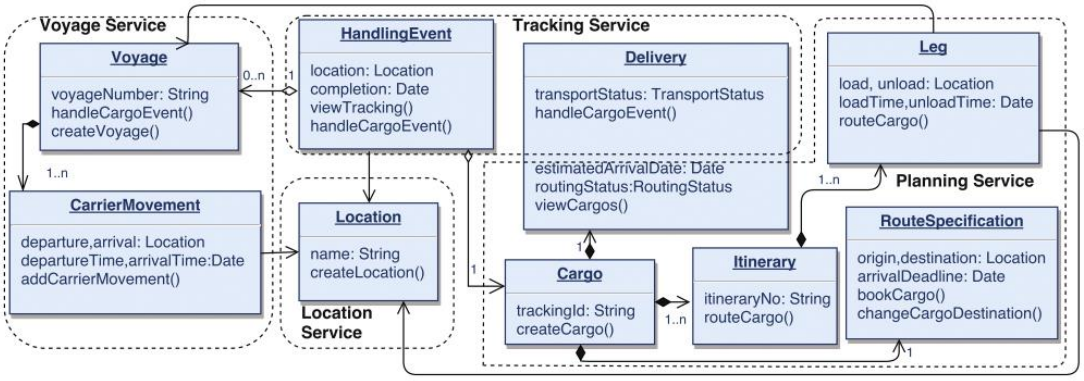


Figura 73. Modelo del dominio de la aplicación Cargo-Tracking. Fuente Baresi y otros [83]

Para el ejemplo se usó el nombre y la descripción de las historias de usuario y sus entidades en español. En la tabla 33 se presenta el ejemplo de la ficha usada para describir las historias de usuario.

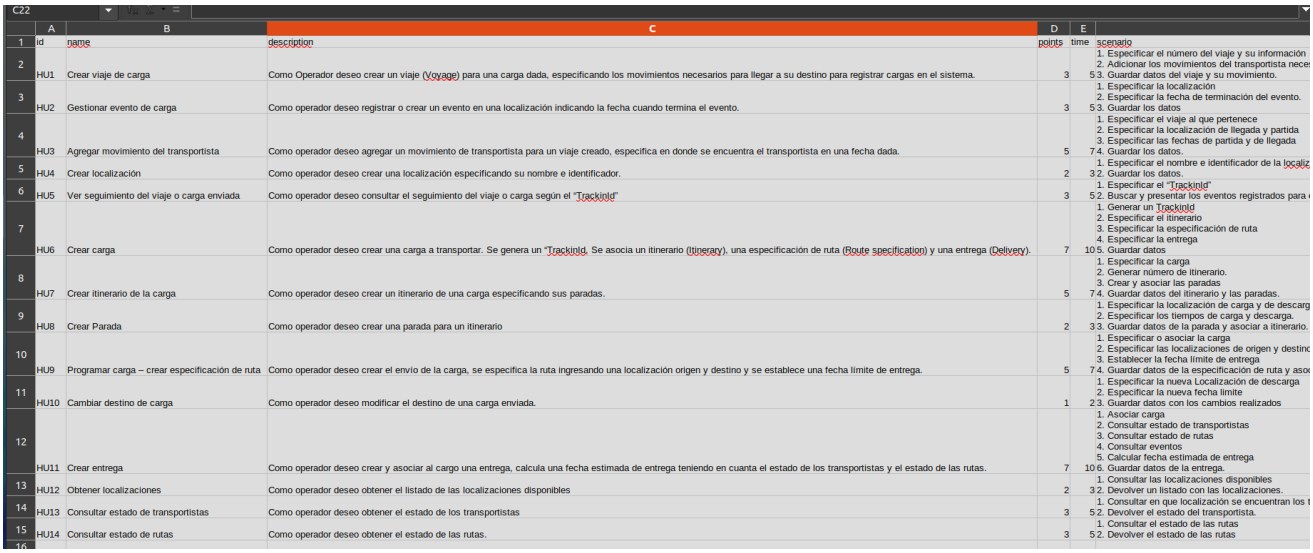
Tabla 33. Ficha usada para identificar las historias de usuario

Historia de Usuario # 1	
Dato	Descripción
Identificador	Hu1
Nombre	Crear viaje de carga - create voyage
Definición – descripción	Como usuario requiero crear un viaje (voyage) para una carga dada. Especificando los movimientos necesarios para llegar a su destino.
Estimación – puntos	3
Estimación – tiempo	5h
Dependencia	Al crear un viaje se deben especificar los movimientos entre localizaciones que debe tener el transportista Las localizaciones deben estar creadas Debe consultar las localizaciones de cada movimiento
Observaciones - escenario	Especificar el número del viaje y su información Adicionar los movimientos del transportista necesarios para el viaje desde su inicio hasta su final. Guardar datos del viaje y su movimiento.

Los puntos y los tiempos estimados corresponden al esfuerzo y al tiempo que implicaría desarrollar cada historia de usuario en orden consecutivo. En un desarrollo usando metodologías ágiles esta estimación la haría el equipo de desarrollo teniendo en cuenta sus propias características.

En la figura 74 se puede apreciar el archivo CVS, que muestra el detalle de las historias de usuario identificadas para este caso de estudio. Este archivo se utiliza en el sistema para cargar las historias de usuario para agilizar el proceso de identificación de microservicios a través del Microservices Backlog.

El “producto backlog” resultante se puede apreciar en la tabla 34, se identificaron 14 historias de usuario, con un total de 51 puntos de historia y 77 horas de tiempo estimado de desarrollo (si se desarrollan las historias de usuario secuencialmente una tras otra).



	A	B	C	D	E	
1	Id	name	description	points	time	scenario
2	HU1	Crear viaje de carga	Como Operador deseo crear un viaje (Voyage) para una carga dada, especificando los movimientos necesarios para llegar a su destino para registrar cargas en el sistema.	3	5	1. Especificar el número del viaje y su información 2. Adicionar los movimientos del transportista nece 3. Guardar datos del viaje y su movimiento.
3	HU2	Gestionar evento de carga	Como operador deseo registrar o crear un evento en una localización indicando la fecha cuando termina el evento.	3	5	1. Especificar la localización 2. Especificar la fecha de terminación del evento. 3. Guardar los datos
4	HU3	Agregar movimiento del transportista	Como operador deseo agregar un movimiento de transportista para un viaje creado, especifica en donde se encuentra el transportista en una fecha dada.	5	7	1. Especificar el viaje al que pertenece 2. Especificar la localización de llegada y partida 3. Especificar las fechas de partida y de llegada 4. Guardar los datos.
5	HU4	Crear localización	Como operador deseo crear una localización especificando su nombre e identificador.	2	3	1. Especificar el nombre e identificador de la localiz 2. Guardar los datos.
6	HU5	Ver seguimiento del viaje o carga enviada	Como operador deseo consultar el seguimiento del viaje o carga según el “TrackingId”	3	5	1. Especificar el “TrackingId” 2. Buscar y presentar los eventos registrados para
7	HU6	Crear carga	Como operador deseo crear una carga a transportar. Se genera un “TrackingId. Se asocia un itinerario (Itinerary), una especificación de ruta (Route specification) y una entrega (Delivery).	7	10	1. Generar un TrackingId 2. Especificar el itinerario 3. Especificar la especificación de ruta 4. Especificar la entrega 5. Guardar datos
8	HU7	Crear itinerario de la carga	Como operador deseo crear un itinerario de una carga especificando sus paradas.	5	7	1. Especificar la carga 2. Generar número de itinerario. 3. Crear y asociar las paradas 4. Guardar datos del itinerario y las paradas.
9	HU8	Crear Parada	Como operador deseo crear una parada para un itinerario	2	3	1. Especificar la localización de carga y de descarg 2. Especificar los tiempos de carga y descarga. 3. Guardar datos de la parada y asociar a itinerario.
10	HU9	Programar carga – crear especificación de ruta	Como operador deseo crear el envío de la carga, se especifica la ruta ingresando una localización origen y destino y se establece una fecha límite de entrega.	5	7	1. Especificar o asociar la carga 2. Especificar las localizaciones de origen y destino 3. Establecer la fecha límite de entrega 4. Guardar datos de la especificación de ruta y aso
11	HU10	Cambiar destino de carga	Como operador deseo modificar el destino de una carga enviada.	1	2	1. Especificar la nueva Localización de descarga 2. Especificar la nueva fecha límite 3. Guardar datos con los cambios realizados
12	HU11	Crear entrega	Como operador deseo crear y asociar al cargo una entrega, calcula una fecha estimada de entrega teniendo en cuanta el estado de los transportistas y el estado de las rutas.	7	10	1. Asociar carga 2. Consultar estado de transportistas 3. Consultar estado de rutas 4. Consultar eventos 5. Calcular fecha estimada de entrega 6. Guardar datos de la entrega.
13	HU12	Obtener localizaciones	Como operador deseo obtener el listado de las localizaciones disponibles	2	3	1. Consultar las localizaciones disponibles 2. Devolver un listado con las localizaciones.
14	HU13	Consultar estado de transportistas	Como operador deseo obtener el estado de los transportistas	3	5	1. Consultar en que localización se encuentran los t 2. Devolver el estado del transportista.
15	HU14	Consultar estado de rutas	Como operador deseo obtener el estado de las rutas.	3	5	1. Consultar el estado de las rutas 2. Devolver el estado de las rutas

