



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

Autor: Mg. Nicolás Battaglia
Director: Dr. Gustavo Rossi
Codirector: Dr. Alejandro Fernández

Tesis presentada para obtener el título de

Doctor en Ciencias Informáticas

Diciembre 2025.

Resumen:

El diseño de arquitecturas basadas en microservicios en contextos ágiles *greenfield* presenta desafíos significativos, especialmente en la correcta identificación de límites de los servicios y en la alineación entre requisitos funcionales y decisiones arquitectónicas. Aunque *Domain-Driven Design* (DDD) y *Behavior-Driven Development* (BDD) ofrecen principios valiosos para comprender el dominio y especificar comportamientos, su aplicación práctica suele darse de manera aislada, sin mecanismos que garanticen trazabilidad, validación continua o diseño iterativo sustentado en evidencia. Como consecuencia, los proyectos enfrentan fragmentación del conocimiento, decisiones arquitectónicas intuitivas y dificultades para adaptar la solución a la evolución de los requisitos.

Esta tesis propone ***Behavior-Driven Microservice Architecture (BDMA)***, un marco metodológico sistemático, iterativo y reproducible que integra prácticas de DDD, BDD y principios de arquitectura evolutiva para guiar la identificación, diseño y evolución de microservicios en proyectos *greenfield*. BDMA transforma escenarios funcionales redactados en lenguaje natural en artefactos arquitectónicos verificables, asegurando trazabilidad funcional completa y favoreciendo la coherencia semántica entre negocio y tecnología. El método se estructura en cinco fases: descubrimiento funcional colaborativo, redacción y validación de escenarios BDD, identificación de contextos delimitados, diseño iterativo de microservicios y validación/refactorización continua. La propuesta fue validada mediante un mapeo sistemático, un experimento exploratorio y un caso de aplicación, evidenciando mejoras en la comprensión del dominio, precisión en la identificación de microservicios y capacidad de adaptación frente a cambios. BDMA contribuye así a la ingeniería de software ofreciendo una metodología aplicable, fundamentada y alineada con los principios ágiles para el diseño sostenible de arquitecturas de microservicios.

Palabras claves:

Microservicios. *Domain-Driven Design*. *Behavior-Driven Development*. Metodologías ágiles. *Greenfield*.

Dedicatoria

A mi abuelo, por haberme guiado e inspirado en este camino.

Dedico este trabajo a mi familia, que me acompañó y sostuvo durante todo este largo proceso. En especial, a mi mamá, por su apoyo constante e incondicional, y a mi papá, que ya no está, pero cuya presencia me sigue acompañando en cada paso.

Agradecimientos

Quiero agradecer, en primer lugar, a mi director, Gustavo Rossi, por la confianza y el acompañamiento a lo largo de todo este recorrido. Su experiencia, su mirada crítica y su permanente disposición al diálogo fueron fundamentales para orientar esta investigación y enriquecerla en cada una de sus etapas, dejando además una huella profunda en mi formación como investigador.

Agradezco también a Alejandro Fernández, quien se desempeñó como codirector durante el proceso doctoral, por sus aportes, sugerencias y disponibilidad para discutir ideas en distintos momentos del trabajo.

Quiero agradecer especialmente a la Universidad Abierta Interamericana, y en particular a Marcelo De Vincenzi, por la confianza depositada en mí y su constante apoyo, y a Carlos Neil, por el acompañamiento permanente a lo largo de este proceso. Asimismo, agradezco a los espacios académicos y de investigación de la institución, así como a los colegas, docentes y estudiantes que participaron en las experiencias empíricas y actividades de validación de esta tesis, cuyas miradas y aportes resultaron fundamentales para fortalecer la propuesta.

Finalmente, agradezco a todas las personas que, de una u otra forma, acompañaron este camino con apoyo, confianza y aliento constante. Imposible nombrarlos a todos.

Jurado

- Dr. Matías Urbietta
- Dr. Cesar Javier Acuña
- Dr. Marcelo Marciszack

Índice general

Capítulo 1. Introducción.....	1
1.1. Problema de investigación	2
1.2. Solución propuesta	2
1.3. Objetivos.....	2
1.4. Metodología	3
1.5. Contribuciones principales.....	3
1.6. Estructura general de la tesis	4
Capítulo 2. Background técnico	6
2.1. Arquitecturas de software	6
2.2. <i>Domain-Driven Design</i>	7
2.3. <i>Behavior-Driven Development</i>	10
2.4. Desarrollo <i>greenfield</i>	14
2.5. Arquitecturas ágiles.....	17
Capítulo 3. Enfoque metodológico	18
3.1. <i>Design Science Research</i> : fundamentos y aplicación.....	18
3.2. Extensión metodológica: ciclo de consolidación	19
3.3. Estructura del proceso DSR en la tesis	19
Capítulo 4. Ciclo de relevancia	21
4.1. Análisis del contexto.....	21
4.2. Identificación del problema	23
Capítulo 5. Ciclo de rigor	29
5.1. Trabajos Relacionados.....	29
5.2. Síntesis crítica.....	38
Capítulo 6. Ciclo de Diseño	40
6.1. Componentes conceptuales del marco BDMA.....	40
6.2. Diferencial metodológico.....	41
6.3. El marco Behavior-Driven Microservices Architecture	41
6.4. El Marco BDMA en detalle	47
6.5. Beneficios de implementar BDMA	60
6.6. Cierre del ciclo de diseño.....	62
Capítulo 7. Caso de aplicación del marco BDMA	63
7.1. Contexto	63
7.2. Aplicación del marco BDMA.....	64
7.3. Evidencia de los beneficios de BDMA en el caso de estudio	86
Capítulo 8. Ciclo de consolidación.....	89
8.1. Estrategia de evaluación	89
8.2. Evaluación de la aplicación del método BDMA	90
Capítulo 9. Conclusión y trabajos futuros	96
9.1. Conclusión.....	96
9.2. Trabajos futuros.....	98
Anexo A. Escenarios BDD Utilizados en el capítulo 7.....	100
Anexo B. Rúbrica y encuesta de percepción del experimento exploratorio	106
Anexo C. Resumen comparativo de los trabajos relacionados con BDMA	109
Bibliografía	117
Acrónimos y Abreviaturas	123

Índice de figuras

Ilustración 1. Enfoque tradicional de desarrollo (Smart and Molak, 2023)	11
Ilustración 2. BDD usa ejemplos para guiar las conversaciones (Smart and Molak, 2023) 11	
Ilustración 3. Ciclo de vida de TDD y BDD	12
Ilustración 4. Escenario BDD para la historia de usuario de inicio de sesión	13
Ilustración 5. Ciclos DSR (Hevner, 2007)	19
Ilustración 6. Marco BDMA	43
Ilustración 7. Mapa de bounded context con relaciones y patrones.....	52
Ilustración 8. Mapa visual de bounded context del caso de estudio de BDMA.....	72

Índice de tablas

Tabla 1. Patrones de relación entre bounded context.....	8
Tabla 2. Example Mapping para un caso de inicio de sesión	13
Tabla 3. Resumen de ventajas y desventajas de los diferentes enfoques	26
Tabla 4. Categorización y resumen de los vacíos observados.....	26
Tabla 5. Resumen del marco BDMA	43
Tabla 6. Relación de BDMA con el ciclo ágil.....	47
Tabla 7. Ejemplo artefacto fase 1 BDMA – mapa de agrupamiento funcional.....	48
Tabla 8. Ejemplo artefacto fase 2 BDMA – mapa de elementos del dominio	50
Tabla 9. Ejemplo artefacto fase 3 BDMA – mapa de bounded contexts	51
Tabla 10. Ejemplo artefacto fase 3 BDMA – mapa de interacción entre contextos.....	53
Tabla 11. Ejemplo artefacto fase 4 BDMA – especificación de contratos API	56
Tabla 12. Artefacto fase 5 BDMA - informe de trazabilidad evolutiva	60
Tabla 13. Mapa de agrupamiento funcional (Artefacto fase 1 BDMA)	66
Tabla 14. Mapa de elementos del dominio por escenario (artefacto fase 2 BDMA)	70
Tabla 15. Mapa de bounded context (artefacto fase 3 BDMA)	72
Tabla 16. Mapa de interacciones entre contextos (artefacto fase 3 BDMA).....	75
Tabla 17. Identificación de contextos que requieren API (artefacto fase 3 BDMA)	76
Tabla 18. Especificación de contratos API (artefacto fase 4 BDMA).....	80
Tabla 19. Informe de trazabilidad evolutiva (artefacto fase 5 BDMA)	84
Tabla 20. Relaciones entre bounded contexts (artefacto fase 5 BDMA)	85
Tabla 21. Conocimientos previos de los participantes de la evaluación inicial de BDMA .	93
Tabla 22. Resultados cuantitativos de la evaluación inicial de BDMA	94
Tabla 23. Rúbrica y encuesta de percepción del experimento exploratorio.	107

Capítulo 1. Introducción

En las últimas décadas, la arquitectura de microservicios se ha consolidado como un enfoque dominante para el desarrollo de sistemas distribuidos debido a su capacidad para mejorar la escalabilidad, la mantenibilidad y el despliegue independiente de componentes (Newman, 2021; Richardson, 2018). Sin embargo, más allá de sus beneficios técnicos, el diseño efectivo de microservicios plantea desafíos significativos, entre los cuales destaca la correcta identificación de sus límites. Esta decisión arquitectónica crítica influye directamente en la cohesión interna, el acoplamiento entre servicios y la capacidad de evolución del sistema (Jamshidi et al., 2018; Taibi and Lenarduzzi, 2018).

Esta dificultad se intensifica en proyectos que comienzan desde cero sin sistemas legados que sirvan como referencia (denominados *greenfield*). En tales escenarios, el conocimiento del dominio aún se encuentra en construcción y, por lo tanto, las decisiones de diseño suelen apoyarse en intuiciones técnicas o criterios poco estructurados. Como consecuencia, es frecuente la aparición de microservicios segmentados de forma arbitraria o desalineada con la lógica del negocio (Su et al., 2024; Vural and Koyuncu, 2021).

