Behavior-Driven Microservice Architecture: un marco metodológico para la identificación iterativa de microservicios en proyectos ágiles greenfield

Behavior-Driven Microservice Architecture: A Methodological Framework for Iterative MicroserviceIdentification in Agile Greenfield Projects

Nicolás Battaglia
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina.

Gustavo Rossi
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina. LIFIA, Facultad de Informática, Universidad Nacional de La Plata (UNLP), Argentina.

Alejandro Fernández
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina.

Daniel Narváez
Universidad Abierta Interamericana (UAI), CAETI, Buenos Aires, Argentina.

DOI *https://doi.org/10.59471/raia2025227*
Enviado: junio 2025. **Aceptado**: octubre 2025. **Publicado:** diciembre 2025
Como citar: Battaglia, N., Rossi, G., Fernández , A., & Narváez, D. (2025). Behavior-Driven Microservice Architecture: un marco metodológico para la identificación iterativa de microservicios en proyectos ágiles greenfield. Revista Abierta De Informática Aplicada, 9(1). *https://doi.org/10.59471/raia2025227*

Resumen

La adopción de arquitecturas basadas en microservicios plantea desafíos significativos en la fase de diseño, particularmente en contextos greenfield donde las decisiones iniciales condicionan la mantenibilidad futura. Aunque existen aportes relevantes desde Domain-Driven Design (DDD) y Behavior-Driven Development (BDD), persiste una brecha metodológica: los enfoques actuales suelen ser teóricos, carecen de mecanismos explícitos de trazabilidad entre requisitos funcionales y decisiones arquitectónicas, o se enfocan en escenarios de reingeniería brownfield. Este trabajo introduce Behavior-Driven Microservice Architecture (BDMA), un marco metodológico sistemático, iterativo y reproducible que guía la identificación, diseño y evolución de microservicios en proyectos ágiles greenfield. BDMA integra principios de DDD, técnicas de BDD y prácticas de arquitectura evolutiva para transformar escenarios funciona¬les en bounded contexts, contratos de servicio y registros de decisiones arquitectónicas. Como aporte principal, BDMA ofrece un enfoque práctico que

asegura alineación entre requisitos y arquitectura, fomenta la colaboración interdisciplinaria y habilita trazabilidad completa desde los escenarios BDD hasta la implementación, demostrada mediante una Architectural Kata ilustrativa.

PALABRAS CLAVES: Microservicios, Arquitecturas de software, Metodos ágiles, Greenfield development.

Abstract

The adoption of microservice-based architectures poses significant challenges in the design phase, particularly in greenfield contexts where initial decisions condition future maintainability. Although there are relevant contributions from Domain-Driven Design (DDD) and Behavior-Driven Development (BDD), a methodological gap remains: current approaches tend to be theoretical, lack explicit mechanisms for traceability between functional requirements and architectural decisions, or focus on brownfield reengineering scenarios. This work introduces Behavior-Driven Microservice Architecture (BDMA), a systematic, iterative, and reproducible methodological framework that guides the identification, design, and evolution of microservices in agile greenfield projects. BDMA integrates DDD principles, BDD techniques, and evolutionary architecture practices to transform functional scenarios into bounded contexts, service contracts, and architectural decision records. As its main contribution, BDMA offers a practical approach that ensures alignment between requirements and architecture, fosters interdisciplinary collaboration, and enables complete traceability from BDD scenarios to implementation, demonstrated through an illustrative Architectural Kata.

KEYWORDS: Microservices, Software architectures, Agile methods, Greenfield development.

Introducción

El diseño arquitectónico constituye la fase donde se estructuran las decisiones de más alto nivel en el desarrollo de software, determinando la organización del sistema y su capacidad de evolución. En proyectos *greenfield*, este proceso plantea un desafío fundamental: la mayoría de las decisiones críticas se toman en etapas tempranas, cuando aún existe un alto grado de incertidumbre y una carencia de artefactos técnicos preexistentes (Cervantes & Kazman, 2024). A diferencia de la migración de sistemas monolíticos, donde el código fuente actúa como referencia tangible para la refactorización, el desarrollo desde cero exige derivar la arquitectura directamente de requisitos funcionales que a menudo son ambiguos o incompletos. Estas decisiones —que incluyen la definición de límites de servicios, contratos de integración y modelos de dominio— suelen ser difíciles de modificar una vez implementadas, debido a su impacto transversal en la evolución del sistema (Ford et al., 2022; Woods et al., 2021). Esta característica convierte a la arquitectura en un conjunto de “decisiones costosas de cambiar” que condicionan la mantenibilidad y la capacidad de adaptación futura.

En los últimos años, las arquitecturas basadas en microservicios se han consolidado como una alternativa predominante al paradigma monolítico, impulsadas por la necesidad de agilidad, escalabilidad, resiliencia y autonomía de despliegue en entornos basados en la nube. Su valor radica en dividir el sistema en servicios pequeños y autónomos, cada uno con un ciclo de vida independiente (Lewis & Fowler, 2014; Newman, 2021). Sin embargo, esta flexibilidad trae consigo retos mayores que las metodologías tradicionales no abordan completamente. La literatura reciente identifica problemas críticos como la correcta identificación de límites de servicio que asegure cohesión interna y baja dependencia, la gestión de la comunicación entre servicios y el mantenimiento de la consistencia de datos en entornos distribuidos (Narváez et al., 2025).

Cuando la delimitación de servicios se realiza de forma inadecuada, emergen síntomas recurrentes de degradación arquitectónica, conocidos como “bad smells”. Estos incluyen servicios demasiado granulares, responsabilidades superpuestas, dependencias cíclicas o la creación de “servicios dios”que acoplan excesivamente el sistema, comprometiendo la mantenibilidad y favoreciendo la acumulación temprana de deuda técnica (Ponce et al., 2022; Taibi & Lenarduzzi, 2018). Además, la descentralización de la gestión de datos introduce complejidades adicionales, como la necesidad de orquestar transacciones distribuidas y garantizar la consistencia eventual, aspectos que frecuentemente se subestiman en la fase de diseño inicial provocando fallos de integración tardíos.

La noción de arquitectura evolutiva enfatiza que, frente a entornos cambiantes, las arquitecturas deben diseñarse para sostener el cambio continuo mediante mecanismos de validación automatizados y la documentación sistemática de decisiones arquitectónicas (Ford et al., 2022). Esta perspectiva resalta la necesidad de marcos metodológicos que no solo guíen el diseño inicial, sino que también acompañen la evolución. En contextos ágiles, donde los requisitos evolucionan de manera iterativa y los equipos buscan entregar valor de forma incremental, esta rigidez inicial genera una tensión significativa. La literatura muestra que las prácticas tradicionales de estimación y diseño en entornos ágiles —como *story points* o *planning poker*— fueron concebidas para arquitecturas monolíticas y no se adaptan naturalmente a las particularidades de sistemas distribuidos, que requieren autonomía de servicios, comunicación asincrónica y límites semánticos explícitos (Ünlü et al., 2024).