Figura 74. Archivo CVS para cargar las historias de usuario al sistema.

En la figura 74 se pueden apreciar todas las historias de usuario identificadas para este caso de estudio.

Tabla 34. Product Backlog para el caso de estudio Cargo Tracking

ID	Nombre	Puntos	Tiempo Des. (hours)
HU_1	Crear viaje de carga	3	5
HU_2	Gestionar evento de carga	3	5
HU_3	Agregar movimiento del transportista	5	7
HU_4	Crear localización	2	3
HU_5	Ver seguimiento del viaje	3	5
HU_6	Crear carga	7	10
HU_7	Crear itinerario de la carga	5	7
HU_8	Crear Parada	2	3
HU_9	Programar carga – crear especificación de ruta	5	7
HU_{10}	Cambiar destino de carga	1	2
HU_{11}	Crear entrega	7	10
HU_{12}	Obtener localizaciones	2	3
HU_{13}	Consultar estado de transportistas	3	5
HU_{14}	Consultar estado de rutas	3	5
	Total:	51	77

ID: Identificador de la historia de usuario. Puntos: Puntos de historia estimados. Tiempo Des: Tiempo de Desarrollo estimado.

Definidas las historias de usuario y el producto backlog se procede a identificar las dependencias entre ellas. En este caso las dependencias se definieron a partir de invocaciones entre historias de usuario y el flujo de datos de la aplicación. La tabla 35 describe las dependencias identificadas para este caso de estudio.

Tabla 35. Dependencias entre las historias de usuarios para la aplicación Cargo Tracking

Historia de usuario	Dependencias	Historias de usuario	Dependencias
HU_1	{HU_{12}, HU_3}	HU_8	{HU_{12}}
HU_2	{HU_{12}}	HU_9	{HU_{12}}
HU_3	{HU_{12}}	HU_{10}	{HU_{12}}
HU_4	{}	HU_{11}	{HU_6, HU_{13}, H_{U14}}
HU_5	{}	HU_{12}	{}
HU_6	{HU_7, HU_9, HU_{11}}	HU_{13}	{HU_5}
HU_7	{HU_8}	HU_{14}	{HU_5}

Teniendo identificadas las historias de usuario y sus dependencias, se pueden cargar en el sistema para obtener las descomposiciones candidatas, evaluarlas y seleccionar la mejor distribución de historias de usuario en los microservicios.

8.1.1. Identificación de microservicios usando diseño impulsado por el dominio (DDD)

Al usar DDD, siguiendo los planteamientos hechos por Evan [24], se deben identificar las entidades, los objetos de valor, los contextos delimitados para así poder identificar los microservicios que van a ser parte de la aplicación.

Entidades Identificadas:

1. Carga
2. Evento carga
3. Movimiento Transportista
4. Itinerario
5. Entrega
6. Localización
7. Parada
8. Viaje
9. Especificación de ruta

Agregados: Se relacionan los nombres de los agregados y las entidades que hacen parte de él.

1. Viaje: Viaje, Movimiento Transportista
2. Carga: Carga, itinerario, especificación de ruta, Parada

Contextos delimitados: Se relacionan los nombres de los contextos delimitados, las entidades asociadas y las historias de usuario relacionadas.

1. Viaje: Viaje, Movimiento Transportista
HU1 - Crear viaje de carga
HU3 - Agregar movimiento del transportista
HU13 - Consultar estado de transportistas

2. Planificación de viaje: Carga, itinerario, especificación de ruta, Parada, Entrega
HU6 - Crear carga

HU7 - Crear itinerario de la carga
HU8 - Crear Parada
HU9 - Programar carga
HU10 - Cambiar destino de carga
HU11 - Crear entrega

3. Localización: Localización
HU4 - Crear localización
HU12 - Obtener localizaciones

4. Seguimiento: Evento carga
HU2 - Gestionar evento de carga
HU5 - Ver seguimiento del viaje o carga enviada
HU14 - Consultar estado de rutas

Por lo tanto, se identificaron 4 microservicios: Viaje, Planificación de Viaje, Localización y Seguimiento, correspondiendo con el planteamiento hecho por Baresi y otros en su artículo. La figura 43 detalla el diagrama Microservices Backlog para este caso de estudio, donde se muestran los 4 microservicios con sus dependencias y métricas.

En el sistema se creó una descomposición de forma manual, se crearon los 4 microservicios identificados y se relacionaron sus historias de usuario. A continuación, se explica el proceso llevado a cabo en el sistema.

8.1.2. Componente Parametrizador: Crear proyecto, cargar historias y definir las dependencias

Una vez identificadas las historias de usuario y sus dependencias se crean en el sistema el proyecto para el caso de estudio; la figura 75 muestra el formulario para crear el proyecto, donde se especifica el nombre del proyecto, abreviatura, la descripción y el lenguaje en que estarán escritas las historias de usuario.

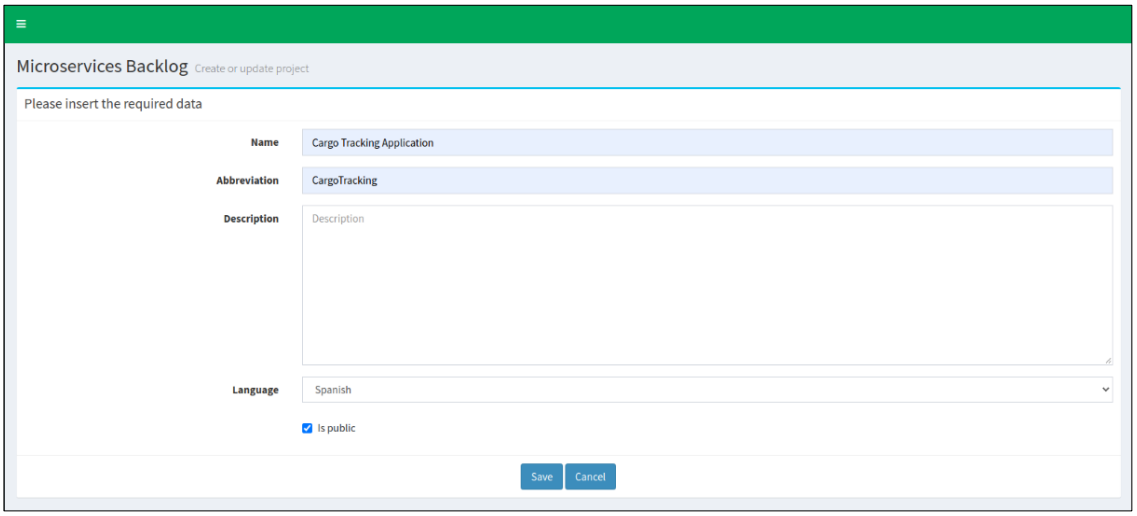


Figura 75. Formulario para crear proyecto en el Microservices Backlog.