Para enfrentar esta problemática, ha sido propuesto *Domain-Driven Design* (DDD) como un enfoque estratégico, ya que promueve la modelación explícita del dominio mediante *bounded contexts*, entidades y eventos, favoreciendo una estructuración funcional coherente del sistema (Evans, 2004; Vernon, 2013). No obstante, su implementación rigurosa en entornos ágiles ha evidenciado tensiones metodológicas: la necesidad de definir estructuras arquitectónicas anticipadamente entra en conflicto con los principios ágiles de diseño emergente, entrega continua y adaptación al cambio (Fowler, 2012a; Kruchten, 1995; Zimmermann, 2017). Diversos estudios coinciden en señalar la falta de lineamientos prácticos para aplicar DDD de forma iterativa y colaborativa en situaciones de alta incertidumbre, como las que caracterizan a los proyectos *greenfield* (Zhong et al., 2024).

Por otro lado, las metodologías ágiles postulan que la arquitectura debe emerger progresivamente a medida que se avanza en la comprensión del dominio. Esta visión, conocida como arquitectura evolutiva, trata las decisiones estructurales como hipótesis sujetas a validación continua, en lugar de decisiones fijas tomadas en fases iniciales (Ford et al., 2017; Nord and Tomayko, 2006). Sin embargo, en ausencia de mecanismos que conecten los requerimientos funcionales con las decisiones de diseño, existe el riesgo de caer en una arquitectura accidental, difícil de mantener y escasamente alineada con las necesidades del negocio (Booch, 2006).

En este contexto, *Behavior-Driven Development* (BDD) emerge como una práctica que puede fortalecer el vínculo entre funcionalidades esperadas y decisiones arquitectónicas. BDD promueve la especificación funcional mediante escenarios de comportamiento redactados en lenguaje natural estructurado, lo que facilita la colaboración entre perfiles técnicos y de negocio, y mejora la trazabilidad desde los requerimientos hasta la implementación (Hellesoy et al., 2017; Smart and Molak, 2023). Aunque ha sido ampliamente utilizado para mejorar la calidad de las pruebas y la

claridad de los requisitos, su potencial como instrumento para la identificación de componentes arquitectónicos permanece poco explotado (Irshad et al., 2021).

1.1. Problema de investigación

A partir del análisis anterior, se identifican tres obstáculos principales que limitan la correcta identificación de microservicios en contextos ágiles *greenfield*:

1. La falta de integración metodológica entre DDD y enfoques ágiles, lo cual dificulta su aplicación en ciclos iterativos y colaborativos (Zhong et al., 2024).
2. La ausencia de mecanismos que aseguren la trazabilidad entre requerimientos funcionales y decisiones arquitectónicas, debilitando la validación continua del diseño (Kapferer and Zimmermann, 2020; Su et al., 2024)
3. La carencia de estrategias sistemáticas para delimitar microservicios en proyectos sin sistemas preexistentes, donde el conocimiento del dominio es aún parcial (Bajaj et al., 2022; Vural and Koyuncu, 2021).

1.2. Solución propuesta

Con el objetivo de abordar estos desafíos, esta tesis propone *Behavior-Driven Microservice Architecture* (BDMA), un marco metodológico iterativo que combina DDD con BDD para facilitar la identificación y el diseño evolutivo de microservicios en entornos ágiles *greenfield*. Es un aporte original que integra prácticas existentes de forma no abordada previamente.

BDMA se estructura en cinco fases y su objetivo central es construir una arquitectura de software de manera emergente, coherente y adaptativa, construida desde el comportamiento observable del sistema, reduciendo la deuda técnica temprana y mejorando la alineación funcional-tecnológica.

La propuesta fue validada mediante estudios exploratorio y un caso de aplicación, evidenciando mejoras en la trazabilidad funcional, la coherencia semántica entre servicios y la capacidad de adaptación ante cambios de requisitos. El marco propuesto se sitúa en la convergencia de tres áreas clave de la ingeniería de software: las arquitecturas de software, los microservicios y el desarrollo ágil.

1.3. Objetivos

Objetivo principal

- Diseñar y validar un marco metodológico denominado *Behavior-Driven Microservice Architecture*, que integre BDD y DDD, para facilitar la identificación iterativa de microservicios en proyectos ágiles *greenfield*, favoreciendo la trazabilidad funcional, la coherencia arquitectónica y la adaptación a cambios del dominio.

Objetivos secundarios

- Caracterizar los desafíos metodológicos actuales en la identificación de microservicios en entornos ágiles *greenfield*, para comprender las limitaciones de DDD en ciclos iterativos y la escasa trazabilidad entre requisitos funcionales y decisiones arquitectónicas.

- Diseñar la metodología BDMA como una secuencia iterativa de fases que combine prácticas de BDD (como *Example Mapping* y Gherkin) con artefactos estratégicos de DDD (contextos delimitados, eventos del dominio) para favorecer la identificación progresiva de microservicios.
- Adaptar y aplicar técnicas de análisis funcional para permitir la agrupación semántica de escenarios BDD y la delimitación automatizada o asistida de microservicios.
- Implementar y documentar la aplicación de BDMA en un caso de estudio representativo del dominio para evidenciar el proceso iterativo desde los requisitos hasta la arquitectura emergente.
- Evaluar empíricamente el impacto de BDMA mediante métricas e instrumentos para valorar su efectividad en términos de trazabilidad, alineación funcional, mantenibilidad y respuesta a cambios de requisitos.

1.4. Metodología

La estrategia metodológica de esta tesis se apoya en contribuciones clave en el campo de la ingeniería de software y los sistemas de información. El enfoque *Design Science Research* (DSR) fue propuesto inicialmente por (Hevner et al., 2004) y ampliado por (Peppers et al., 2007), (Wieringa, 2014) y (Gregor and Hevner, 2013), quienes establecen su aplicabilidad en contextos donde se requiere diseñar soluciones innovadoras a problemas prácticos. Por su parte, en (Wohlin et al., 2012), (Kitchenham and Charters, 2007), y (Runeson and Höst, 2009) se aportan lineamientos sólidos para la conducción y validación de estudios empíricos en ingeniería de software, complementado el proceso con una evaluación empírica exploratoria. Esta combinación metodológica resulta adecuada para investigaciones en ingeniería de software que buscan desarrollar y validar propuestas innovadoras de carácter práctico (Hevner et al., 2004).

1.5. Contribuciones principales

Esta tesis aporta tres contribuciones principales al campo la Ingeniería de Software en el contexto del diseño de arquitecturas ágiles basadas en microservicios:

1. **Un marco metodológico iterativo** que permite la identificación progresiva y colaborativa de microservicios en contextos *greenfield*.
2. **Un caso de aplicación concreto** que ilustra paso a paso cómo se implementa la propuesta metodológica en un sistema realista, mostrando la viabilidad práctica del enfoque.
3. **Una validación exploratoria controlada** que proporciona evidencia preliminar sobre los beneficios del enfoque en términos de coherencia arquitectónica, capacidad de adaptación frente a cambios y comprensión compartida del dominio.

1.5.1. Trabajos presentados vinculados con la tesis

- Battaglia, N. (2025). Identificación Ágil de Microservicios Utilizando DDD y BDD. *XXVIII Ibero-American Conference on Software Engineering (CIBSE 2025 Doctoral Symposium)*.

- Battaglia, N., Garcia, A. N., & Congiusti, A. (2024). Descubrimiento de Microservicios en Metodologías Ágiles: un mapeo sistemático de la literatura. *XXX Congreso Argentino de Ciencias de La Computacion*, 1506–1510.
- Battaglia, N., Rossi, G., & Fernández., A. (2025). Behavior-Driven Microservice Architecture: Un Marco Metodológico para la Identificación Iterativa de Microservicios en Proyectos Ágiles Greenfield. *Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería Del Conocimiento IIISIC 2025*.
- Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025). Designing microservices using AI: A systematic literature review. *Software*, 4(1), 6.
- Narváez, D., Rossi, G., & Battaglia, N. (2024). Aplicación de inteligencia artificial en el diseño de microservicios: Un mapeo sistemático de la literatura. Libro de Actas- 30° Congreso Argentino de Ciencias de La Computación-CACIC 2024, 639–648. <http://dx.doi.org/10.35537/10915/172755>

1.6. Estructura general de la tesis

- **Capítulo 1 – Introducción:** define el problema y los desafíos de identificar microservicios en entornos ágiles *greenfield*. Presenta la motivación y BDMA como respuesta, junto con los objetivos, la metodología y las principales contribuciones que guían la tesis.
- **Capítulo 2 – Background técnico:** presenta los pilares teóricos del trabajo, estilos de arquitectura, fundamentos de DDD y BDD, y conceptos clave del desarrollo *greenfield* y la arquitectura ágil. Proporciona el marco necesario para comprender la propuesta BDMA.
- **Capítulo 3 – Enfoque metodológico:** explica la metodología basada en *Design Science Research* (DSR), organizada en los ciclos de relevancia, rigor, diseño y consolidación que guiaron la construcción y validación de BDMA. Garantiza el rigor científico y prepara la presentación del método.
- **Capítulo 4 – Ciclo de relevancia:** profundiza la identificación de la problemática mediante un mapeo sistemático y un experimento inicial, que revelan brechas en la integración DDD–BDD, la trazabilidad y los desafíos en proyectos sin sistemas previos. Con ello se define el problema de investigación que aborda BDMA.
- **Capítulo 5 – Ciclo de rigor:** presenta una revisión crítica de enfoques existentes para identificar microservicios, analizando sus fortalezas, limitaciones y aplicabilidad en contextos *greenfield*. Concluye con una síntesis comparativa que muestra los vacíos actuales y cómo BDMA los supera
- **Capítulo 6 – Ciclo de diseño:** presenta en detalle el marco BDMA, sus principios y componentes conceptuales. Describe sus cinco fases iterativas, con actividades y artefactos, y explica su diferencial metodológico y relación con el desarrollo ágil.
- **Capítulo 7 – Caso de aplicación del Método BDMA:** demuestra la aplicación de BDMA mediante un caso de estudio, documentando resultados. El ejemplo evidencia su viabilidad y la trazabilidad desde los requisitos hasta la arquitectura.