Diversos enfoques han buscado mitigar este problema desde perspectivas teóricas y técnicas. El *Domain-Driven Design* (DDD) ha resaltado la relevancia de los *bounded contexts* como unidad para la separación de responsabilidades e identificación de microservicios (Evans, 2004; Vernon, 2016). Sin embargo, estudios empíricos y revisiones sistemáticas recientes evidencian que la aplicación directa de DDD enfrenta limitaciones prácticas, especialmente la falta de mecanismos prescriptivos para traducir conceptos abstractos en diseños concretos en contextos ágiles (Narváez et al., 2025; Zhong et al., 2024). Por su parte, el *Behavior-Driven Development* (BDD) ha enfatizado el valor de escenarios funcionales expresados en lenguaje ubicuo como medio para alinear negocio y técnica (Evans, 2004; Smart & Molak, 2023). No obstante, existe una brecha metodológica significativa: estos enfoques

suelen aplicarse de manera aislada, sin articular un proceso integral que permita transformar sistemáticamente requisitos funcionales textuales en decisiones arquitectónicas verificables y trazables (Rademacher et al., 2017). Aunque herramientas basadas en Inteligencia Artificial comienzan a emerger para asistir en la descomposición (Narváez et al., 2025), estas requieren de un marco metodológico subyacente que estructure los artefactos de entrada y salida para ser efectivas.

En este escenario, proponemos *Behavior-Driven Microservice Architecture* (BDMA), un marco metodológico que integra principios de DDD y BDD con prácticas de arquitectura evolutiva (Ford et al., 2022; Kopp et al., 2018). BDMA se orienta específicamente a proyectos *greenfield*, proporcionando un proceso iterativo, colaborativo y reproducible que acompaña la evolución funcional sin perder coherencia estructural. Su aporte principal radica en ofrecer un mecanismo sistemático para transformar escenarios BDD en artefactos arquitectónicos tangibles —mapas de agrupamiento funcional, *bounded contexts* explícitos, contratos API *contractfirst* y registros de decisiones (ADR)— que sostienen la alineación continua entre la visión de negocio y la implementación técnica.

Las contribuciones de este trabajo son:

- La definición de un marco metodológico que integra DDD, BDD y prácticas de arquitectura evolutiva en un proceso reproducible paso a paso.
- La propuesta de artefactos intermedios formalizados que facilitan la trazabilidad completa desde el requisito funcional hasta la interfaz del servicio y su validación continua.
- La presentación de un ejemplo ilustrativo y una validación preliminar que evidencian la aplicabilidad de BDMA para reducir la incertidumbre en fases tempranas de diseño.

El resto del artículo se estructura de la siguiente manera: La Sección 2 analiza los trabajos relacionados, clasificando los enfoques existentes y destacando la brecha metodológica. La Sección 3 detalla la metodología de investigación basada en Design Science Research (DSR). En la Sección 4, se presentan los fundamentos teóricos y principios rectores de BDMA. La Sección 5 describe en profundidad las cinco fases del marco y sus artefactos. La Sección 6 ilustra la aplicación del método mediante un caso de estudio de una plataforma científica. La Sección 7 presenta la validación empírica y el análisis de resultados. Finalmente, la Sección 9 ofrece las conclusiones y líneas de trabajo futuro.

Trabajos Relacionados

La identificación y diseño de microservicios constituye un problema de optimización complejo que ha sido abordado desde múltiples perspectivas en la última década. La literatura

actual puede clasificarse en tres corrientes principales: enfoques funcionales basados en el dominio, estrategias algorítmicas o asistidas por inteligencia artificial, y marcos de gobernanza para la arquitectura evolutiva. A pesar de los avances, persiste una fragmentación metodológica, particularmente en escenarios *greenfield*, donde la ausencia de código legado impide el uso de técnicas de refactorización tradicionales.

Enfoques de Descomposición Basados en el Dominio y Heurísticas

En el contexto de la ingeniería de software tradicional, diversos trabajos han buscado derivar microservicios a partir de descripciones funcionales de alto nivel y modelado del dominio.

(Josélyne et al., 2018) proponen una partición de microservicios basada en ingeniería de dominio y líneas de producto. Si bien ofrecen un marco estructurado para gestionar la variabilidad, su enfoque carece de mecanismos explícitos para la iteración ágil o la validación continua de los límites definidos.

La integración de *Domain-Driven Design* (DDD) y *Behavior-Driven Development* (BDD) ha sido explorada como una vía para alinear los requisitos con la arquitectura. (Hippchen et al., 2017) examinan la relación teórica entre DDD y BDD, resaltando su potencial sinérgico, pero sin formalizar un proceso metodológico operativo que guíe al arquitecto paso a paso. Por su parte, (Rademacher et al., 2017) introducen perfiles UML para representar *bounded contexts* y sus interacciones. Aunque aportan estandarización visual y formalización, su propuesta mantiene una orientación principalmente teórica y de difícil adopción en equipos ágiles que priorizan la documentación ligera.

Más recientemente, (Zhong et al., 2024) llevaron a cabo una investigación empírica extensiva sobre el uso de DDD en la industria. Sus conclusiones revelan que, si bien DDD es valioso para estructurar el dominio, su aplicación directa enfrenta tensiones metodológicas significativas en entornos ágiles, donde la falta de guías prescriptivas deriva a menudo en arquitecturas monolíticas distribuidas. Asimismo, (Cardoso, 2021) propone una guía de lineamientos heurísticos para proyectos *greenfield*, reconociendo la especificidad de diseñar desde cero. Sin embargo, al limitarse a principios generales sin artefactos verificables, su aplicabilidad depende excesivamente de la experiencia tácita del arquitecto.

Enfoques Algorítmicos y Asistidos por Inteligencia Artificial

Con el auge de la inteligencia artificial, han surgido herramientas que automatizan la descomposición de sistemas. En escenarios de migración (*brownfield*), herramientas como Mono2Micro (Kalia et al., 2021) utilizan técnicas de *clustering* temporoespacial sobre trazas de ejecución y análisis estático de código para recomendar particiones. Sin embargo, como se destaca en revisiones sistemáticas recientes (Narváez et al., 2025), estos enfoques son inaplicables en proyectos *greenfield* debido a la inexistencia de artefactos de código o logs de ejecución previos.

Para nuevos desarrollos, la literatura propone el análisis de requisitos textuales mediante Procesamiento de Lenguaje Natural (NLP). (Bajaj et al., 2022) presentan *GreenMicro*, un enfoque que utiliza casos de uso UML y entidades de base de datos para identificar servicios mediante algoritmos de agrupamiento. De manera similar, (Vera-Rivera et al., 2020) y su herramienta *SEMGROMI* (Vera-Rivera et al., 2023) emplean algoritmos de similitud semántica sobre historias de usuario para proponer agrupaciones de microservicios. Más recientemente, enfoques basados en *Deep Learning* como *GTMicro* (Bajaj et al., 2024) utilizan modelos Transformers (BERT) para mejorar la precisión de estas recomendaciones.