Creado el proyecto aparecen el listado de proyectos que han sido creados en el sistema (Ver figura 76), cuando se ingresan las historias de usuario se calculan el total de puntos de historia y el tiempo estimado de desarrollo.

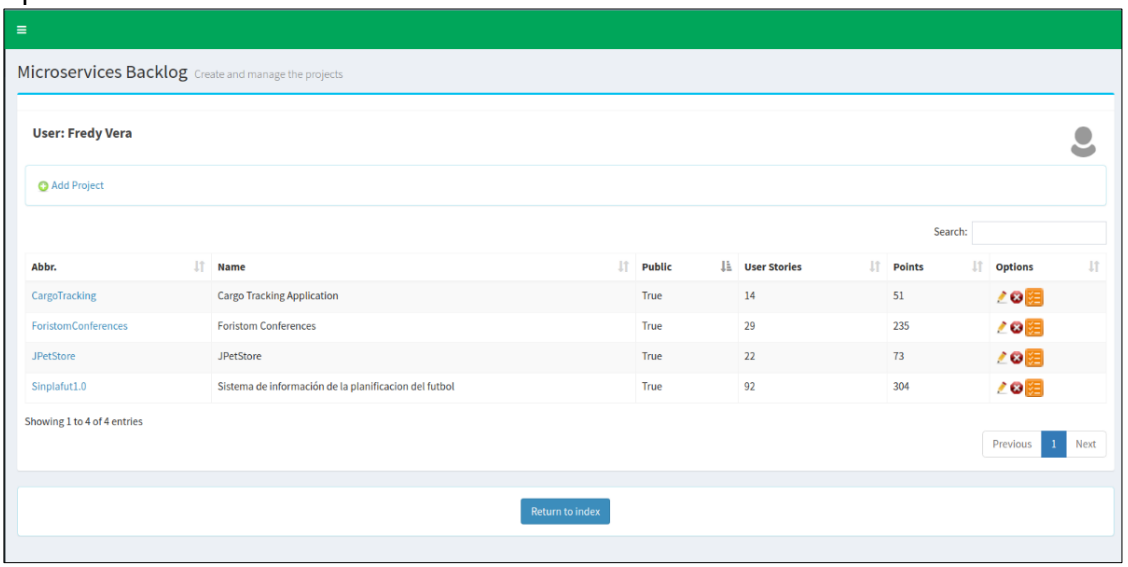


Figura 76. Gestión de proyectos ingresados al Microservices Backlog.

Las funcionalidades disponibles para aplicar a cada proyecto aparecen en la última columna, las cuales de izquierda a derecha corresponden a 1) editar proyecto, 2) eliminar proyecto y 3) gestión de historias de usuario del proyecto.

Al ingresar a la gestión de historias de usuario se presenta la interfaz de la figura 77, la cual corresponde al “product backlog” del proyecto ingresado.

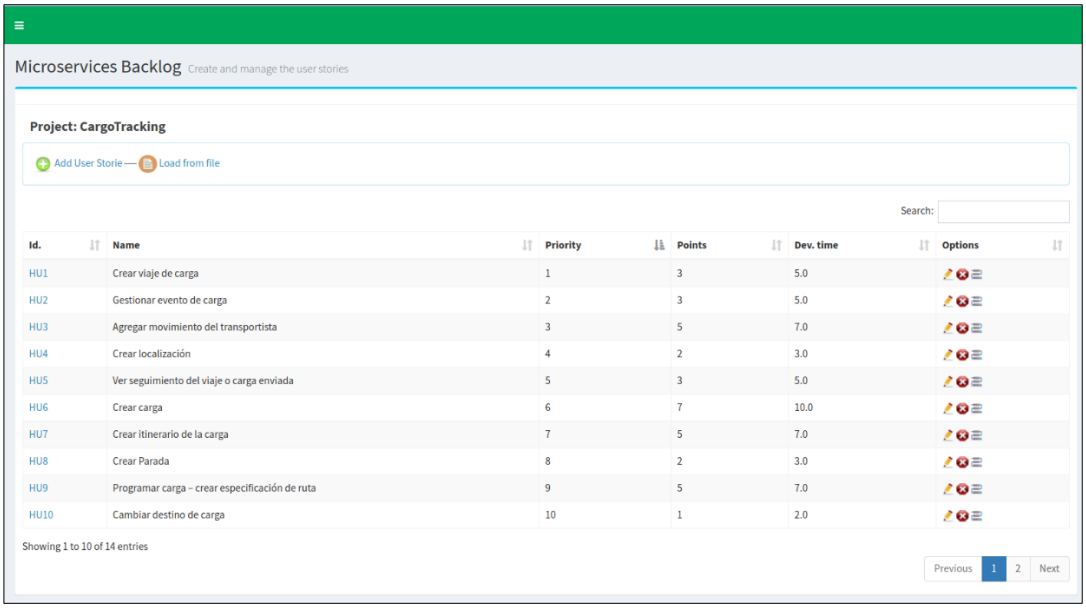


Figura 77. Gestión de historias de usuario.

En la parte superior aparecen las opciones de adicionar historia de usuario, en la cual se ingresan manualmente los datos de la historia de usuario (Id, nombre, descripción, prioridad, puntos estimados, tiempo estimado, el escenario y las observaciones); también aparece la opción de

cargar las historias de usuario desde un archivo CVS (ver figura 78), la estructura del archivo CVS se presenta en la figura 74. Al cargar el archivo se muestra la interfaz de la figura 77 con las historias de usuario subidas.

En la última columna de la figura 77 aparecen las funcionalidades disponibles para ejecutar sobre cada historia de usuario, las cuales son 1) editar historia de usuario, 2) eliminar historia de usuario y 3) definir las dependencias de la historia de usuario.

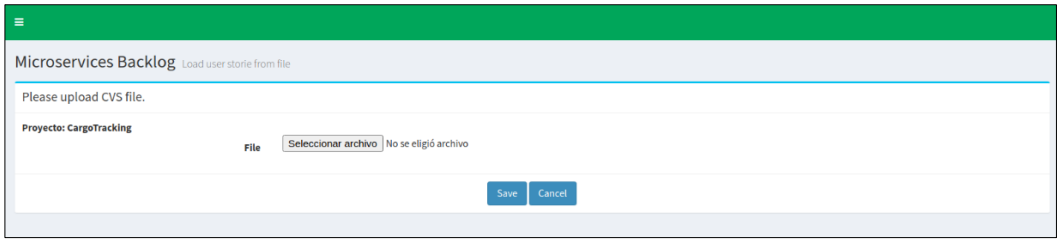


Figura 78. Cargar las historias de usuario al sistema.

Para definir las dependencias entre las historias de usuario, las cuales deben ser previamente identificadas, se utiliza la interfaz de la figura 79.