- **Capítulo 8 – Ciclo de consolidación:** presenta la evaluación integral de la propuesta BDMA a partir de dos experimentos controlados. Por cada uno de ellos se analizan los resultados obtenidos y se realiza un análisis crítico.
- **Capítulo 9 – Conclusión y trabajos futuros:** el capítulo final presenta las conclusiones del estudio, los aportes y limitaciones de BDMA. Asimismo, propone líneas de trabajo que permiten ampliar y profundizar la investigación, destacando la relevancia del marco desarrollado.

Capítulo 2. *Background* técnico

Este capítulo desarrolla los pilares conceptuales que sustentan la propuesta metodológica: el diseño guiado por el dominio (DDD), el desarrollo guiado por el comportamiento (BDD), los principios arquitectónicos de microservicios y la noción de arquitectura emergente en contextos ágiles. Cada uno de estos enfoques aporta prácticas complementarias que, al integrarse, permiten abordar la complejidad del diseño de sistemas distribuidos desde una perspectiva evolutiva y colaborativa.

Los conceptos desarrollados en este capítulo constituyen la base teórica necesaria para comprender las decisiones metodológicas del marco BDMA presentado posteriormente en el Capítulo 6.

2.1. Arquitecturas de software

Antes de analizar los estilos arquitectónicos modernos, es crucial definir qué entendemos por arquitectura de software. Desde sus orígenes, la disciplina ha sido conceptualizada como el conjunto de estructuras y decisiones fundamentales que permiten organizar y evolucionar un sistema (Shaw and Garlan, 1996). Según el estándar ISO/IEC 42010:2011 (Author, 2011), la arquitectura también expresa cómo un sistema está pensado para cambiar, no solo cómo está estructurado. Autores como (Bass et al., 2012) vinculan estas decisiones con atributos de calidad como rendimiento, modificabilidad y seguridad. (Hofmeister et al., 2000; Kruchten, 1995) proponen abordar la arquitectura desde múltiples vistas complementarias, como lógica, desarrollo o escenarios de uso. En contextos ágiles, (Fowler, 2003) redefine la arquitectura como las decisiones difíciles de cambiar, enfatizando el equilibrio entre anticipación y evolución. Esta perspectiva ha sido ampliada por (Ford et al., 2017; M. G. Waterman, 2014) quienes plantean la arquitectura evolutiva como una práctica basada en adaptabilidad, automatización y trazabilidad. En síntesis, la arquitectura de software constituye un puente entre el negocio y la implementación técnica. En entornos ágiles y *greenfield* su rol es estratégico para garantizar sostenibilidad y relación con el negocio. A continuación, se exploran estilos arquitectónicos clave, desde monolitos hasta microservicios y EDA, con foco en sus implicancias para el diseño iterativo y basado en el dominio.

2.1.1. Estilos de arquitecturas

En el ámbito de la ingeniería de software, un estilo arquitectónico se refiere a un patrón de organización estructural repetible que define un conjunto de componentes de software, sus roles, las formas en que estos interactúan y las restricciones de composición entre ellos (Shaw and Garlan, 1996). La selección de un estilo u otro tiene implicaciones directas sobre la capacidad del sistema para satisfacer ciertos atributos de calidad (como rendimiento, escalabilidad, modificabilidad, mantenibilidad, etc.), por lo que diversas guías metodológicas enfatizan la importancia de elegir el estilo arquitectónico en función de los requerimientos y contexto del proyecto (Hofmeister et al., 2000). En particular, en (Bass et al., 2012) señala que las decisiones de arquitectura deben estar vinculadas con los escenarios de calidad esperados del sistema, ya que la estructura arquitectónica condiciona propiedades como la facilidad de cambio, la escalabilidad o la tolerancia a fallos.

2.2. Domain-Driven Design

DDD es un enfoque que sitúa el dominio del negocio como eje central de las decisiones arquitectónicas (Evans, 2004), buscando alinear la estructura del software con modelos semánticos coherentes. Su propuesta se articula en dos niveles complementarios: estratégico, que delimita el alcance y significado de los conceptos del negocio, y táctico, que define cómo se implementa el modelo dentro de cada límite conceptual (Evans, 2004).

En el plano **estratégico**, DDD identifica contextos delimitados o *bounded contexts*, regiones del sistema donde los términos del dominio tienen un significado preciso y compartido. El lenguaje ubicuo (utilizado por equipos técnicos y de negocio) evita ambigüedades, mientras que los mapas de contexto o *context maps* representan las relaciones entre contextos, empleando patrones. En arquitecturas de microservicios, cada *bounded context* suele corresponder a un servicio autónomo, favoreciendo la cohesión, la escalabilidad organizacional y la separación de responsabilidades. En el plano **táctico**, DDD proporciona patrones que permiten implementar la lógica del dominio dentro de cada contexto de manera consistente y aislada. Entre ellos se encuentran las entidades con identidad persistente, los objetos de valor, los agregados como unidades de consistencia, los servicios de dominio para lógica que no pertenece a una entidad específica y los eventos de dominio, que comunican cambios relevantes de forma desacoplada. La complementariedad entre niveles estratégico y táctico permite afrontar la complejidad de sistemas distribuidos mediante una estructura coherente y guiada por el conocimiento del negocio.

No obstante, la aplicación estricta de DDD puede resultar desafiante en entornos ágiles. Su énfasis inicial en modelar antes de construir puede entrar en tensión con prácticas iterativas que priorizan entrega continua y aprendizaje emergente (Beck, 2003; Kruchten, 1995, 2010). En escenarios *greenfield*, donde el dominio aún está en descubrimiento, intentar definir todos los contextos por adelantado puede llevar a diseños rígidos o prematuros (Brandolini, 2018; Zhong et al., 2024). Ante estas dificultades, se han propuesto enfoques que complementan DDD con mecanismos adaptativos, como lenguajes específicos de dominio (DSL) para facilitar colaboración y trazabilidad (Kapferer and Zimmermann, 2020), o advertencias sobre el riesgo del diseño accidental, cuando las decisiones arquitectónicas tempranas no se revisan (Booch, 2006). En consecuencia, las visiones contemporáneas conciben DDD como un marco dinámico que debe integrarse con prácticas ágiles para modelar, validar y refinar el dominio de manera iterativa.

2.2.1. Patrones clásicos de integración entre *bounded contexts*

Los *bounded contexts* son una pieza central del enfoque DDD, definidos como límites semánticos dentro de los cuales un modelo de dominio tiene un significado coherente y consistente. Cuando múltiples contextos coexisten en una misma solución, las interacciones entre ellos deben ser cuidadosamente diseñadas para evitar acoplamiento excesivo, ambigüedad semántica y dependencias técnicas rígidas.

Para ello, en (Evans, 2004) se introdujo un conjunto de patrones de relación entre contextos, que describen formas típicas de comunicación, colaboración y dependencia entre contextos. Esta línea fue luego ampliada por (Vernon, 2013), quien aplicó estos patrones a arquitecturas distribuidas modernas.

Además, enfoques como el propuesto en (Rademacher et al., 2018) introdujeron representaciones formales de estas relaciones mediante perfiles UML adaptados a arquitecturas basadas en microservicios, lo que refuerza su utilidad como componente estructural en el diseño de sistemas evolutivos. En la tabla 1 se observa un resumen general de los patrones

Patrón	Descripción	Referencias
Conformist	Un contexto consumidor adopta sin adaptación el modelo del proveedor. Minimiza esfuerzo, pero genera dependencia fuerte.	(Evans, 2004; Vernon, 2013)
Partnership	Ambos contextos colaboran activamente para definir una interfaz compartida. Alta coordinación, menor autonomía.	(Evans, 2004; Vernon, 2013)
ACL (Anti-Corruption Layer)	El consumidor traduce y adapta el modelo del proveedor para mantener su independencia semántica.	(Evans, 2004; Rademacher et al., 2018)
Open Host Service / Published Language	El proveedor define una API o esquema compartido como contrato estable. Promueve interoperabilidad.	(Evans, 2004)
Shared Kernel	Dos contextos comparten explícitamente parte del modelo, acordando su evolución conjunta.	(Evans, 2004; Vernon, 2013)

Tabla 1. Patrones de relación entre bounded context

La relevancia de los patrones de contexto se ha intensificado con la adopción de microservicios, donde cada contexto suele implementarse como un servicio independiente. Patrones como *OHS* y *Published Language* encajan naturalmente con API bien documentadas, permitiendo a los equipos generar clientes sin adaptadores adicionales, mientras que ACL actúa como aislante frente a cambios externos.

DDD también se vincula con prácticas ágiles contemporáneas. En (Skelton and Pais, 2019) relacionan los límites de servicio con la estructura organizacional. Los autores proponen heurísticas y eventos colaborativos (como *Event Storming*) para descubrir dominios emergentes. En proyectos *greenfield*, los patrones de DDD se aplican desde el comienzo: primero se modela el dominio y se construyen los mapas de contextos; luego se definen contratos, esquemas de eventos y mecanismos de integración (Evans, 2004).

Finalmente, se reconoce que parte de estas prácticas proviene de literatura técnica no académica (blogs, documentación de industria) coherente con el origen pragmático de DDD. Esta tesis integra dichas fuentes con referencias académicas para garantizar validez y actualidad.

2.2.2. Estilos arquitectónicos: evolución y comparativa

Los estilos arquitectónicos representan soluciones recurrentes para organizar los componentes y flujos de un sistema. Esta sección resume brevemente los estilos más relevantes para la propuesta de esta tesis.