A pesar de su sofisticación, estos métodos algorítmicos presentan limitaciones críticas para la práctica industrial: (1) operan a menudo como “cajas negras”, careciendo de validación semántica humana; (2) dependen de la calidad y completitud de los requisitos textuales, que suelen ser ambiguos en etapas tempranas; y (3) no integran explícitamente la negociación de contratos de interfaz (*APIs*) como parte del proceso de diseño, un aspecto vital para la mantenibilidad.

Arquitectura Evolutiva y Gobernanza

El problema de la evolución arquitectónica ha sido abordado desde perspectivas de adaptación y documentación continua. (Ford et al., 2022) introducen el concepto de *fitness functions* como mecanismos para validar automáticamente que la arquitectura cumpla con restricciones arquitectónicas a lo largo del tiempo. (Woods et al., 2021) amplían esta noción bajo el paradigma de *Continuous Architecture*, enfatizando la necesidad de mantener la trazabilidad entre decisiones técnicas y objetivos de negocio.

Para la gobernanza de decisiones, (Kopp et al., 2018) formalizan los *Architectural Decision Records* (ADR) como técnica estándar para registrar decisiones de diseño de forma versionable junto al código. En el plano de la implementación, (Zimmermann et al., 2022) sistematizan patrones de diseño de APIs *contract-first*. Estos aportes son fundamentales para la construcción de interfaces verificables, pero la literatura actual tiende a tratarlos como actividades aisladas, desconectadas del proceso de descubrimiento de servicios basado en requisitos.

Análisis de Brechas y Posicionamiento

Estudios secundarios recientes corroboran la fragmentación del campo. (Schmidt & Thiry, 2020) evidencian la falta de consenso metodológico, mientras que (Ünlü et al., 2024) concluyen que las prácticas de diseño en equipos ágiles continúan ancladas en supuestos monolíticos por falta de herramientas adecuadas.

La Tabla 1 resume el análisis comparativo de los enfoques discutidos frente a la propuesta de este trabajo. Se observa que, mientras existen soluciones robustas para migración y propuestas teóricas para *greenfield*, carecemos de marcos metodológicos que operacionalicen

la trazabilidad completa desde la especificación del comportamiento (BDD) hasta la definición de contratos y gobernanza, integrando validación humana en el ciclo. BDMA busca cubrir esta brecha específica.

TABLA 1: Comparativa de enfoques para el diseño de microservicios

Enfoque	Escenario	Técnica Dominante	Limitación Principal
Mono2Micro (Kalia et al., 2021)	*Brownfield*	*Clustering de trazas y análisis estático*	*Inaplicable sin código existente.*
GreenMicro (Bajaj et al., 2022)	*Greenfield*	*Clustering de Casos Uso y de Entidades*	*Enfoque rígido; falta validación semántica experta.*
SEMGROMI (Vera-Rivera et al., 2023)	*Greenfield*	*NLP y similitud semántica en Historias de Usuario*	*No aborda la definición de contratos API ni gobernanza.*
DDD Puro (Evans, 2004)	*Agnóstico*	*Modelado estratégico manual*	*Falta de prescripción metodológica en Agile.*
BDMA	**Greenfield**	**Integración BDD-DDD y Arquitectura Evolutiva**	**Requiere adopción de BDD; enfoque estructural.**

Metodología: Enfoque de Design Science Research

El enfoque metodológico adoptado en esta investigación se basa en *Design Science Research* (DSR), un marco fundamental en Ingeniería de Software para abordar problemas que requieren soluciones prescriptivas e innovadoras (Hevner et al., 2004; Peffers et al., 2007; Wieringa, 2014). La elección de DSR se justifica por la naturaleza del problema: la identificación de microservicios en proyectos *greenfield* presenta desafíos de incertidumbre que no pueden resolverse únicamente mediante modelos descriptivos, sino que exigen la creación de un artefacto de tipo *método* o *proceso* (March & Smith, 1995) que integre rigor científico y utilidad práctica.

Siguiendo el modelo de ciclos de Hevner (Hevner, 2007) y las extensiones propuestas por Gregor y Hevner (Gregor & Hevner, 2013), la investigación se estructuró en torno a tres ciclos interdependientes, complementados por una fase de consolidación empírica (ver Figura 1).

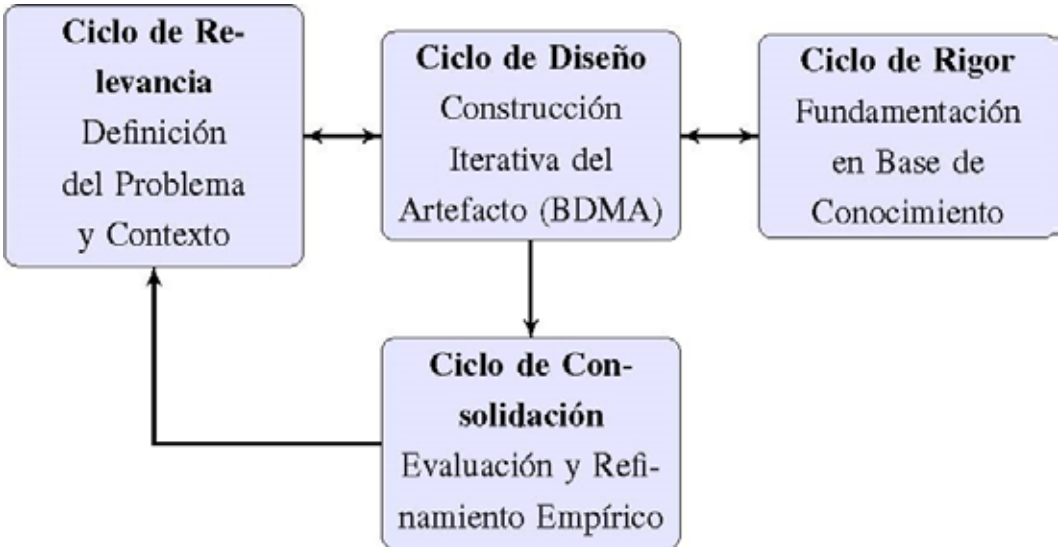


Figura 1: *Ciclos de investigación DSR con extensión de consolidación.*

Ciclo de Relevancia

Este ciclo conecta la investigación con el entorno real para asegurar que el artefacto responda a una necesidad genuina. Se ejecutaron actividades de diagnóstico que incluyeron un mapeo sistemático de la literatura (Battaglia et al., 2024) y el análisis de la práctica industrial. Se identificó que, en entornos ágiles, los equipos carecen de guías sistemáticas para transitar desde historias de usuario hacia diseños distribuidos sin incurrir en deuda técnica temprana.