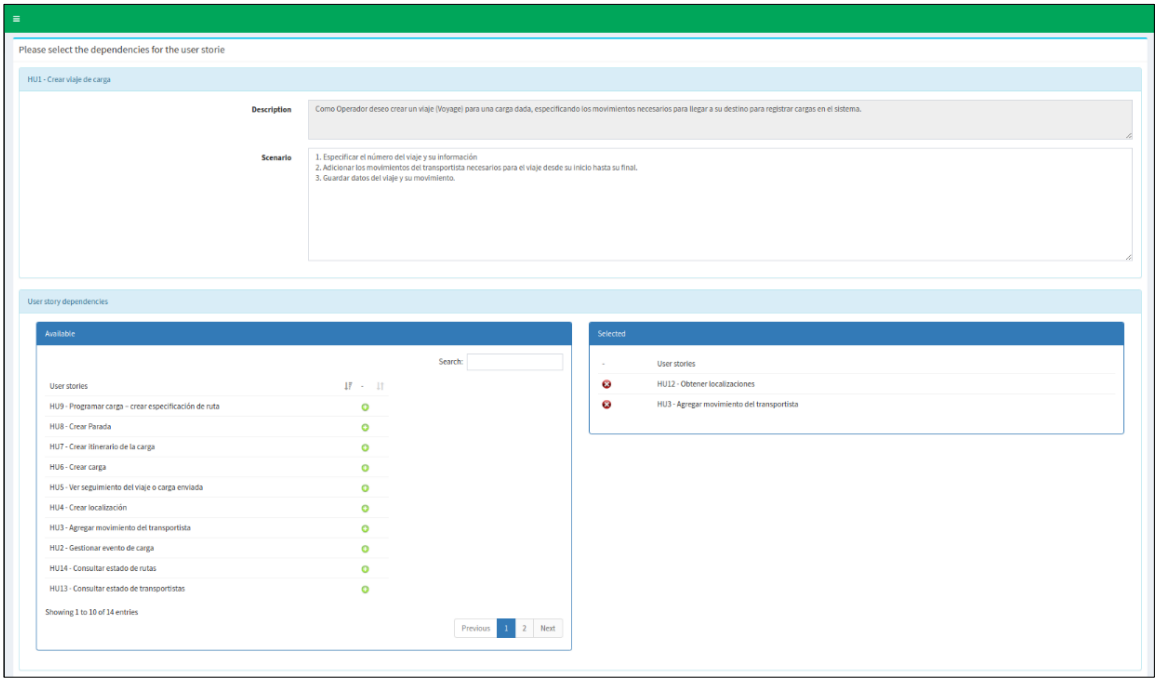


Figura 79. Definir las dependencias entre las historias de usuario.

En la parte superior de la interfaz de definición de dependencias (figura 79), aparece el nombre, la descripción y el escenario definido para la historia de usuario seleccionada para definir sus dependencias; luego en la parte izquierda aparecen las historias de usuario disponibles para ser seleccionadas y en la parte derecha las historias de usuario que han sido seleccionadas como dependencia. El usuario puede agregar (botón verde) o eliminar (botón rojo) las historias de usuario de las dependencias.
Teniendo definidas las historias de usuario del proyecto y sus dependencias se procede a ingresar o generar automáticamente las descomposiciones para el proyecto a través del componente agrupador.

8.1.3. Componente agrupador: Crear descomposiciones, ejecutar el algoritmo genético y de agrupamiento

Primero se debe crear aplicación basadas en microservicios (o descomposición) a través del formulario presentado en la figura 80.

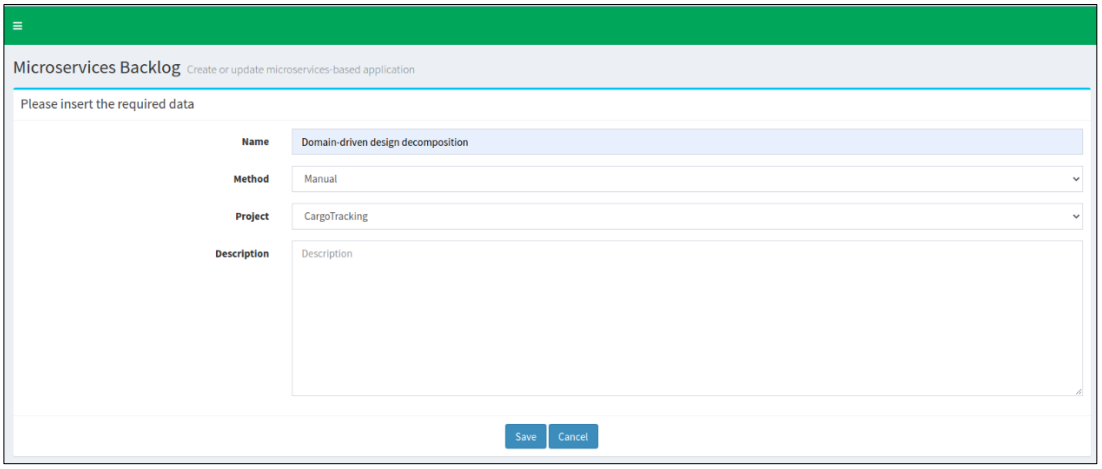


Figura 80. Agregar descomposición o aplicación basada en microservicios.

En el formulario se ingresa el nombre de la descomposición, el método por el cual se va a generar la descomposición, el proyecto al que pertenece y la descripción.

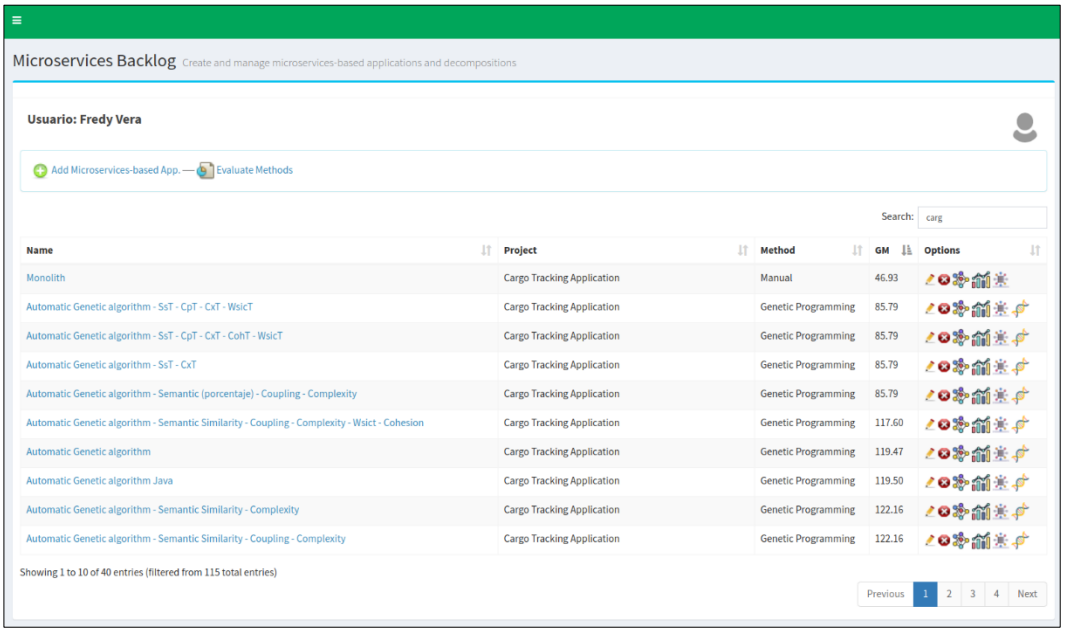


Figura 81. Agregar descomposición o aplicación basada en microservicios.