- **Arquitectura monolítica:** es un estilo donde todos los módulos del sistema se despliegan y ejecutan como una única unidad. Aunque su simplicidad inicial facilita la entrega temprana, con el tiempo tiende a generar acoplamientos excesivos y dificultades para escalar o mantener (Richards and Ford, 2020).
- **Arquitectura en capas:** es un subtipo de monolito que organiza la lógica en capas (por ejemplo: presentación, negocio, persistencia), promoviendo separación de responsabilidades. Sin embargo, no elimina el problema de despliegue único y acopla todo con las capas inferiores.
- **Arquitectura hexagonal y limpia:** estos enfoques (Cockburn, 2005; Martin, 2012) abordan el acoplamiento técnico en monolitos, separando explícitamente el núcleo del dominio de sus adaptadores externos (UI, bases de datos). Son útiles para mantener coherencia y testeabilidad.
- **Microservicios:** constituyen una evolución del enfoque monolítico. Se trata de un conjunto de servicios independientes, autónomos y desplegables por separado, orientados a capacidades del negocio (Newman, 2021). Esta estrategia permite escalabilidad granular y despliegue independiente, pero requiere identificar correctamente los límites funcionales (Richardson, 2018).
- **SOA y Event-Driven Architecture (EDA):** la arquitectura orientada a servicios (SOA) promueve la reusabilidad mediante contratos bien definidos y un bus de servicios compartido. Aunque similar a microservicios en modularidad, suele implicar mayor acoplamiento tecnológico. Por otro lado, EDA se basa en la comunicación asíncrona mediante eventos, lo que facilita la extensibilidad, pero aumenta la complejidad operativa.

Los estilos se diferencian en criterios como mantenibilidad, capacidad de adaptación al cambio e independencia de despliegue, entre otros.

2.2.3. Arquitectura de microservicios

La arquitectura de microservicios surge como respuesta a los límites de escalabilidad y mantenibilidad de sistemas monolíticos. Propone descomponer aplicaciones en servicios pequeños y autónomos, orientados a capacidades del negocio, que se comunican mediante mecanismos ligeros como HTTP o mensajería (Lewis and Fowler, 2014; Newman, 2021). Cada servicio encapsula su lógica, almacenamiento y API, siguiendo el principio de mínima compartición (Richards and Ford, 2020). Este estilo ofrece ventajas sustanciales: refuerza la modularidad, permite desarrollo y despliegue independientes, habilita escalabilidad granular y soporta diversidad tecnológica (Bass et al., 2012; Fowler, 2015c, 2015a, 2015b). Además, facilita la evolución de *stacks* tecnológicos al reducir el riesgo de cambio. Sin embargo, también introduce una complejidad significativa. Al distribuir el sistema, aparecen desafíos como fallos de red, latencia, consistencia eventual y necesidad de patrones como sagas (Richardson, 2018). Además, requiere infraestructura robusta y automatización (contenedores, orquestadores, observabilidad) para gestionar servicios, versiones y configuraciones (Newman, 2021; Richards and Ford, 2020).

2.2.4. Monolito vs. microservicios

La elección entre una arquitectura monolítica o basada en microservicios debe ajustarse al contexto del proyecto. Para sistemas simples o en etapas iniciales, un

monolito modular suele ser más eficiente. La filosofía *Monolith First* (Fowler, 2015c) propone comenzar con un monolito bien estructurado y solo migrar a microservicios si la complejidad lo justifica (Fowler, 2015b; Richardson, 2018). Cuando se anticipa escalabilidad alta, equipos paralelos o cambios frecuentes, los microservicios ofrecen ventajas claras de modularidad y despliegue independiente (Jamshidi et al., 2018; Newman, 2021). Sin embargo, su implementación exitosa requiere prácticas *DevOps* maduras e inversión en automatización y monitoreo. Aunque muchas empresas migran hacia microservicios para resolver cuellos de botella, estudios como Su et al. (2024) documentan casos de vuelta al monolito motivados por latencia entre servicios, sobrecarga operativa o dificultades de consistencia. Estos hallazgos confirman que los microservicios no son adecuados en todos los casos.

En síntesis, monolitos y microservicios representan extremos de un continuo arquitectónico: los primeros favorecen simplicidad y cohesión centralizada; los segundos, escalabilidad distribuida y modularidad. No existe una solución universal: la decisión debe basarse en el tamaño del sistema, la estrategia de evolución, la estructura del equipo y los objetivos de (Bass et al., 2012).

2.3. Behavior-Driven Development

BDD (North, 2006) es un enfoque de desarrollo centrado en el comportamiento esperado del sistema, expresado mediante lenguaje natural estructurado para facilitar la comprensión entre negocio, analistas, testers y desarrolladores. Propone definir primero el comportamiento desde la perspectiva del usuario mediante escenarios, automatizarlos como pruebas y utilizar estas pruebas para guiar el diseño del dominio y de la aplicación. A diferencia de los enfoques tradicionales que son susceptibles a errores de comunicación e interpretaciones divergentes, BDD orienta las conversaciones a ejemplos concretos y automatizables (Smart and Molak, 2023), reduciendo significativamente la pérdida de información y las ambigüedades.

Las ilustraciones 1 y 2 reflejan este contraste entre el desarrollo tradicional y el enfoque guiado por ejemplos.

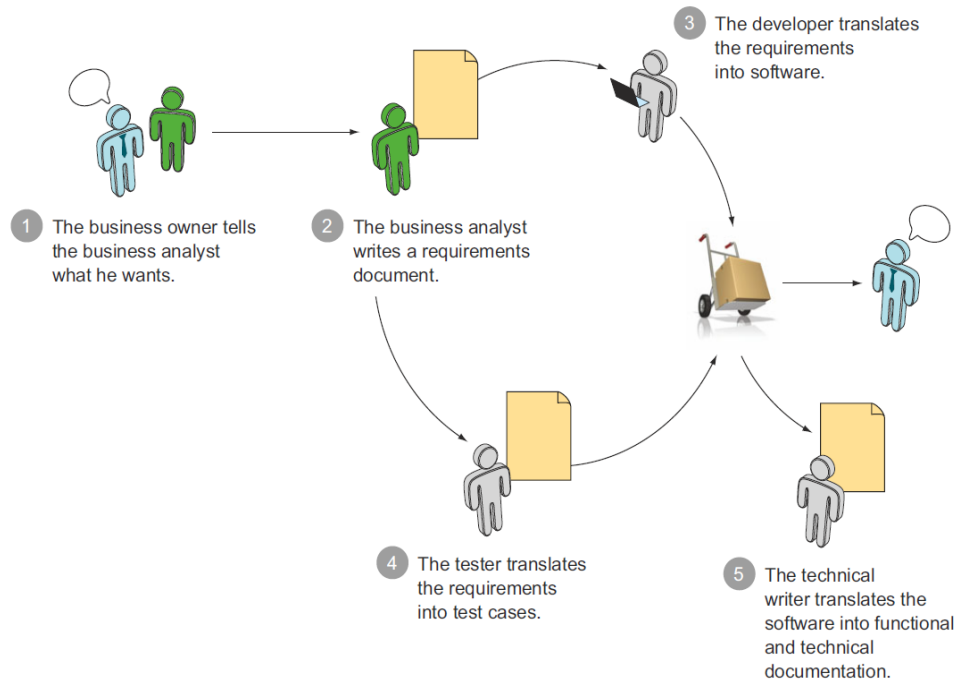


Ilustración 1. Enfoque tradicional de desarrollo (Smart and Molak, 2023)

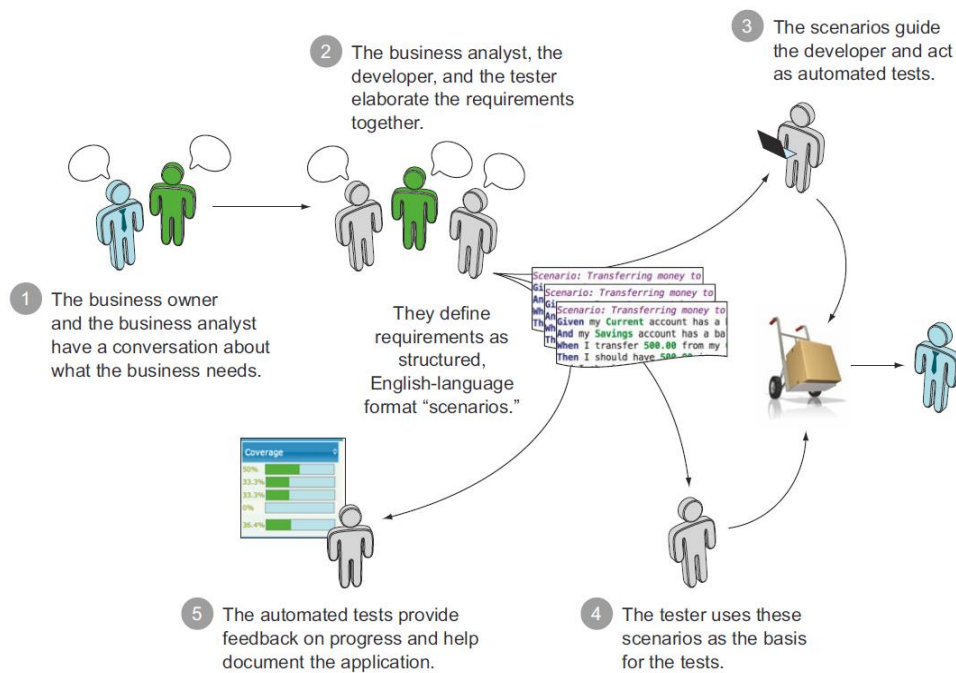


Ilustración 2. BDD usa ejemplos para guiar las conversaciones (Smart and Molak, 2023)

Con el tiempo, BDD ha evolucionado más allá del ámbito de pruebas y se ha consolidado como una práctica de diseño colaborativo orientada al descubrimiento funcional y al modelado compartido del comportamiento. Esto refuerza su rol como complemento de DDD, ya que permite capturar la lógica del negocio mediante lenguaje ubicuo y transformar los escenarios en artefactos verificables y mantenibles (Smart and Molak, 2023) Los autores señalan que el principal obstáculo en el desarrollo de software es la incertidumbre: los equipos comienzan con un conocimiento limitado del dominio

y este se clarifica progresivamente a medida que se descubren reglas, excepciones e interacciones reales.

BDD surge como una evolución de *Test-Driven Development* (TDD) (Beck, 2003; North, 2006), enfocándose en mejorar la comunicación entre perfiles técnicos y de negocio mediante descripciones del comportamiento en escenarios Gherkin (*Given–When–Then*), que funcionan como especificaciones ejecutables (Adžić, 2011) y documentación viva del sistema (Hellesoy et al., 2017). Su objetivo no es solo garantizar la verificación funcional, sino alinear el diseño con la intención del negocio. En sistemas complejos, su aplicación mejora la trazabilidad de requisitos, reduce la ambigüedad y facilita la validación continua de reglas (Irshad et al., 2021), favoreciendo la identificación de casos límite y manteniendo una relación estrecha entre la necesidad funcional y la implementación técnica.