Ciclo de Rigor

El ciclo de rigor garantiza el sustento teórico de la propuesta mediante la conexión con la base de conocimiento existente. El diseño de BDMA no es *ad-hoc*, sino que se fundamenta en la integración operativa de teorías consolidadas: los patrones estratégicos de *Domain-Driven Design* (DDD), la especificación comportamental de *Behavior-Driven Development* (BDD) y los principios de Arquitectura Evolutiva. Este anclaje teórico asegura la validez conceptual de los artefactos propuestos.

Ciclo de Diseño

Constituye el núcleo constructivo de la investigación. En este ciclo se desarrolló el artefacto BDMA de manera iterativa, definiendo sus cinco fases, las reglas de transformación de requisitos y los mecanismos de documentación (ADRs). El artefacto fue refinado mediante iteraciones de diseño lógico antes de su validación externa.

Ciclo de Consolidación

Siguiendo los lineamientos contemporáneos de (Pries-Heje et al., 2008), se incorporó un ciclo de consolidación explícito para profundizar la validación del artefacto más allá de su construcción. Esta fase se centró en la evaluación empírica mediante experimentación

controlada (detallada en la Sección 7), permitiendo contrastar la efectividad de la propuesta frente a enfoques no estructurados y analizar su utilidad percibida por los usuarios en escenarios de aprendizaje.

Fundamentos de BDMA y Principios de Diseño

Behavior-Driven Microservice Architecture (BDMA) se define como un marco metodológico iterativo y reproducible diseñado para sistematizar la identificación, diseño y evolución de microservicios en proyectos ágiles *greenfield*. A diferencia de los enfoques heurísticos o puramente algorítmicos, BDMA no trata la arquitectura como un evento único de planificación (*Big Design Up Front*), sino como un flujo continuo que transforma requisitos narrativos en decisiones estructurales verificables.

Principios Rectores

El marco se fundamenta en cuatro pilares teóricos que garantizan su coherencia metodológica y su aplicabilidad en contextos de alta incertidumbre:

1. **Trazabilidad Bidireccional:** BDMA promueve una trazabilidad total, asegurando que cada microservicio, contrato de interfaz y decisión arquitectónica pueda rastrearse explícitamente hasta uno o más escenarios BDD validados (Smart & Molak, 2023). Esto mitiga la erosión arquitectónica al vincular la solución técnica directamente con el valor de negocio.

2. **Arquitectura Evolutiva:** Basándose en los postulados de (Ford et al., 2022), el marco asume que la estructura del sistema debe adaptarse iterativamente. Se rechaza la rigidez de los modelos estáticos en favor de una arquitectura que evoluciona en sincronía con los cambios funcionales, apoyada por funciones de aptitud (*fitness functions*) y validación continua.

3. **Diseño Colaborativo y Lenguaje Ubicuo:** El enfoque privilegia la eliminación de silos entre expertos del dominio y equipos técnicos. Al utilizar escenarios Gherkin como *lingua franca*, se refuerza la comprensión compartida y se validan las decisiones de diseño mediante técnicas colaborativas (Evans, 2004; Vernon, 2016).

4. **Gobernanza Ligera y Evidencia Funcional:** BDMA incorpora una gobernanza basada en el registro sistemático de decisiones mediante *Architectural Decision Records* (ADR) (Kopp et al., 2018). Asimismo, utiliza las pruebas BDD no solo como validación de QA, sino como mecanismo activo de verificación de la integridad arquitectónica (“Spec-as-Test”).

Estructura Procedimental del Marco

El proceso BDMA se organiza en cinco fases secuenciales e iterativas. Cada fase posee entradas definidas, actividades de transformación y artefactos de salida que alimentan la

siguiente etapa, garantizando un flujo de trabajo reproducible en cada *sprint* ágil (ver Tabla 7).

Fase 1: Clasificación Funcional de Escenarios

Objetivo: Esta fase inicial organiza el backlog de escenarios BDD desestructurados en agrupaciones coherentes que representan Epics o procesos de negocio de alto nivel. Proceso: Se analizan los escenarios redactados en lenguaje Gherkin (Given-When-Then) para identificar afinidades semánticas. Esta actividad se fundamenta en los principios de Continuous Architecture (Woods et al., 2021), donde las decisiones iniciales buscan alinear la estructura del software con los flujos de valor del negocio.

TABLA 2: Artefactos de la Fase 1

Entradas	Salidas
- Backlog funcional en Gherkin	*- Mapa de agrupamiento funcional.*
- Metainformación contextual (Tags, Epics).	*- Tabla de trazabilidad preliminar (Epic → Escenarios).*

Fase 2: Extracción Colaborativa de Elementos del Dominio

Objetivo: Transformar la narrativa de los escenarios agrupados en elementos técnicos tangibles. Se analizan las sentencias Gherkin para extraer *Actores*, *Comandos* (intenciones del usuario) y *Eventos* (resultados observables). Proceso: El conocimiento del dominio provisto por expertos complementa la interpretación sintáctica. Se utiliza BDD como mecanismo de descripción de comportamiento (Smart & Molak, 2023) para identificar entidades candidatas y sus ciclos de vida.

TABLA 3: Artefactos de la Fase 2

Entradas	Salidas
- Mapa de agrupamiento funcional (F1).	*- Mapa de elementos del dominio (Comandos, Eventos, Entidades).*
- Conocimiento tácito de expertos.	*- Matriz de relación Escenario → Elemento.*

Fase 3: Delimitación y Formalización de Bounded Contexts Objetivo:

Objetivo: Consolidar los elementos dispersos en *Bounded Contexts* coherentes, definiendo explícitamente sus límites y las relaciones estratégicas entre ellos.

Proceso: Se emplean técnicas colaborativas como *Event Storming* para visualizar flujos. Se aplican patrones estratégicos de DDD para definir la relación entre contextos: *Partnership* (colaboración estrecha), *Conformist* (adhesión estricta) o *Anti-Corruption Layer* (ACL) para aislamiento defensivo (Vernon, 2016).

TABLA 4: Artefactos de la Fase 3

Entradas	Salidas
- Mapa de elementos del dominio (F2).	*- Mapa de Contextos (Context Map) formalizado.*
- Restricciones organizacionales.	*- Declaración de reglas de interacción (ACL,OHS).*

Fase 4: Derivación de Interfaces y Contratos API

Objetivo: Traducir los comandos y eventos de cada contexto en contratos de servicio (*APIs*) verificables, siguiendo un enfoque *Contract-First*.

Proceso: Se fundamenta en patrones de diseño de APIs (Zimmermann et al., 2022). Cada interacción definida en el Mapa de Contextos se especifica como una interfaz técnica (ej. OpenAPI/AsyncAPI), asegurando que la implementación respete la semántica del negocio.