Los métodos disponibles son: 1) Manual, el usuario crea los microservicios y relaciona las historias de usuario a los microservicios por el mismo; de esta forma se ingresaron al sistema las descomposiciones obtenidas por los métodos del estado del arte comparados en esta investigación; 2) Programación genética, este es un método automático que asocia las historias

de usuario a los microservicios de forma óptima; y 3) Algoritmo de agrupamiento, corresponde a otro método automático que usa similitud semántica y métricas de acoplamiento para agrupar los microservicios.

Después de agregar la descomposición se presenta el listado de las aplicaciones creadas (ver figura 81). En la parte superior de esa interfaz aparecen dos opciones 1) agregar aplicación basada en microservicios y 2) evaluar métodos (ver figura 92). La primera opción permite agregar una nueva descomposición (ver figura 80).

En la columna de opciones aparecen 1) Editar la descomposición o aplicación, 2) eliminar la descomposición o aplicación, 3) gestionar microservicios, 4) calcular métricas, 5) el diagrama microservices backlog, 6) ejecutar algoritmo genético (sólo sale cuando el método seleccionado es el algoritmo genético), o también aparece la opción de ejecutar algoritmo de agrupamiento (sólo sale cuando el método seleccionado es el algoritmo de agrupamiento).

Al seleccionar la opción de gestionar microservicios se presenta el listado de microservicios creados u obtenidos para la aplicación seleccionada (ver figura 82).

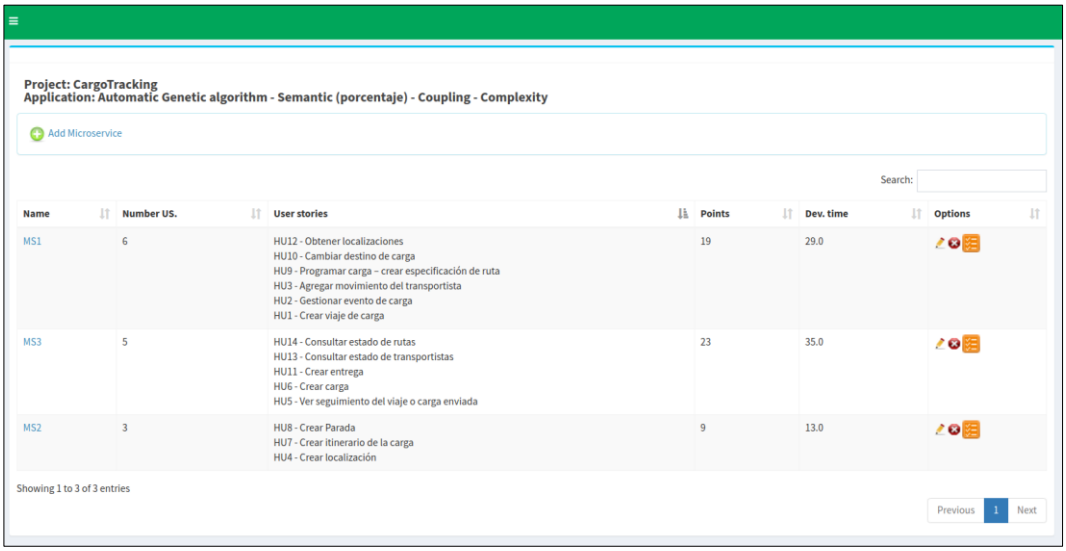


Figura 82. Gestión de microservicios de la descomposición o aplicación basada en microservcios.

En la parte superior aparece la opción de agregar microservicio (ver figura 83), la cual permite crear manualmente un microservicio para posteriormente asignarle historias de usuario, puede ser en el caso de querer dividir dos microservicios obtenidos automáticamente o si el usuario estima conveniente crear otro microservicio, o también para ingresar manualmente una descomposición.

En la columna de opciones (figura 82) aparecen 1) editar microservicio, en esta opción se puede cambiar el nombre y la descripción del microservicio; 2) eliminar microservicio y 3) Asociar historias de usuario al microservicio.

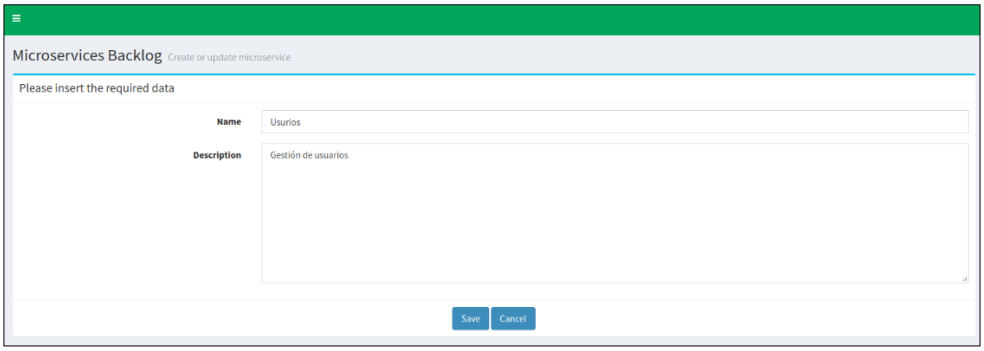


Figura 83. Crear microservicio.

Al seleccionar la opción de asociar historias de usuario al microservicio se presenta interfaz de la figura 84. En la parte superior aparecen el nombre y la descripción del microservicio; en la parte izquierda aparecen las historias de usuario disponibles para ser asociadas al microservicio y en la parte derecha las historias de usuario que ya han sido asignadas al microservicio, se puede agregar una historia (botón verde) o quitar una historia (botón rojo).

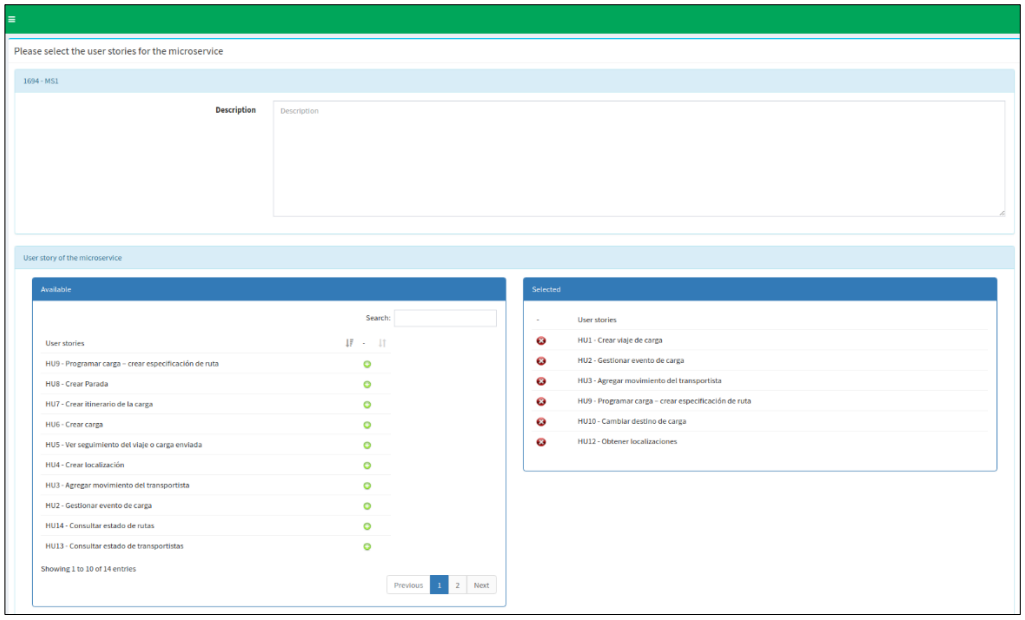


Figura 84. Asociar historias de usuario al microservicio.