Aunque BDD y TDD comparten raíces en el desarrollo guiado por pruebas, difieren en su nivel de abstracción: TDD busca construir el código correcto desde el interior del sistema, mientras que BDD busca construir el comportamiento correcto desde la perspectiva del usuario. Ambos enfoques se complementan: BDD guía la dirección funcional y TDD asegura la calidad técnica. La ilustración correspondiente muestra cómo se relacionan ambos ciclos durante el desarrollo.

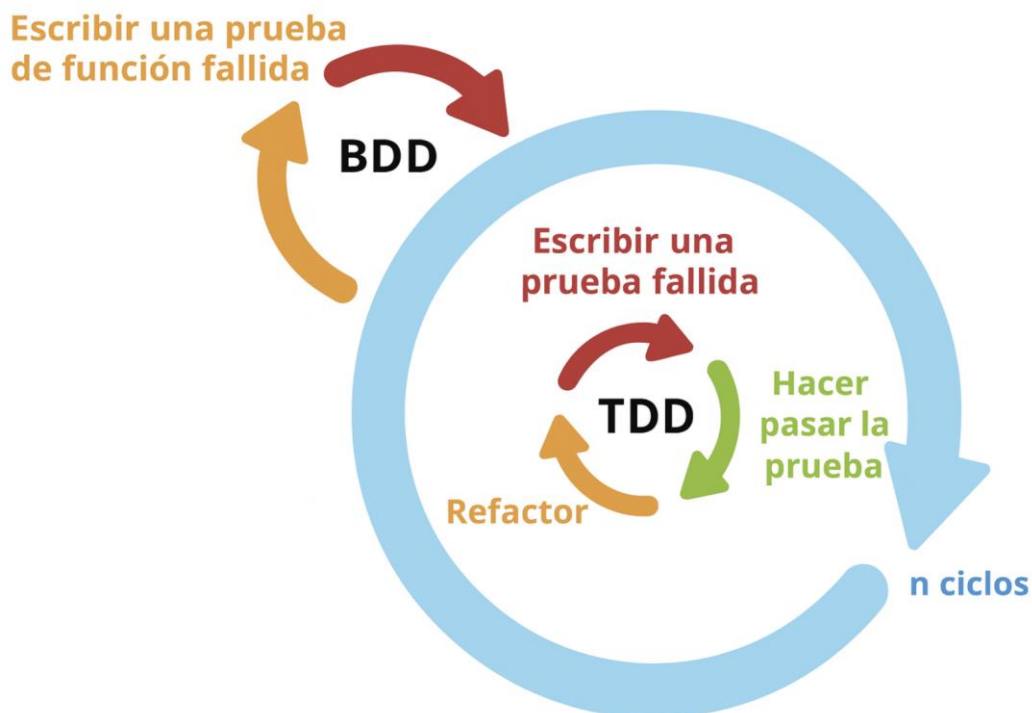


Ilustración 3. Ciclo de vida de TDD y BDD

En este sentido, BDD permite una integración trazable entre la especificación legible y las pruebas ejecutables (Solís and Wang, 2011). En conjunto, BDD es una técnica de diseño de software colaborativa que organiza ejemplos del comportamiento deseado en una especificación concreta y ejecutable, maximizando la colaboración entre todos los interesados (equipo técnico y negocio) y reduciendo malentendidos.

2.3.1. Principios fundamentales del enfoque

El núcleo de BDD son los escenarios escritos en *Gherkin*, un lenguaje semi-formal (a menudo en inglés) de tipo *Given–When–Then*. Cada historia de usuario se describe mediante escenarios que exponen un contexto inicial, una acción y un resultado esperado. Por ejemplo, un escenario de inicio de sesión podría expresarse así (en español):

```
Feature: User login

Scenario: User provides valid credentials
  Given the user has an active account
  And is on the login page
  When they enter their email address in the appropriate field
  And type the associated password in the password field
  And click the "Log in" button
  Then they should successfully log into the system
```

Ilustración 4. Escenario BDD para la historia de usuario de inicio de sesión

Estos escenarios actúan como documentación viva y como pruebas automatizadas a la vez. Tales interacciones de ejemplo se expresan en términos de dominio (lenguaje ubicuo) y son ejecutables, sirviendo tanto como especificación de requerimientos como suite de pruebas de aceptación (Binamungu et al., 2018). Así, cada escenario comunica claramente la funcionalidad deseada al equipo completo.

Para elaborar las historias y criterios de aceptación de forma colaborativa se puede emplear la técnica de *Example Mapping* (Hellesoy et al., 2017; Wynne, 2015). Este método consiste en usar tarjetas de colores para clasificar la información surgida en la discusión de una historia de usuario. Se organiza una tabla o diagrama en el que, partiendo de la historia, se identifican reglas, ejemplos y dudas hasta dejar claros los criterios. A continuación, se muestra la tabla 2, representativa de *Example Mapping* para un caso de inicio de sesión:

Historia (Usuario)	Reglas (Condiciones)	Ejemplos (Casos)	Preguntas (Dudas)
Como usuario quiero iniciar sesión para acceder a mi cuenta.	- Usuario registrado con contraseña válida. - Usuario existente con contraseña inválida. - Máximo 3 intentos antes de bloquear la cuenta.	- Juan tiene cuenta <code>juan@example.com</code> con clave correcta. - Ana tiene cuenta <code>ana@example.com</code> pero clave incorrecta.	- ¿Qué ocurre si olvido mi contraseña? - ¿Se bloquea la cuenta tras 3 intentos erróneos?

Tabla 2. *Example Mapping* para un caso de inicio de sesión

El *Example Mapping* facilita iterar rápidamente sobre requisitos, aclarar ambigüedades y preparar los escenarios *Gherkin* finales.

Especificación por comportamiento y ciclo de vida

La especificación por comportamiento es un principio de BDD que implica definir los requerimientos mediante ejemplos concretos, lo que integra el análisis, la implementación y la validación desde etapas tempranas. En la práctica, BDD cubre todo el ciclo de vida del software: desde la visión inicial del producto hasta el código y el despliegue. BDD puede abarcar todo el ciclo de vida del software, desde la visión hasta el código y el lanzamiento. Cada escenario de BDD, al estar ligado a pruebas automáticas, garantiza que el desarrollo avance conforme a la visión de negocio acordada, actuando como criterio de aceptación continuamente verificado.

2.3.2. Integración de BDD con DDD

BDD y DDD comparten la preocupación por el dominio del problema y el uso de un lenguaje ubicuo común. De hecho, BDD utiliza el concepto de lenguaje ubicuo del DDD. Ambos enfoques buscan que desarrolladores y negocio se entiendan sin ambigüedades mediante términos del dominio. En DDD, el modelo de dominio y el lenguaje ubicuo reflejan las reglas de negocio centrales; en BDD, esas reglas se expresan directamente en los escenarios. Por ejemplo, (Biswas, 2022) señala que las pruebas BDD deben centrarse en la lógica de negocio central (dominio), aislando dependencias técnicas. De este modo, BDD complementa DDD: cada escenario refleja un aspecto del comportamiento deseado de la entidad de dominio bajo ciertas condiciones. Como resultado, las *features files* de BDD actúan como documentación ejecutable del modelo de dominio, manteniendo vivo el lenguaje de negocio a lo largo del desarrollo.

Un ejemplo práctico es aplicar BDD en una arquitectura hexagonal, donde los escenarios prueban directamente los puertos y casos de uso del dominio. Esto garantiza que las pruebas solo fallen por cambios en la lógica de negocio, no por infraestructura. Además, los beneficios documentales de BDD (documentación viva) refuerzan la coherencia del lenguaje ubicuo en DDD. En definitiva, BDD y DDD se complementan: BDD utiliza el modelo de DDD para formular escenarios claros, y DDD se beneficia de los ejemplos BDD para validar que el modelo de dominio satisface los requerimientos del negocio.

2.4. Desarrollo *greenfield*

El desarrollo *greenfield* se refiere a la creación de un sistema desde cero en un entorno completamente nuevo, sin código heredado ni restricciones previas. En contraste, el desarrollo *brownfield* implica mejorar o extender un sistema existente, donde se retienen datos, procesos o configuraciones heredadas. El desarrollo *greenfield* implica partir de una pizarra en blanco, mientras que el *brownfield* requiere integrar o modificar artefactos existentes (Giardino et al., 2016). Esta libertad inicial permite elegir nuevas tecnologías y patrones arquitectónicos sin las limitaciones de la arquitectura previa, pero simultáneamente conlleva riesgos significativos, ya que todas las decisiones clave deben definirse y validarse durante el proyecto, sin guía documental previa.

2.4.1. Retos inherentes en entornos *greenfield*

Trabajar en *greenfield* conlleva alta incertidumbre y ambigüedad. Al inicio suele haber poca definición de requisitos ni evidencia concreta del dominio, por lo que los

equipos deben descubrir gradualmente las necesidades del negocio. Los entornos *greenfield* típicos (como startups) operan en un contexto caótico, en rápida evolución e incierto, con presión intensa de tiempo del mercado (Giardino et al., 2016). Esto implica que las metas de negocio pueden cambiar con rapidez y que no existe una historia operativa previa que guíe el diseño. Por ello, es común que los equipos prioricen iteraciones cortas para lanzar cuanto antes una versión MVP (producto mínimo viable) y validar su ajuste al mercado. A menudo se implementan soluciones de baja precisión inicialmente y luego se reestructura el producto conforme se obtienen nuevos aprendizajes. En resumen, los retos principales incluyen:

- Incertidumbre de requisitos: no existe un catálogo de funcionalidades heredadas, por lo que los requerimientos deben explorarse y refinarse dinámicamente. La falta de información previa y el ritmo cambiante generan un alto nivel de incertidumbre.
- Ambigüedad del alcance: la visión del producto suele ser imprecisa al inicio, lo que obliga a clarificar el dominio mediante investigación progresiva y validaciones con el usuario.
- Presión de tiempo y recursos: al no haber código existente, se suele invertir más en investigación y prototipado. Además, en startups o proyectos *greenfield* es frecuente que los recursos (equipos pequeños, poco presupuesto) sean limitados, agravando la presión para entregar rápido.
- Aprendizaje continuo: el equipo debe adaptarse constantemente conforme descubre el dominio. Esto demanda una cultura de aprendizaje iterativo, donde las iteraciones tempranas sirven para aprender y replanificar el diseño interno.