TABLA 5: Artefactos de la Fase 4

Entradas	Salidas
- Mapa de Bounded Contexts (F3).	*- Especificaciones de Contratos API.*
- Escenarios BDD originales.	*- Pruebas de contrato automatizadas.*

Fase 5: Validación Continua y Evolución Controlada

Objetivo: Garantizar la consistencia arquitectónica a lo largo del tiempo, registrando cambios y verificando la alineación con los requisitos.

Proceso: Se utiliza el registro de ADRs (Kopp et al., 2018) para documentar el “por qué”de cada cambio estructural. Las pruebas BDD se ejecutan contra los contratos derivados para validar que la arquitectura sigue soportando los escenarios de negocio.

TABLA 6: Artefactos de la Fase 5

Entradas	Salidas
- Contratos API (F4).	*- Bitácora de ADRs (ADR Log).*
- Escenarios BDD actualizados.	*- Context Map versionado.*

TABLA 7: Resumen del Marco BDMA y Fundamentación Teórica

Fase	Concepto Clave	Fundamentación Principal
F1	*Concepto Clave* *Arquitectura alineada a Features/Epics*	*Fundamentación Principal* *Evolutionary Arch. (Ford et al., 2022), Continuous Arch. (Woods et al., 2021)*
F2	*BDD para extracción de elementos de dominio*	*BDD in Action (Smart & Molak, 2023), Relación BDD-DDD (Hippchen et al., 2017)*
F3	*Bounded Contexts y Event Storming*	*DDD (Evans, 2004), Event Storming (Brandolini, 2013)*
F4	*Diseño de APIs Contract-First*	*API Patterns (Zimmermann et al., 2022)*
F5	*ADRs y Validación Continua*	*Markdown ADRs (Kopp et al., 2018)*

Fases del Proceso Metodológico

La ejecución de BDMA se estructura como un flujo secuencial pero iterativo, diseñado para reducir progresivamente la incertidumbre arquitectónica. Antes de describir las fases, es crucial comprender que la delimitación de microservicios no se restringe a trazar fronteras internas, sino que requiere explicitar la gobernanza de las interacciones. Siguiendo los patrones estratégicos de *Domain-Driven Design* (Evans, 2004; Vernon, 2016), BDMA clasifica las relaciones entre contextos para prevenir el acoplamiento accidental.

Durante el proceso, se identifican y formalizan tres tipos de relaciones críticas:

1. **Partnership (Asociación):** Dos contextos colaboran de manera simétrica y estrecha. El fallo en uno impacta inmediatamente al otro, lo que requiere una coordinación sincrónica entre equipos.

2. **Conformist (Conformista):** Un contexto consumidor se adhiere estrictamente al modelo de datos del proveedor. Aunque simplifica la integración, subordina la evolución del consumidor a las decisiones del proveedor.

3. **Anticorruption Layer (ACL):** Un contexto introduce una capa de traducción defensiva para aislar su modelo de dominio interno de modelos externos inestables o legados. Este patrón es vital en arquitecturas evolutivas para mantener la pureza del diseño (Ford et al., 2022).

A continuación, se detallan las cinco fases del marco, describiendo la mecánica de transformación de artefactos en cada etapa.

Fase 1: Clasificación Funcional de Escenarios

Esta fase organiza el *backlog* de escenarios BDD validados en agrupaciones coherentes que representan *Epics* o procesos de negocio transversales. El objetivo es superar

la visión fragmentada de las historias de usuario individuales para identificar “zonas de afinidad”funcional (Erder & Pureur, 2015; Ford et al., 2022).

Las entradas son los escenarios redactados en lenguaje Gherkin (*Given-When-Then*). El proceso de clasificación no es meramente sintáctico; implica analizar la metainformación con-textual para detectar qué escenarios comparten disparadores o estados finales comunes. La salida principal es el **Mapa de Agrupamiento Funcional,** un artefacto de referencia que actúa como hipótesis inicial de los límites del sistema, alineando las decisiones técnicas tempranas con los objetivos macro del negocio.

Fase 2: Extracción Colaborativa de Elementos del Dominio

Una vez agrupados los escenarios, se procede a la disección técnica de la narrativa. En esta fase, los escenarios dejan de tratarse solo como requisitos de prueba y pasan a ser fuentes de minería de conocimiento (Smart & Molak, 2023). Se aplica un análisis semántico —asistido por expertos del dominio— para mapear las sentencias Gherkin a elementos tácticos de DDD:

- Las sentencias *When* (acciones) se transforman en candidatos a Comandos.
- Las sentencias *Then* (resultados) se identifican como Eventos de Dominio.
- Los sustantivos recurrentes en *Given* emergen como candidatos a Entidades o Agregados.

Este proceso genera el **Mapa de Elementos del Dominio**, un artefacto que aporta trazabilidad explícita: cada elemento técnico (ej. un tópico de Kafka o un endpoint REST) tiene un origen directo en una sentencia de comportamiento validada (Rademacher et al., 2017).

Fase 3: Delimitación y Formalización de Bounded Contexts

Tomando como insumo los mapas de elementos, esta fase consolida los límites arquitectónicos. Aquí se aplica el principio de *alta cohesión y bajo acoplamiento* para agrupar comandos y eventos en *Bounded Contexts* definitivos.

Se utiliza la técnica de *Event Storming* (Brandolini, 2013) para visualizar el flujo de eventos a través de los grupos funcionales. El resultado es el Context Map, que no solo define qué funcionalidad pertenece a qué servicio, sino cómo interactúan (Sincrónico vs. Asincrónico) y bajo qué patrón de gobernanza (Partnership, ACL, etc.) (Vernon, 2013). La formalización en esta etapa es crítica para detectar dependencias cíclicas antes de escribir código.

Fase 4: Derivación de Interfaces y Contratos API

Con los límites definidos, se procede a la especificación técnica de las fronteras. Los comandos y eventos identificados se transforman en contratos de interfaz (*APIs*) siguiendo un enfoque *Contract-First* (Zimmermann et al., 2019).

- Los comandos públicos se traducen en especificaciones de API (ej. OpenAPI).

- Los eventos de dominio se formalizan en esquemas de mensajería (ej. AsyncAPI/ Avro).

Esta fase asegura que la implementación técnica respete la semántica del negocio. Los contratos generados sirven doble propósito: documentan la interfaz para los consumidores y actúan como .[es]pecificación ejecutable"para las pruebas automatizadas en la arquitectura evolutiva (Ford et al., 2022).

Fase 5: Validación Continua y Evolución Controlada

La arquitectura de microservicios no es estática; debe evolucionar. Esta fase institucionaliza la gobernanza mediante *Architectural Decision Records* (ADR) (Kopp et al., 2018). Cada vez que un escenario BDD cambia o se añade, se evalúa su impacto en los contratos existentes.