Una vez ingresados los microservicios y asociadas sus historias de usuario se procede a calcular las métricas; al seleccionar la opción de calcular métricas el sistema automáticamente las calcula y presenta al usuario (ver figura 85).

En esa interfaz se pueden ver las métricas a nivel de aplicación o sistema (parte superior) y métricas específicas para cada microservicio (parte inferior). El usuario puede evaluar el acoplamiento, la cohesión, la complejidad de cada microservicio y de la aplicación, la granularidad, las llamadas y peticiones de cada microservicio y el promedio de la aplicación. El usuario obtiene información importante del diseño de la aplicación basada en microservicios para tomar decisiones.

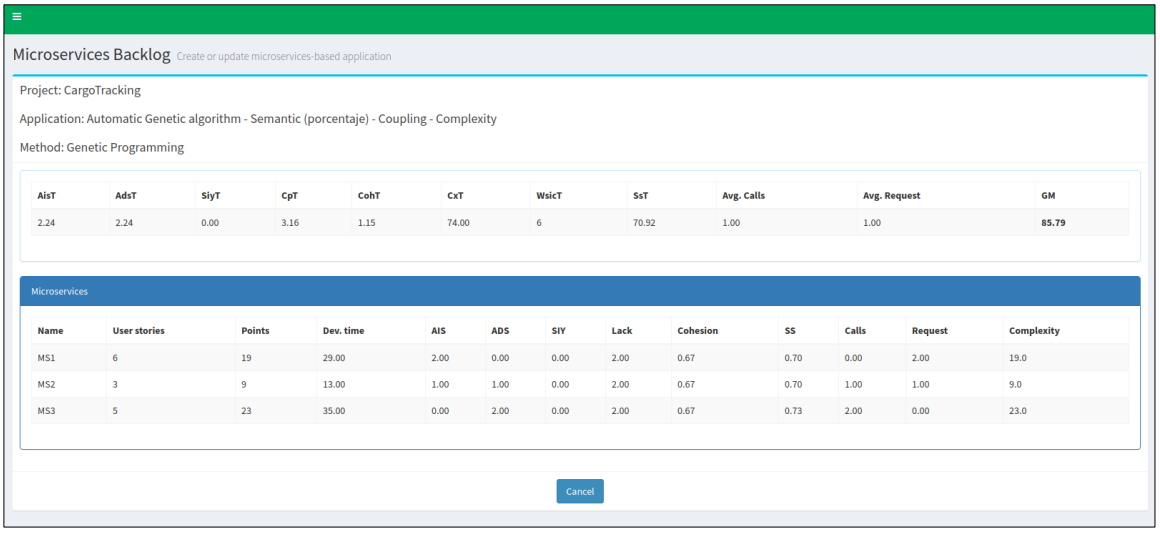


Figura 85. Métricas calculadas para la descomposición o aplicación basada en microservicios.

La opción 6 disponible en la interfaz de la figura 81 corresponde a la generación automática de la distribución de historias de usuario en los microservicios candidatos (descomposición) a partir de las dependencias definidas para las historias de usuario del proyecto. En la figura 86 se presentan los parámetros del algoritmo genético, los cuales son el tamaño de la población, el número de iteraciones, el número de hijos a obtener en cada iteración, el número de mutaciones a realizar en cada iteración y las variables que va a usar en la función objetivo (acoplamiento CpT, cohesión CohT, complejidad cognitiva CxT, número de historias asociadas a un microservicio WsicT y la similitud semántica SsT). Los detalles de implementación del algoritmo genético se explican en la sección 4.2.

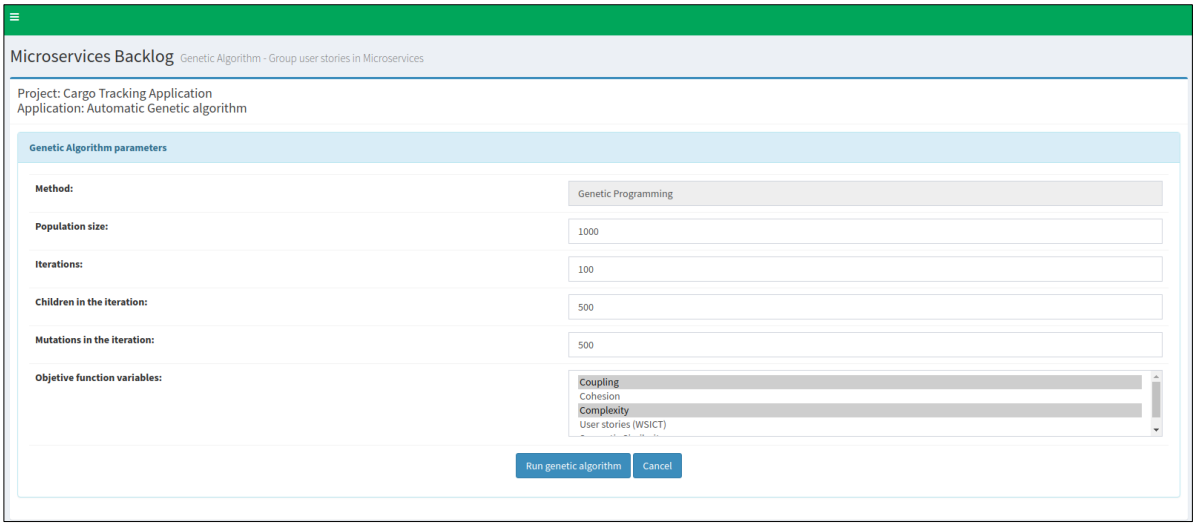


Figura 86. Parámetros del algoritmo genético.

Después de ingresar los parámetros se ejecuta el algoritmo genético, y después de unos minutos el sistema presenta la solución obtenida para ser evaluada por el usuario, si el usuario desea puede variar los parámetros y volver a ejecutar el algoritmo hasta obtener una solución optima y acorde a sus requerimientos. Los resultados obtenidos se presentan en la figura 87.