Desde el punto de vista arquitectónico, el entorno *greenfield* requiere una exploración activa del dominio. No existen manuales ni documentación previa, por lo que resulta crucial modelar el dominio con los interesados (por ejemplo, mediante talleres de *Event Storming* o entrevistas de usuario) para ir conformando el modelo de dominio. La ausencia de restricciones heredadas brinda libertad para elegir tecnologías, patrones (por ejemplo, microservicios, arquitecturas dirigidas por eventos, CQRS, etc.) y para diseñar interfaces limpias desde el inicio, pero a la vez aumenta el riesgo de decisiones incorrectas. Como advierte Booch, decisiones de diseño apresuradas pueden cristalizar en una arquitectura accidental de difícil mantenimiento. Por ello, a pesar de la libertad inicial, es recomendable adoptar criterios claros desde el comienzo: por ejemplo, definir métricas o *fitness functions* para guiar las elecciones arquitectónicas críticas.

Además, la organización del equipo debe alinearse con el dominio: según la Ley de Conway, la estructura de comunicación interna tenderá a reflejar la arquitectura resultante. En *greenfield* es común formar equipos multifuncionales cross-funcionales, cada uno centrado en un contexto de negocio, para facilitar que el diseño se alinee con las necesidades del dominio.

2.4.2. Diseño iterativo y adaptativo

En un proyecto *greenfield* la arquitectura debe construirse de forma evolutiva. En (Ford et al., 2017) definen la arquitectura evolutiva como aquella que soporta cambios incrementales y guiados en múltiples dimensiones. Esto significa que el diseño se desarrolla a través de ciclos iterativos, evaluando constantemente nuevos avances y

corrigiendo rumbo. La arquitectura evolutiva incluye tres aspectos clave: cambio incremental, funciones de fitness y acoplamiento apropiado. En la práctica, esto se traduce en construir prototipos arquitectónicos, definir métricas que representen los objetivos de calidad (las citadas funciones de *fitness*) y permitir que la estructura del sistema cambie a medida que se obtienen datos reales de desempeño y uso. De esta forma se minimiza el riesgo de encerrar prematuramente decisiones equivocadas.

Por tanto, el proceso de desarrollo *greenfield* es altamente iterativo y adaptativo. En cada iteración se implementan partes del sistema, se prueban bajo hipótesis del negocio, y luego se refina la arquitectura basándose en el feedback y el aprendizaje adquirido. La arquitectura debe ser guiada hacia el objetivo de diseño deseado, evitando soluciones temporales que degeneren la calidad. Este enfoque ágil permite gestionar la ambigüedad inicial al descubrir y validar progresivamente los componentes fundamentales del sistema.

2.4.3. Prácticas y enfoques recomendados

En entornos *greenfield* es recomendable aplicar prácticas enfocadas en explorar el dominio y validar requisitos de forma colaborativa:

- **DDD:** diseñar el sistema partiendo del modelo de dominio promovido por expertos en el negocio, identificando contextos delimitados (*bounded contexts*) que correspondan a microservicios independientes. Esto ayuda a estructurar el sistema según conceptos del dominio, aun cuando el dominio esté poco definido inicialmente. Por ejemplo, un estudio demostró que en una migración cada *bounded context* del negocio se implementó como un servicio independiente.
- **BDD y modelado colaborativo:** definir los requisitos mediante ejemplos de comportamiento en lenguaje natural facilita la comunicación entre negocio y desarrollo. Talleres como *Event Storming* o *Example Mapping* ayudan a descubrir eventos clave, flujos de trabajo y reglas de negocio desde etapas tempranas.
- **Iteración incremental:** descomponer la entrega en versiones sucesivas que implementen casos de uso concretos permite obtener retroalimentación temprana. Herramientas de pruebas automatizadas y entregas continuas (CI/CD) ayudan a verificar en cada iteración que las decisiones de diseño mantienen la calidad (por ejemplo, evitando excesivo acoplamiento o problemas de rendimiento).
- **Uso de herramientas avanzadas:** recientes investigaciones proponen técnicas basadas en inteligencia artificial para guiar la identificación de servicios en proyectos *greenfield*. Por ejemplo, GreenMicro (Bajaj et al., 2022) y GTMicro (Bajaj et al., 2024) son métodos que utilizan agrupamiento (*clustering*) y análisis semántico de casos de uso para sugerir inicialmente cómo particionar el dominio en servicios, mejorando cohesión y reduciendo acoplamientos. Estas herramientas aprovechan minería de requisitos y NLP para asistir en la definición de servicios de forma más objetiva.
- **Enfoque orientado a la mantenibilidad:** uno de los principales impulsores en arquitecturas de microservicios es la mantenibilidad (junto con la escalabilidad) (Taibi et al., 2017). Aunque su estudio se centró en migraciones, esta prioridad aplica también en *greenfield*: se debe diseñar de forma que el sistema resultante

sea fácil de modificar, mediante prácticas como alto nivel de cohesión interna, pruebas automatizadas y documentación evolutiva.

En conjunto, estas prácticas centradas en exploración de dominio, colaboración temprana con los *stakeholders* y retroalimentación continua, permiten que el equipo adapte la arquitectura de forma segura conforme se van descubriendo los requisitos (Dragičević and Bošnjak, 2019) reales, minimizando así los riesgos de construir sobre supuestos equivocados.

2.5. Arquitecturas ágiles

El Manifiesto Ágil (Beck et al., 2001) introdujo una visión del desarrollo basada en colaboración interdisciplinaria, adaptación al cambio y entrega continua de valor. Bajo esta perspectiva, la arquitectura deja de ser una fase anticipada y cerrada para convertirse en una práctica iterativa que evoluciona junto con el entendimiento funcional del sistema (Ford et al., 2017; Fowler, 2012b). En síntesis, la arquitectura ágil utiliza técnicas ágiles para sostener una estructura flexible y adaptable mediante un proceso de diseño iterativo e incremental (Dragičević and Bošnjak, 2019).

Dos de sus principios centrales son YAGNI y el diseño emergente. En YAGNI, (Fowler, 2015d) propone evitar construir funcionalidades hasta que realmente sean necesarias, promoviendo simplicidad y evitando sobre ingeniería. El diseño emergente, por su parte, sostiene que la arquitectura debe evolucionar gradualmente a medida que se abordan los requisitos de cada iteración. Sin embargo, se reconoce que, sin un mínimo de orientación inicial, este enfoque puede derivar en entropía y estructuras frágiles o desalineadas con el negocio. En este contexto surge la tensión entre diseño anticipado y diseño emergente. El *sprint zero* permite definir una base arquitectónica mínima para evitar decisiones tardías costosas (Nord and Tomayko, 2006), pero extenderlo en exceso reduce adaptabilidad. La falacia del *Big Design Up Front* (BDUF) advierte que ningún arquitecto puede prever todos los “*unknown unknowns*” (Richards and Ford, 2020), por lo que los grandes diseños anticipados suelen fracasar. El enfoque ágil propone entonces que la arquitectura evolucione continuamente. (Ford et al., 2017) formalizan esta idea mediante la arquitectura evolutiva, que incorpora mecanismos como las *fitness functions* para guiar decisiones estructurales y asegurar que la arquitectura se mantenga alineada con objetivos críticos a medida que cambia el sistema. Este enfoque no implica irresponsabilidad, sino equilibrio entre cambio rápido y rigor arquitectónico, evaluando y refinando la arquitectura regularmente. No obstante, confiar exclusivamente en la emergencia puede producir una arquitectura accidental (Booch, 2006). La clave está en tomar las decisiones necesarias en el momento oportuno, combinando anticipación y flexibilidad (Blair et al., 2010). Aunque el diseño emergente evita la sobre ingeniería, experiencias previas demuestran que también puede derivar en incremento de entropía y pérdida de alineación con los objetivos del dominio.

Capítulo 3. Enfoque metodológico

El enfoque metodológico adoptado en esta tesis se basa en *Design Science Research* (DSR) (Hevner et al., 2004; Peffers et al., 2007; Wieringa, 2014), un marco ampliamente utilizado en Ingeniería de Software para investigaciones que requieren diseñar y evaluar soluciones innovadoras a problemas complejos. A diferencia de metodologías orientadas a describir o explicar fenómenos, DSR se centra en la creación de un artefacto que responda a una necesidad real del entorno, integrando rigor científico y utilidad práctica.

La elección de DSR se justifica por la naturaleza del problema de investigación. La identificación de microservicios en proyectos ágiles *greenfield* presenta desafíos que no pueden resolverse solamente mediante análisis comparativos o modelos conceptuales: exige el desarrollo de una propuesta prescriptiva, sistemática y aplicable por equipos de ingeniería en contextos de alta incertidumbre. Según la clasificación de artefactos de (March and Smith, 1995) este tipo de contribución se corresponde con un método o proceso, una de las categorías centrales de artefactos que DSR está diseñado para producir y evaluar. En consecuencia, DSR ofrece un marco adecuado para guiar la construcción, fundamentación teórica y evaluación progresiva de este tipo de solución.

En este sentido, DSR proporciona un enfoque riguroso para estructurar el desarrollo de la propuesta, garantizando su validez conceptual y práctica. El modelo se organiza en torno a tres ciclos interdependientes:

- el ciclo de relevancia, que vincula la investigación con el problema real;
- el ciclo de rigor, que asegura el sustento teórico mediante el uso de conocimiento existente; y
- el ciclo de diseño, que guía la creación y refinamiento iterativo del artefacto.

La interacción continua entre estos ciclos permite que la solución resulte fundamentada, coherente y evaluable desde múltiples perspectivas.