Las salidas de esta fase incluyen bitácoras de decisiones inmutables y versiones actualizadas del *Context Map*. Esto habilita una auditoría completa del sistema: ante un cambio en la implementación, es posible rastrear hacia atrás hasta el ADR que motivó el cambio, el contrato afectado y el escenario BDD original, cerrando el ciclo de trazabilidad metodológica (Erder & Pureur, 2015).

Ejemplo de Aplicación: Plataforma de Gestión Científica

Para validar la aplicabilidad del marco BDMA en un escenario de complejidad realista, se seleccionó como caso de estudio el diseño de una **Plataforma Distribuida para la Gestión de Publicaciones Científicas**. Este dominio fue elegido por presentar desafíos arquitectónicos típicos de sistemas distribuidos: flujos de trabajo asincrónicos, múltiples actores con permisos granulares (autores, revisores, editores), reglas de negocio cambiantes (políticas editoriales) y la necesidad de integración con ecosistemas externos.

Descripción del Escenario y Requisitos

La organización requiere un sistema que gestione el ciclo de vida completo de una publicación, desde el envío del manuscrito hasta su aceptación final y certificación. Los requisitos funcionales, capturados inicialmente como Epics, incluyen:

- Gestión de Identidad Federada: Registro de investigadores y autenticación segura, con soporte para perfiles enriquecidos mediante ORCID.

- Core Editorial: Soporte para envíos versionados, control estricto de *deadlines* y gestión de conflictos de interés en la asignación de revisores.

- Procesos de Revisión: Manejo de flujos de revisión ciega (simple o doble) y emisión de dictámenes editoriales basados en quórum.

- Integración y Métricas: Sincronización bidireccional con bases de datos externas (Scopus, Google Scholar) y generación de reportes de impacto.

Ejecución del Proceso BDMA

Aplicando las fases del marco, se procesó un *backlog* de escenarios BDD para derivar la estructura de servicios. La Tabla 8 presenta una muestra representativa de la trazabilidad generada durante la Fase 3 y 4, vinculando requisitos funcionales con decisiones estructurales y eventos de dominio.

TABLA 8: Matriz de Trazabilidad: Escenarios BDD a Elementos de Arquitectura

Feature / Epic	Escenario BDD (Resumido)	Bounded Context	Eventos/Comandos Clave
Registro y Autenticación	*Registro exitoso con ORCID y verificación de email*	*Identity & Access*	*UsuarioRegistrado, EmailConfirmado*
Gestión de Perfil	*Importar métricas de impacto desde fuentes externas*	*Profiles External Integrations*	*MetricasSincronizadas, PerfilActualizado*
Envío de Artículos	*Envío válido a evento antes del deadline*	*Submissions*	*EnvioRegistrado, ComprobanteGenerado*
Envío de Artículos	*Rechazo automático por envío fuera de fecha*	*Submissions*	*EnvioRechazado, DeadlineExpirado*
Asignación de Revi¬sores	*Asignación automática respetando conflictos de interés*	*Reviewing*	*RevisoresAsignados, ConflictoDetectado*
Decisión Editorial	*Emisión de decisión basada en reglas y quórum*	*Editorial Decisioning*	*DecisionEmitida, CartaGenerada*
Integraciones Externas	*Validación de DOI y metadatos al registrar envío*	*External Integrations*	*DOIValidado, MetadatosCompletos*

Arquitectura Emergente y Mapa de Contextos

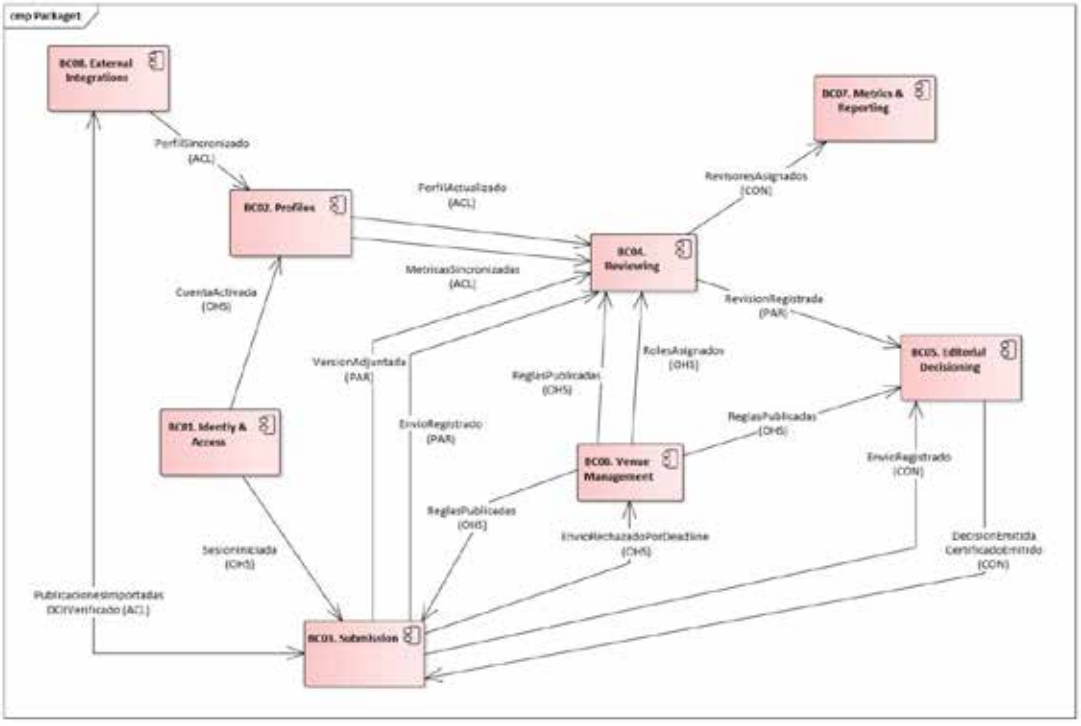
Como resultado de la **Fase 3 (Delimitación)**, se identificaron ocho *Bounded Contexts* con responsabilidades segregadas: *Identity & Access*, *Profiles*, *Submissions*, *Reviewing*, *Editorial Decisioning*, *Venue Management*, *Metrics & Reporting* y *External Integrations*.

La Figura 2 ilustra el Mapa de Contexto derivado. Es crucial notar cómo BDMA prescribió los patrones de interacción para proteger la integridad del dominio:

1. **Anticorruption Layer (ACL) en Integraciones:** El contexto *Profiles* consume datos de *External Integrations* mediante una ACL. Esto aísla el modelo interno de perfiles de los cambios frecuentes en las APIs de terceros (ORCID, Scopus), garantizando estabilidad evolutiva (Ford et al., 2022).

2. **Open Host Service (OHS) en Gestión de Sedes:** *Venue Management* expone sus reglas (fechas, políticas) como un servicio público estandarizado (*ReglasPublicadas*), permitiendo que *Submissions* y *Reviewing* consuman estas reglas sin acoplarse a la lógica interna de configuración de eventos.