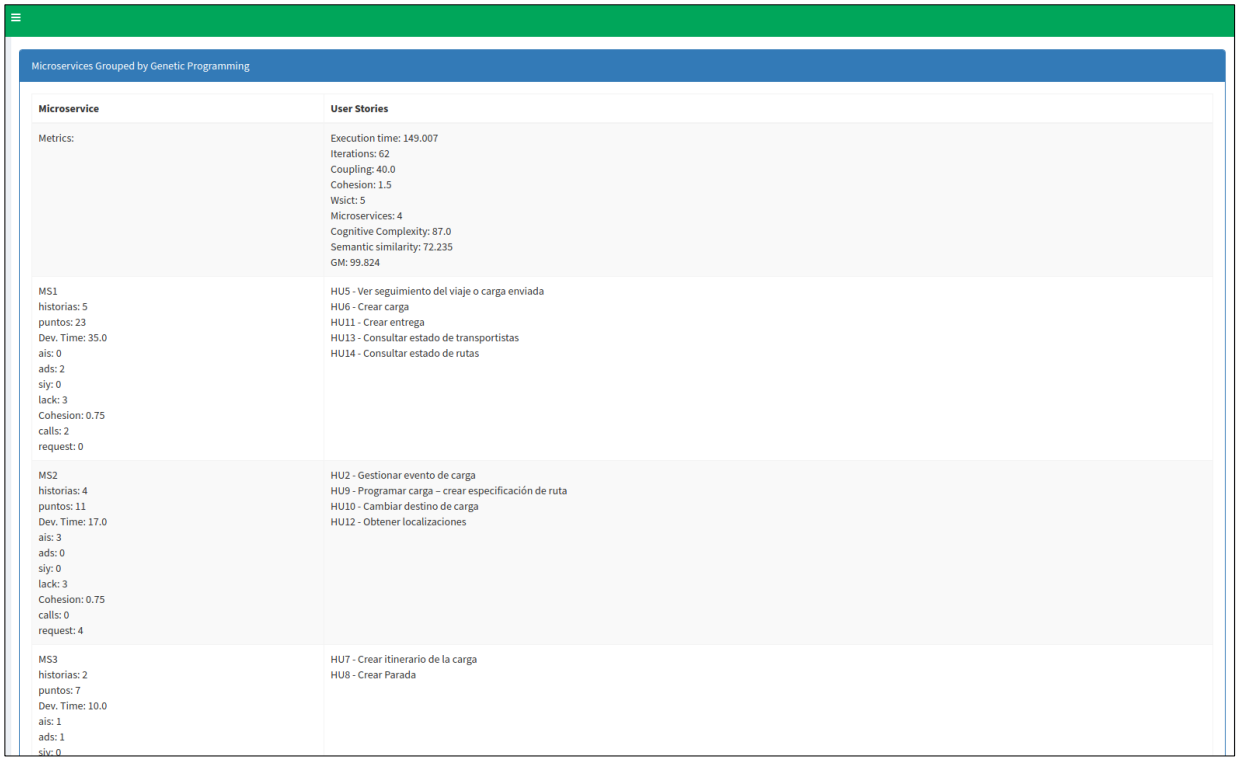


Figura 87. Resultados obtenidos con el algoritmo genético.

En los resultados se presenta la siguiente información: el tiempo de ejecución, el número de iteraciones usadas para obtener convergencia, el número de microservicios y los resultados de las métricas; también se presentan como quedaron asociadas las historias de usuario en los microservicios y las métricas específicas de cada microservicio.

Si los resultados son coherentes el sistema guarda automáticamente los resultados para poder ser comparados con otras opciones o métodos de descomposición.

La otra opción disponible para generar automáticamente los microservicios candidatos corresponde al algoritmo de agrupamiento por similitud semántica y distancia de acoplamiento entre los microservicios. La figura 88 presenta los parámetros que usa el algoritmo de agrupamiento para obtener la descomposición.

Este es un proceso iterativo donde el usuario puede ejecutar las veces que requiera tanto el agrupamiento por similitud semántica (ver figura 89) como el agrupamiento por las métricas de acoplamiento o distancia de acoplamiento (ver figura 90), puede variar los parámetros y validar los resultados, cuando encuentra una opción válida el sistema la guarda automáticamente para ser comparada con otros métodos.

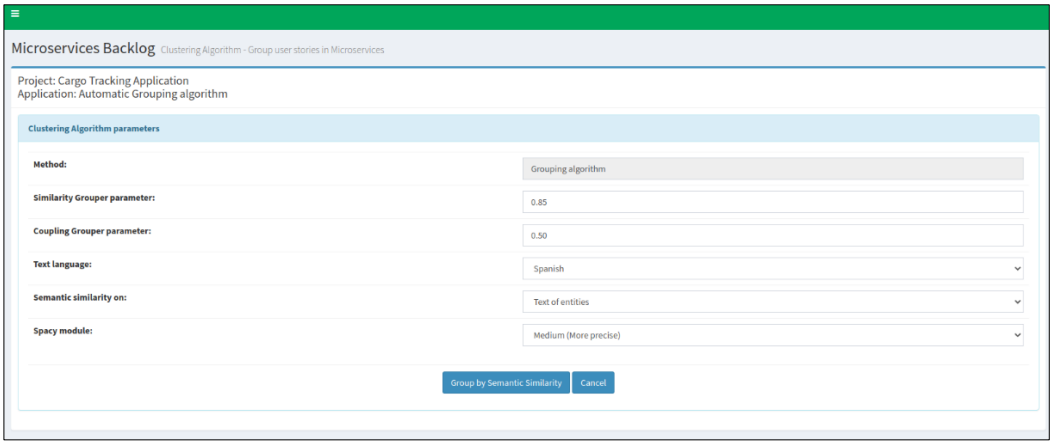


Figura 88. Parámetros del algoritmo de agrupamiento semántico.

Los parámetros corresponden a: 1) parámetro de agrupamiento semántico (el cual corresponde al valor mínimo de similitud semántica obtenido por Spacy al comparar las dos historias de usuario paras ser consideradas similares) 2) parámetro de agrupamiento por acoplamiento (valor de referencia de la distancia de acoplamiento, si el valor es mayor a 0.5 se considera que los microservicios comparados tienen un alto acoplamiento y deben ser unidos) 3) lenguaje del texto de la historia de usuario, puede ser español o inglés, 4) Aplicar similitud semántica sobre el texto de las entidades o sobre el lema de las entidades y 5) el módulo de Spacy a utilizar, con el modulo pequeño los resultados son más rápidos pero menos precisos, se recomienda usar siempre el modulo mediano.

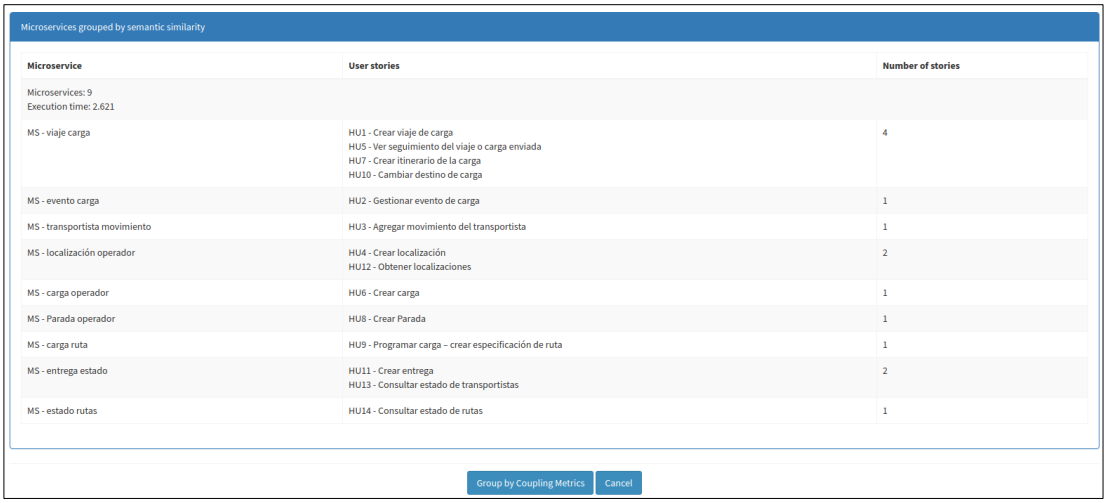


Figura 89. Resultados obtenidos con el algoritmo de agrupamiento por similitud semántica.

Primero el usuario obtiene los resultados de agrupar las historias que se refieren a la misma entidad, en el ejemplo de la figura 89, se definió el parámetro de agrupamiento semántico en el 75% (0.75), como resultado se agruparon las historias de usuario en 9 microservicios (ver figura 89); obtenida esta solución el usuario puede variar el parámetro de agrupamiento semántico para agrupar más historias o procede a agrupar por métricas de acoplamiento. Al agrupar por la distancia de agrupamiento se obtuvieron los microservicios presentados en la figura 90, con 5 microservicio y una similitud semántica del 75%. Una vez obtenida la solución el usuario debe calcular las métricas a través de la opción correspondiente disponible en el sistema.