Complementariamente, esta tesis incorpora una evaluación empírica basada en experimentación controlada (Runeson and Höst, 2009; Wohlin et al., 2012), lo que permite contrastar la efectividad de la propuesta frente a enfoques no estructurados, analizar su comprensión por parte de los usuarios y observar su capacidad de adaptación ante cambios de requisitos. La combinación de DSR con evaluación empírica fortalece la validez del trabajo, garantizando que la contribución final sea robusta desde el punto de vista científico y útil para la práctica profesional (Gregor and Hevner, 2013; Venable et al., 2016a).

3.1. *Design Science Research*: fundamentos y aplicación

El enfoque *Design Science Research* se centra en crear artefactos innovadores (métodos, modelos, procesos o herramientas) capaces de resolver problemas relevantes del mundo real (Hevner et al., 2004). DSR se organiza conceptualmente en tres ciclos interrelacionados (Ilustración 5), que guían la investigación:

1. **Ciclo de Relevancia:** conecta la investigación con su contexto de aplicación, asegurando que el artefacto responda a una necesidad real. Aporta los requisitos del problema, los criterios de aceptación y los escenarios de uso.
2. **Ciclo de Rigor:** vincula la investigación con el cuerpo de conocimiento existente, integrando teorías, modelos y trabajos previos que fundamentan y guían el diseño del artefacto.
3. **Ciclo de Diseño:** constituye el corazón del proceso DSR: construcción iterativa del artefacto, validación de su utilidad y refinamiento progresivo en función del feedback obtenido.

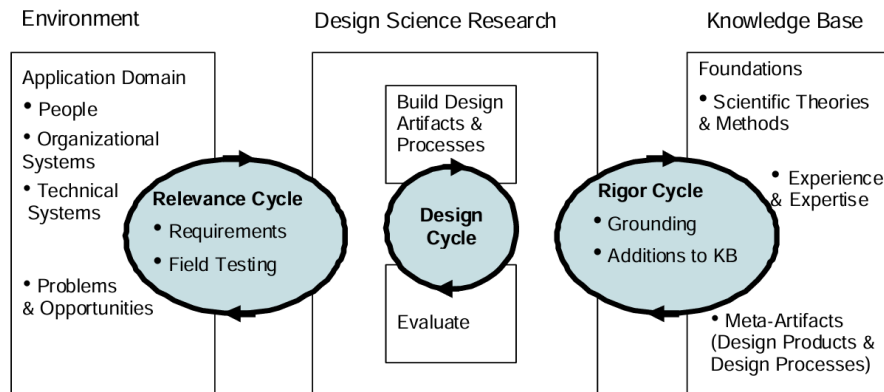


Ilustración 5. Ciclos DSR (Hevner, 2007)

Es importante destacar que el gráfico de DSR no representa un orden secuencial, sino una interacción iterativa y continua entre los tres ciclos (Hevner, 2007). Cada uno alimenta y condiciona a los demás: la construcción del artefacto se nutre de la teoría (rigor), responde a un problema real (relevancia) y evoluciona mediante evaluación constante.

3.2. Extensión metodológica: ciclo de consolidación

Si bien el modelo clásico de DSR integra diseño, construcción y evaluación en un único ciclo, esta tesis incorpora una fase adicional denominada ciclo de consolidación, siguiendo lineamientos contemporáneos de (Gregor and Hevner, 2013; Venable et al., 2016a). Este ciclo permite profundizar la validación del artefacto mediante casos de estudio y experimentos controlados, sistematizar la recolección de evidencia, realizar análisis crítico y documentar ajustes iterativos. Se concibe como un proceso reflexivo de refinamiento científico orientado a garantizar la utilidad, aplicabilidad y sostenibilidad de la propuesta en contextos ágiles reales.

3.3. Estructura del proceso DSR en la tesis

Aunque el proceso real de investigación se desarrolló de manera iterativa, en línea con el modelo de los tres ciclos, la presentación en la tesis sigue la convención narrativa recomendada (Peffer et al., 2007), organizada en cuatro capítulos principales:

- **Ciclo de Relevancia: capítulo 4**
 - Análisis del contexto (industria y literatura), mapeo sistemático y experimento inicial que permiten caracterizar el problema.
- **Ciclo de Rigor: capítulo 5**

- Revisión crítica del estado del arte y trabajos relacionados, identificación de vacíos y fundamentación conceptual de la propuesta.
- **Ciclo de Diseño: capítulos 6 y 7**
 - Construcción detallada del artefacto BDMA, sus fases, actividades y artefactos, seguido de un caso de aplicación que demuestra su funcionamiento.
- **Ciclo de Consolidación: capítulo 8**
 - Evaluación exploratoria mediante experimento pedagógico, rúbricas e instrumentos perceptuales que permiten analizar la efectividad y utilidad del artefacto.

Este orden narrativo organiza la argumentación de manera lineal para facilitar la lectura, mientras que el proceso subyacente sí refleja la interacción iterativa entre relevancia, rigor y diseño.

Capítulo 4. Ciclo de relevancia

En coherencia con el ciclo de relevancia de *Design Science Research*, esta sección examina el contexto que motiva el desarrollo de la metodología BDMA.

4.1. Análisis del contexto

El análisis combina evidencia proveniente de distintas fuentes con el objetivo de comprender en profundidad las limitaciones actuales y la naturaleza del problema. En primer lugar, se presenta un mapeo sistemático que caracteriza el estado del arte y revela la fragmentación de enfoques existentes para identificar microservicios en entornos ágiles *greenfield*. En segundo lugar, se expone un experimento inicial que permite observar cómo los equipos afrontan estas decisiones sin contar con una guía metodológica explícita. La convergencia de estos insumos —literatura y evidencia experimental— conforma el fundamento empírico y conceptual que sostiene la identificación formal del problema desarrollada en la sección 4.2.

4.1.1. Mapeo sistemático

Como primer paso del análisis del contexto, se realizó un mapeo sistemático (Battaglia et al., 2024) de la literatura con el objetivo de obtener una visión integral del estado del arte en identificación de microservicios en entornos ágiles *greenfield*. Este mapeo constituye la base conceptual del ciclo de relevancia, ya que permite determinar qué problemas han sido resueltos, cuáles persisten y en qué medida la literatura actual ofrece guías metodológicas aplicables a escenarios de construcción desde cero. Asimismo, habilita comparar enfoques, identificar patrones comunes y reconocer vacíos metodológicos que justifican el desarrollo de un nuevo marco como BDMA.

Para definir el contexto del mapeo, se identificaron 5110 artículos entre 2022 y 2024 y luego se redujo a 194, asegurando su relevancia, calidad y aporte al tema de estudio. Los criterios aplicados son los siguientes:

- **Inclusión:** Se incluyen estudios que discutan la identificación de microservicios, que proporcionen enfoques y estrategias. Investigaciones que aborden metodologías ágiles, centrándose en cómo afectan la identificación y diseño de microservicios. Artículos publicados en los últimos dos años y publicaciones en inglés.
- **Exclusión:** No se incluyen estudios que no hayan sido revisados por pares, ya que estos no ofrecen la misma garantía de calidad y publicaciones que no aborden el contexto de empresas de tecnología, dado que el enfoque del estudio es específico para este tipo de empresas.

Asimismo, se realizó un filtrado sobre el título y resumen para eliminar los que no eran relevantes. A continuación, describimos las preguntas de investigación que motivaron el estudio:

- P1: ¿Cuáles son las estrategias más comunes para identificar microservicios en empresas con arquitecturas complejas?
- P2: ¿Qué técnicas se utilizan para definir los límites de los microservicios en metodologías ágiles?

- P3: ¿Cuáles son los desafíos y soluciones reportados en la literatura sobre la implementación de microservicios en metodologías ágiles?
- P4: ¿Cómo se aplica la teoría en la práctica dentro de una empresa?
- P5: ¿En qué idiomas y en qué países se produce la investigación sobre descubrimiento de microservicios?

El trabajo realizado revela una diversidad de enfoques y técnicas para identificar y definir los límites de los microservicios, adaptándose a un entorno de desarrollo iterativo e incremental (Ünlü et al., 2024).

El principal enfoque se da a partir del uso de DDD y la creación de modelos de dominio que reflejan la lógica del negocio. Otro enfoque es aprovechando el modularidad de arquitecturas SOA, centrándose en unidades funcionales del negocio. Por último, algunos autores proponen la utilización de *Event Storming* para mapear eventos de negocio y descubrir potenciales MS. Por otro lado, diversos estudios analizan técnicas para descomposición impulsada por dominio, en la que se aprovecha el conocimiento del negocio para identificar límites naturales así también como técnicas de descomposición basadas en análisis de código para identificar módulos independientes. La aplicabilidad práctica de estas técnicas en entornos empresariales reales sugiere que, aunque la teoría proporciona una base sólida, la implementación exitosa de microservicios requiere una adaptación continua y un enfoque colaborativo para abordar los cambios frecuentes en los requisitos. Otra conclusión relevante es que no existe gran cantidad de estudios que versen sobre la creación de arquitecturas basadas en MS desde su concepción ya que, en general, trabajan sobre la descomposición de monolitos previos.

4.1.2. Experimento inicial

Con el objetivo de complementar la evidencia obtenida en el mapeo sistemático y observar cómo se manifiestan en la práctica las dificultades reportadas por la literatura, se llevó a cabo un primer experimento empírico (Battaglia, 2025). Este experimento tuvo como propósito analizar cómo distintos equipos abordan la identificación de microservicios cuando no disponen de una metodología estructurada, así como evaluar el impacto que tienen los cambios de requisitos en la estabilidad del diseño. Al situar a los participantes en un escenario realista de análisis y partición de un dominio complejo, este estudio permite identificar patrones de error, sesgos comunes y limitaciones prácticas que no siempre emergen en la revisión bibliográfica. Los resultados obtenidos sirven como línea de base para comprender el problema y justifican la necesidad de explorar enfoques metodológicos más integrados, como el propuesto posteriormente por BDMA.