3. **Partnership (PAR) en el Núcleo:** Se estableció una relación de asociación entre *Submissions* y *Reviewing*. Dado que el ciclo de vida de una revisión depende intrínsecamente de la versión del manuscrito enviado, ambos equipos deben coordinar estrechamente sus cambios de esquema para mantener la consistencia del flujo EnvioRegistrado → RevisionIniciada.



Fgura 2: *Mapa de contexto resultante de la primera iteración BDMA, explicitando patrones estratégicos (ACL, OHS, PAR) para gobernar las interacciones entre microservicios.*

La combinación de la matriz de trazabilidad (Tabla 8) y el mapa estratégico (Figura 2) evidencia que BDMA no solo identifica çajas"de servicios, sino que define la semántica de sus relaciones, produciendo una arquitectura verificable y coherente con los requisitos funcionales desde la primera iteración.

Validación Empírica

Para evaluar la efectividad y aplicabilidad del marco BDMA, se diseñó un estudio empírico de carácter exploratorio en un contexto académico controlado. El objetivo principal fue

medir el impacto del uso de un método estructurado frente a enfoques *ad-hoc* en la calidad percibida del diseño arquitectónico y la comprensión del dominio.

Participantes y Demografía

El estudio contó con la participación de N=28 profesionales matriculados en un programa de posgrado en Arquitectura de Software. La muestra presentó una heterogeneidad relevante para el estudio:

- **Perfil Profesional:** Predominancia de desarrolladores senior y líderes técnicos de la industria TI.

- **Experiencia Previa:** Mediante una encuesta de caracterización (escala 0–5), se identificó un nivel alto en metodologías ágiles ($\mu = 4.2$) y patrones de diseño ($\mu = 3.9$), pero una experiencia limitada en BDD ($\mu = 1.8$) y diseño de microservicios distribuidos ($\mu = 2.1$).

Esta configuración demográfica es representativa de equipos de transición en la industria: profesionales competentes en monolitos que enfrentan la curva de aprendizaje de sistemas distribuidos.

Diseño Experimental

Se empleó un diseño cuasi-experimental intra-sujeto con medidas repetidas. Los participantes se organizaron en equipos y abordaron el mismo problema de diseño (la plataforma de gestión científica descrita en la Sección 6) en dos fases consecutivas:

1. Fase de Control (Sin BDMA): Los equipos diseñaron la solución utilizando su conocimiento previo y herramientas estándar de DDD (diseño táctico libre), sin una guía procedimental específica.

2. Fase Experimental (Con BDMA): Se introdujo el marco metodológico. Los equipos refactorizaron su solución aplicando secuencialmente las cinco fases de BDMA, generando los artefactos prescritos (Mapas de Agrupamiento, Tablas de Elementos, Contratos API y ADRs).

Instrumentos de Recolección de Datos

Tras completar la fase experimental, se administró un cuestionario de percepción basado en el Modelo de Aceptación Tecnológica (TAM), utilizando una escala Likert de 5 puntos para evaluar dimensiones de utilidad, facilidad de uso y mejora en la calidad del diseño. Adicionalmente, se recolectaron respuestas abiertas para análisis cualitativo.

Análisis de Resultados y Discusión

En esta sección se presentan e interpretan los hallazgos derivados del estudio empírico realizado con los 28 profesionales participantes. El análisis se ha estructurado para triangular la evidencia desde múltiples perspectivas, permitiendo una evaluación holística del impacto del marco BDMA en el proceso de diseño. A continuación, se desglosan los resultados en dos dimensiones principales: una evaluación cuantitativa basada en las métricas de percepción de utilidad y facilidad de uso, y un análisis cualitativo de la retroalimentación abierta, concluyendo con una discusión crítica sobre las implicaciones y las amenazas a la validez del estudio.

Resultados Cuantitativos

El análisis estadístico descriptivo de las respuestas (Tabla 9) revela una aceptación altamente positiva del marco.

TABLA 9: Resultados de la encuesta de validación (N=28)

Dimensión Evaluada	Media	DE	Min	Max
Comprensión del Dominio (Utilidad del enfoque sistemático)	4.75	0.44	4	5
Razonamiento sobre Comportamiento (Uso de BDD)	4.68	0.47	4	5
Soporte a la Decisión (Guía paso a paso)	4.57	0.69	3	5
Precisión en la Identificación de Servicios	4.46	0.74	3	5
Percepción de Mantenibilidad	4.50	0.63	3	5
Reflexión Arquitectónica (Comparativa A/B)	4.57	0.50	4	5

Los valores medios superiores a 4.45 en todas las dimensiones, con desviaciones estándar bajas (< 0.75), indican un fuerte consenso sobre la utilidad de BDMA. Destaca especialmente la mejora en la **Comprensión del Dominio** ($\mu = 4.75$), lo que sugiere que la fase de clasificación y extracción de elementos (Fases 1 y 2) es efectiva para reducir la ambigüedad de los requisitos iniciales.

Análisis Cualitativo

En las respuestas abiertas, los participantes señalaron que el marco actúa como un "mecanismo de ordenamiento cognitivo". La obligatoriedad de generar artefactos intermedios (como el mapa de elementos del dominio) forzó a los equipos a discutir detalles que habían pasado por alto en la fase de control. Como contraparte, se identificó una curva de aprendizaje inicial en la adopción de la sintaxis Gherkin y en la distinción entre *Comando* y *Evento*, lo que sugiere la necesidad de herramientas de soporte o capacitación previa.

Discusión y Amenazas a la Validez

Los resultados sugieren que BDMA es particularmente eficaz para cerrar la brecha de conocimiento en equipos que transicionan hacia microservicios. Al comparar las soluciones

de la Fase 1 vs. Fase 2, se observó que los diseños guiados por BDMA presentaban límites de servicio más cohesivos y un uso más explícito de patrones de comunicación (ACL, OHS), reduciendo el acoplamiento accidental.

Amenazas a la Validez:

- **Validez Interna:** El diseño intra-sujeto podría introducir un efecto de aprendizaje (los participantes entendían mejor el problema en la segunda fase). Sin embargo, el salto cualitativo en la documentación de decisiones (ADRs) es atribuible directamente al método.

- **Validez Externa:** Al tratarse de un entorno académico con un problema de juguete (*toy problem*), la generalización a proyectos industriales de gran escala con deuda técnica heredada debe tomarse con cautela.

- **Validez de Constructo:** Las métricas se basan en percepción subjetiva. Futuros estudios deberán incorporar métricas objetivas de arquitectura (ej. métricas de acoplamiento estructural).

Conclusiones y Trabajo Futuro

Este trabajo ha presentado *Behavior-Driven Microservice Architecture* (BDMA), un marco metodológico diseñado para abordar la complejidad inherente al diseño de sistemas distribuidos en escenarios *greenfield*. A través de un proceso iterativo de cinco fases, BDMA operacionaliza la integración de DDD y BDD, transformando la incertidumbre de los requisitos funcionales en una arquitectura verificable y trazable.