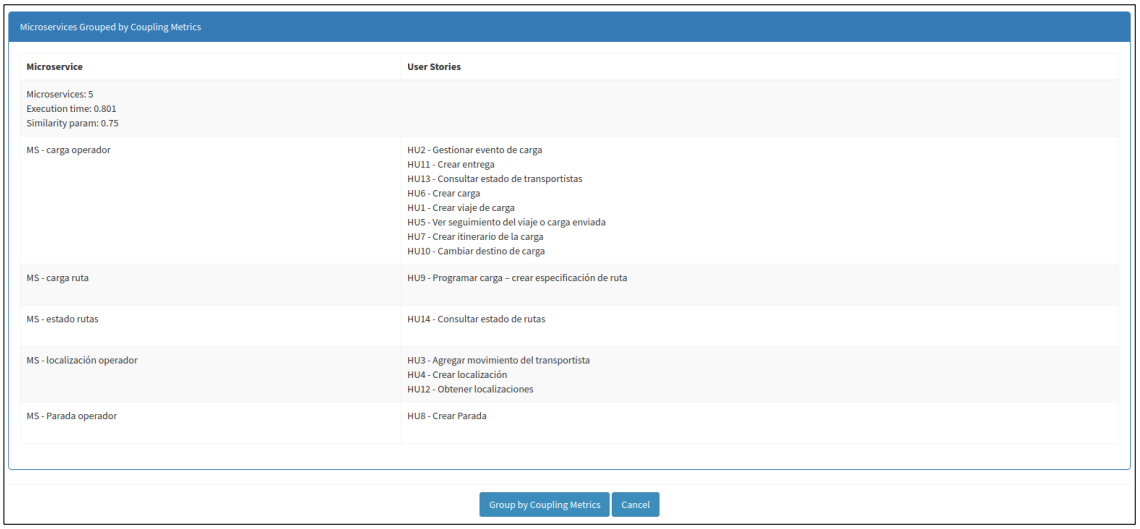


Figura 90. Resultados obtenidos con el algoritmo de agrupamiento por similitud semántica.

8.1.4. Diagrama Microservices Backlog

La opción 5 de la interfaz presentada en figura 81, corresponde al Microservices Backlog. Al ingresar el sistema presenta la interfaz de la figura 91.

Figura 91. Diagrama Microservices Backlog para la mejor solución (menor *Gm*)

El diagrama del Microservice Backlog corresponde al log de microservicios que se deben implementar, detallando sus dependencias y métricas tanto a nivel de aplicación como de cada microservicio, también en la parte inferior aparecen los detalles de historias de usuario asociadas a cada microservicio.

Se presenta el grafo de dependencias entre los microservicios que van a ser parte de la aplicación a desarrollar. Se puede apreciar que entre más grande es el nodo (microservicio) más complejo de implementar es (tiene mayor puntos de historia asociados) y requiere mayor tiempo de desarrollo; entonces el microservicio MS_1 es más complejo que los otros, el menos complejo es el microservicio MS_2.

El diagrama presentado corresponde a la mejor solución obtenida para el caso de estudio Cargo-Tracking.

8.1.5. Evaluar descomposiciones

Para comparar las descomposiciones obtenidas automáticamente por los algoritmos del sistema o ingresadas manualmente por el usuario, se tiene disponible la opción de evaluar métodos que aparece en la parte superior de la interfaz presentada en la figura 81. Al ingresar se presenta una lista desplegable de la cual se selecciona el proyecto a evaluar, el sistema presenta las soluciones candidatas junto con las métricas calculadas (ver figura 92), el usuario puede ordenar por cualquier métrica y seleccionar la que cumpla con sus requerimientos.

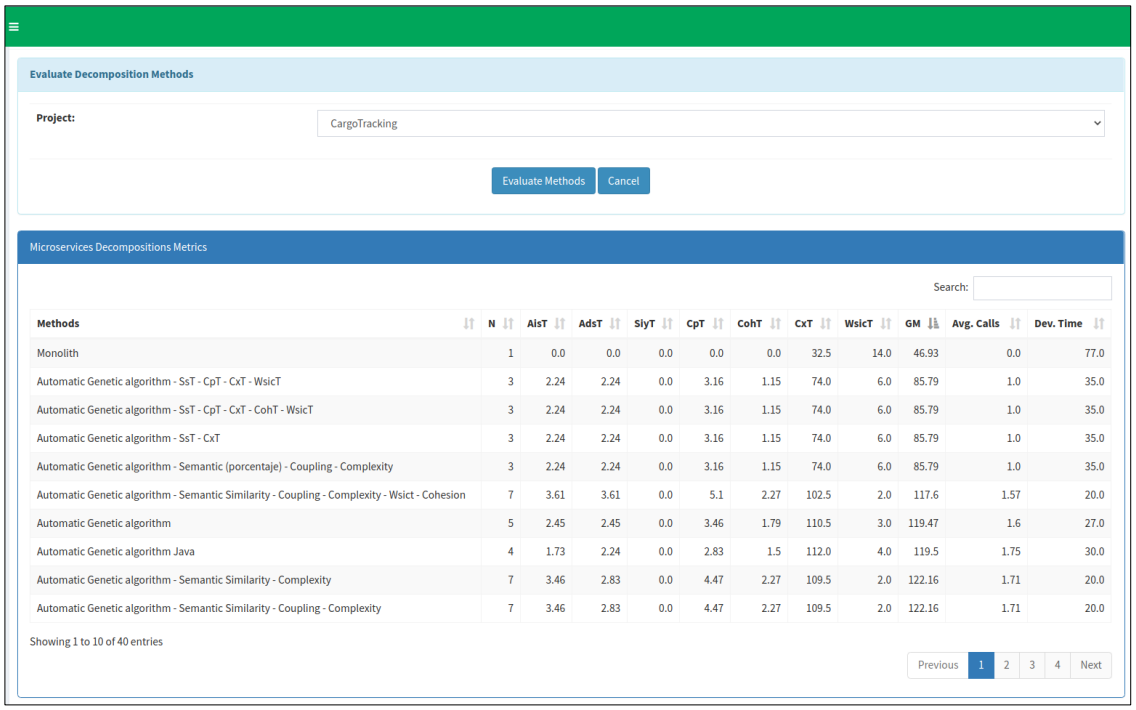


Figura 92. Evaluar métodos para el proyecto seleccionado

El usuario puede unir y dividir los microservicios a través de las funcionalidades disponibles en el sistema y evaluar de nuevo los resultados, de tal forma que se puedan obtener mejores soluciones o descomposiciones de las historias de usuario en microservicios.

8.1.6. Seleccionar la mejor solución

En interfaz presentada en la figura 92 el usuario puede ordenar por diferentes métricas y seleccionar la solución que cumpla y se adapte a sus requerimientos, para este caso de estudio

se seleccionó la solución que presenta menor *Gm* que corresponde a la descomposición obtenida por el algoritmo genético, esta solución se presenta en la figura 93.

Name	Number US.	User stories	Points	Dev. time
MS1	6	HU12 - Obtener localizaciones HU10 - Cambiar destino de carga HU9 - Programar carga – crear especificación de ruta HU3 - Agregar movimiento del transportista HU2 - Gestionar evento de carga HU1 - Crear viaje de carga	19	29.0
MS3	5	HU14 - Consultar estado de rutas HU13 - Consultar estado de transportistas HU11 - Crear entrega HU6 - Crear carga HU5 - Ver seguimiento del viaje o carga enviada	23	35.0
MS2	3	HU8 - Crear Parada HU7 - Crear itinerario de la carga HU4 - Crear localización	9	13.0

Figura 93. Mejor descomposición obtenida para el caso de estudio Cargo Tracking.

En este anexo se demostró la forma como se utiliza el sistema implementado para especificar la granularidad de los microservicios que forman una aplicación. Se evalúa su arquitectura en tiempo de diseño comparando métricas, que pueden dar una visión de cómo será implementado el sistema, analizar puntos críticos y su complejidad.