Las contribuciones principales radican en: (1) la formalización de un proceso reproducible que elimina la dependencia exclusiva de la intuición del arquitecto experto; (2) la definición de un conjunto de artefactos intermedios que aseguran la alineación continua entre negocio y tecnología; y (3) evidencia empírica preliminar que demuestra su eficacia pedagógica y práctica para mejorar la cohesión del diseño.

Implicaciones para la Automatización

Un hallazgo colateral significativo es que la estructura sistemática de BDMA lo convierte en un candidato ideal para la automatización. Al estandarizar las entradas (Gherkin) y las salidas (Contratos, Mapas), el marco sienta las bases para el desarrollo de agentes inteligentes. Futuras investigaciones explorarán el uso de Modelos de Lenguaje Grande (LLMs) para ejecutar las fases de extracción y clasificación de BDMA de manera autónoma, utilizando este marco metodológico como el .ª ndamiaje"de razonamiento para la IA.

Líneas de Trabajo Futuro

La agenda de investigación se expande en las siguientes direcciones:

- **Validación Industrial:** Ejecutar estudios de caso longitudinales en empresas de desarrollo de software para evaluar el impacto de BDMA en el *Time-to-Market* y la tasa de defectos arquitectónicos.
- **Métricas Objetivas:** Desarrollar un modelo de calidad cuantitativo que mida automáticamente el acoplamiento y la cohesión de los diseños generados por BDMA.
- **Herramientas de Soporte:** Construir una herramienta CLI o plugin de IDE que asista a los desarrolladores en la generación de los artefactos del marco, reduciendo la fricción manual.

Referencias

Bajaj, D., Bharti, U., Gupta, I., Gupta, P., & Yadav, A. (2024). GTMicro—Microservice identification approach based on deep NLP transformer model for greenfield developments. *International Journal of Information Technology*, 16(5), 2751-2761.

Bajaj, D., Goel, A., & Gupta, S. C. (2022). GreenMicro: identifying microservices from use cases in greenfield development. *IEEE Access*, 10, 67008-67018.

Battaglia, N., García, A. N., & Congiusti, A. (2024). Descubrimiento de Microservicios en Metodologías Ágiles: un mapeo sistemático de la literatura. *XXX Congreso Argentino de Ciencias de la Computación (CACIC)*(La Plata, 7 al 11 de octubre de 2024).

Brandolini, A. (2013). *Introducing event storming. blog*, Ziobrando's Lair, 18.

Cardoso, J. P. S. (2021). *A guide for microservices in greenfield projects* [Tesis de maestría, Instituto Politecnico do Porto (Portugal)].

Cervantes, H., & Kazman, R. (2024). *Designing software architectures: a practical approach*. Addison-Wesley Professional.

Erder, M., & Pureur, P. (2015). *Continuous architecture: sustainable architecture in an agile and cloud-centric world*. Morgan Kaufmann.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Ford, N., Parsons, R., Kua, P., & Sadalage, P. (2022). *Building evolutionary architectures*. ."O'Reilly Media, Inc."

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS quarterly*, 337-355.

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2), 4.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75-105.

Hippchen, B., Giessler, P., Steinegger, R., Schneider, M., & Abeck, S. (2017). Designing microservicebased applications by using a domain-driven design approach. *International Journal on Advances in Software*, 10(3&4), 432-445.

Josélyne, M. I., Tuheirwe-Mukasa, D., Kanagwa, B., & Balikuddembe, J. (2018). Partitioning microservices: A domain engineering approach. *Proceedings of the 2018 International Conference on Software Engineering in Africa*, 43-49.

Kalia, A. K., Xiao, J., Krishna, R., Sinha, S., Vukovic, M., & Banerjee, D. (2021). Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. Proceedings of the 29th *ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 1214-1224.

Kopp, O., Armbruster, A., & Zimmermann, O. (2018). Markdown Architectural Decision Records: Format and Tool Support. *ZEUS*, 55-62.

Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. *MartinFowler.com*, 25(14-26), 12.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, 15(4), 251-266.

Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025). *Designing microservices using ai: A systematic literature review*. Software, 4(1), 6.

Newman, S. (2021). *Building microservices: designing fine-grained systems*. .^{O}'Reilly Media, Inc."

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3), 45-77.

Ponce, F., Soldani, J., Astudillo, H., & Brogi, A. (2022). Smells and refactorings for micro-services security: *A multivocal literature review. Journal of Systems and Software*, 192, 111393.

Pries-Heje, J., Baskerville, R., & Venable, J. (2008). Evaluation risks in design science research: A framework. *Proceedings from the 3rd International Conference on Design Science Research in IT*, May 2008, Atlanta, Georgia, USA, 329-334.

Rademacher, F., Sachweh, S., & Zündorf, A. (2017). Towards a UML profile for domain-driven design of microservice architectures. *International Conference on Software Engineering and Formal Methods*, 230-245.

Schmidt, R. A., & Thiry, M. (2020). Microservices identification strategies: A review focused on Model-Driven Engineering and Domain Driven Design approaches. *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, 1-6.

Smart, J. F., & Molak, J. (2023). *BDD in Action: Behavior-driven development for the whole software lifecycle*. Simon; Schuster.

Taibi, D., & Lenarduzzi, V. (2018). On the definition of microservice bad smells. *IEEE software*, 35(3), 56-62.

Ünlü, H., Kennouche, D. E., Soylu, G. K., & Demirörs, O. (2024). Microservice-based projects in agile world: A structured interview. *Information and Software Technology*, 165, 107334.

Vera-Rivera, F. H., Cuadros, E. G. P., Perez, B., Astudillo, H., & Gaona, C. (2023). SEM-GROMI—a semantic grouping algorithm to identifying microservices using semantic similarity of user stories. *PeerJ Computer Science*, 9, e1380.

Vera-Rivera, F. H., Puerto-Cuadros, E. G., Astudillo, H., & Gaona-Cuevas, C. M. (2020). Microservices backlog-a model of granularity specification and microservice identification. *International Conference on Services Computing*, 85-102.

Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.

Vernon, V. (2016). *Domain-driven design distilled. Addison-Wesley Professiona*l.

Wieringa, R. (2014). D*esign science methodology for information systems and software engineering*. Springer.

Woods, E., Erder, M., & Pureur, P. (2021). *Continuous architecture in practice: Software architecture in the age of agility and DevOps*. Addison-Wesley Professional.

Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domaindriven design for microservices: An evidence-based investigation. *IEEE Transactions on Software Engineering*, 50(6), 1425-1449.

Zimmermann, O., Stocker, M., Lubke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API design: simplifying integration with loosely coupled message exchanges.* Addison-Wesley Professional.

Zimmermann, O., Stocker, M., Lübke, D., Pautasso, C., & Zdun, U. (2019). Introduction to microservice API patterns (MAP).