Participaron 26 estudiantes de posgrado divididos en 10 equipos, organizados en dos grupos: el Grupo A, que resolvió el problema sin seguir una metodología específica, y el Grupo B, que utilizó metodologías ágiles y *Domain-Driven Design* (DDD). El trabajo práctico consistió en identificar los MS necesarios para un escenario de gestión de manufactura, seguido de una segunda etapa en la que se introdujeron nuevos requisitos para analizar el impacto del cambio según el enfoque aplicado inicialmente. En cuanto a la demografía, el 68.5% de los estudiantes no tenían un posgrado finalizado, mientras que el 77% contaban con 4 o más años de experiencia en desarrollo de software. El 65% tenía al menos 3 años de experiencia con metodologías ágiles, y el 57% había trabajado

con MS, aunque solo el 35% acumulaba 3 o más años de experiencia en esta área. Respecto a BDD, únicamente el 35% poseía experiencia previa.

Al finalizar el experimento, se encuestó a los alumnos participantes, obteniendo los siguientes resultados: el 97% calificó el enunciado del problema como complejo o muy complejo. Respecto al nuevo requisito, el 57% tuvo que refactorizar su trabajo inicial, y el 87% lo consideró complejo o muy complejo. Entre quienes no usaron metodología, solo el 36% refactorizó, mientras que el 50% de los que usaron DDD y metodologías ágiles rediseñaron su trabajo. El análisis destaca que combinar un diseño inicial ligero con metodologías ágiles es clave para sistemas basados en microservicios (MS). Aunque el diseño inicial no elimina la necesidad de ajustes, facilita la identificación de límites contextuales y dominios clave, optimizando la granularidad y reduciendo riesgos. Los datos sugieren que el enfoque ágil y DDD requiere más refactorización, pero ofrece un marco sólido que equilibra planificación y adaptabilidad, reduciendo la deuda técnica y mejorando escalabilidad. Los resultados preliminares confirmaron la brecha identificada

4.2. Identificación del problema

A partir de la revisión de la literatura y de los resultados del experimento inicial, se identifican una serie de desafíos persistentes en el diseño de arquitecturas de microservicios en contextos ágiles *greenfield*. El crecimiento de este estilo arquitectónico ha transformado la forma de construir sistemas distribuidos, promoviendo escalabilidad, mantenibilidad y despliegue independiente de componentes (Newman, 2021; Richardson, 2018). Sin embargo, determinar límites de servicio adecuados sigue siendo una decisión crítica: una partición incorrecta impacta directamente en la cohesión interna, el acoplamiento entre servicios y la capacidad de evolución del sistema (Fowler, 2015c; SAID et al., 2024). En proyectos *greenfield* ágiles, donde el dominio es aún incierto y los requisitos cambian con rapidez, este problema se profundiza, especialmente al intentar definir límites coherentes con el negocio en contextos dinámicos.

La literatura muestra que este desafío ha sido abordado de manera fragmentada desde distintas perspectivas: minería de sistemas legados, modelado estructurado, heurísticas específicas o enfoques *model-driven*, pero sin consolidarse en metodologías operativas que integren de forma sistemática prácticas de DDD y BDD en proyectos *greenfield*. Como consecuencia, en la práctica los equipos suelen enfrentar (1) dificultades para delimitar microservicios de forma adecuada en ciclos iterativos, (2) ausencia de trazabilidad explícita entre los escenarios de comportamiento (requisitos) y las decisiones arquitectónicas, lo que debilita la validación y la adaptación continua, y (3) carencia de un proceso reproducible que conecte la especificación funcional con el diseño de microservicios, más allá de recomendaciones teóricas aisladas. Esta brecha metodológica se traduce en que los escenarios BDD identificados no se proyectan de manera sistemática en la arquitectura, produciendo soluciones *ad-hoc* o excesivamente rígidas, con efectos negativos sobre la mantenibilidad y coherencia de las arquitecturas de microservicios en entornos ágiles (Battaglia et al., 2025). Esta situación se ve respaldada por revisiones sistemáticas recientes. En (Schmidt and Thiry, 2020) analizan 27 estrategias de identificación de microservicios y evidencian una gran dispersión de enfoques, clasificados según sus entradas (requisitos, código, procesos), técnicas (*clustering*, análisis semántico, transformaciones modelo-código) y niveles de automatización. Los autores destacan que la aplicación sistemática de DDD sigue siendo

escasa y que son pocos los trabajos que combinan estrategias orientadas al dominio con procesos estructurados y validados. Este vacío confirma la necesidad de propuestas que, como BDMA, articulen principios colaborativos (BDD), modelado semántico y ciclos iterativos de validación funcional. Si bien DDD es reconocido como práctica recomendada para enfrentar este reto, su aplicación tradicional presenta tensiones metodológicas con los principios ágiles, debido a su énfasis más bien estructural y anticipado.

Un segundo aspecto problemático se vincula con la evolución de los requisitos y su impacto en el diseño arquitectónico. Aunque numerosos estudios abordan la gestión del cambio desde la perspectiva ágil, se observa una falta de discusión profunda sobre las implicancias de esa evolución en la arquitectura de microservicios y sobre las consecuencias de tomar decisiones estructurales en momentos inadecuados. Esto incluye comprender cuándo se viola el principio *“You Aren’t Gonna Need It”* (Fowler, 2015d) o se incurre en la falacia *del Big Design Up Front* (M. Waterman et al., 2015; Zimmermann, 2017).

En este contexto, las arquitecturas ágiles se proponen como una vía de conciliación entre agilidad y arquitectura (M. G. Waterman, 2014), reduciendo el riesgo asociado a dedicar demasiado esfuerzo anticipado al diseño arquitectónico sin evidencia suficiente. DDD se ha consolidado como estrategia recomendada para alinear el diseño del sistema con el conocimiento del dominio mediante patrones como *bounded contexts*, entidades, agregados y eventos del dominio. No obstante, su aplicación rigurosa suele requerir decisiones arquitectónicas anticipadas, lo que puede entrar en tensión con la filosofía de las metodologías ágiles, que promueven diseño emergente, iteración rápida y entrega temprana de valor (Beck and Fowler, 2001; Booch, 2006; Highsmith, 2002; Kruchten, 2010). Investigaciones recientes advierten que la falta de guías metodológicas prácticas que articulen DDD con entornos ágiles representa una barrera para su adopción efectiva (Battaglia, 2025; Battaglia et al., 2024; Zhong et al., 2024). Ante esta limitación, algunos autores proponen tratar la arquitectura como una hipótesis sujeta a validación y refactorización continuas (Ford et al., 2017; Zimmermann, 2017), lo que demanda enfoques colaborativos que combinen exploración funcional con diseño evolutivo.

La noción de arquitectura ágil se alinea precisamente con esta idea: las decisiones arquitectónicas deben formularse como hipótesis que se validan iterativamente, en lugar de estructuras rígidas definidas desde el inicio (Zimmermann, 2017). Las decisiones clave deberían tomarse en el momento más informado posible, cuando el riesgo de error es menor (Ford et al., 2017). Esta forma de concebir la arquitectura exige competencias metodológicas avanzadas y prácticas que equilibren rigor técnico con flexibilidad organizacional. En ausencia de lineamientos sistemáticos, el diseño puede derivar en arquitecturas accidentales o en la imposición de estructuras inadecuadas (Booch, 2006).

En este contexto, la integración de DDD con técnicas de exploración funcional como BDD se perfila como una vía prometedora para acompañar la evolución del sistema. BDD aporta una forma concreta de expresar comportamientos esperados desde el negocio mediante escenarios estructurados, mientras que DDD ofrece el marco estratégico para traducir ese conocimiento en límites y patrones arquitectónicos.

En los últimos años, diversos enfoques han intentado abordar el desafío de identificar límites de microservicios en contextos ágiles. El uso de DDD ha sido recomendado para delimitar microservicios (Battaglia, 2025; Sangabriel-Alarcón et al., 2025; Vernon, 2013), pero otros estudios (Jamshidi et al., 2018; Taibi and Lenarduzzi, 2018) advierten que la falta de conocimiento suficiente del dominio en etapas tempranas puede conducir a segmentaciones incorrectas y otros problemas: (1) se requiere un alto nivel de madurez organizacional y técnica, con fuerte participación del negocio; (2) la transición desde modelos de dominio a servicios desplegados no está completamente formalizada; y (3) los enfoques tradicionales de DDD no se adaptan fácilmente a equipos ágiles distribuidos ni a desarrollos *greenfield* sin base de código, en los que los requisitos evolucionan rápidamente.

Otros trabajos refuerzan esta diagnosis. (Oumoussa and Saidi, 2024) muestran que muchas propuestas asumen la existencia de sistemas previos (*brownfield*) o modelos de dominio bien establecidos, lo que limita su aplicabilidad en escenarios *greenfield*, donde el conocimiento del dominio se construye progresivamente. En (Zhong et al., 2024) señalan que, aunque DDD es ampliamente utilizado, carece de guías prácticas para su implementación iterativa bajo condiciones de incertidumbre. Un mapeo sistemático reciente (Battaglia et al., 2024) confirma que la mayoría de las estrategias analizadas no articulan de forma explícita prácticas de descubrimiento funcional con modelado arquitectónico. La falta de integración entre herramientas como BDD y DDD ha sido identificada como una de las causas de baja trazabilidad entre requerimientos y límites arquitectónicos. En (Su et al., 2024) aportan evidencia empírica adicional: decisiones desacertadas en el diseño de microservicios han obligado incluso a revertir migraciones debido a particiones poco sostenibles. Asimismo, (Kapferer and Zimmermann, 2020) advierten que, sin un lenguaje compartido ni prácticas colaborativas de modelado, el diseño tiende a fragmentarse y a generar artefactos inconsistentes.

Estudios empíricos recientes refuerzan esta preocupación respecto a la adopción aislada de DDD o BDD. En (Hippchen et al., 2017) documentan la aplicación de DDD en una arquitectura de microservicios, observando beneficios en la definición de contextos, pero también dificultades para alinear equipos en torno al lenguaje ubicuo. Por otro lado, en (Irshad et al., 2021) evalúan la aplicación de BDD en sistemas de gran escala y concluyen que, aunque mejora la trazabilidad, su adopción exige entrenamiento sostenido y fuerte involucramiento del negocio. Ambos trabajos coinciden en que la adopción de estas prácticas de manera aislada limita su efectividad. Esto refuerza la hipótesis de que una integración explícita entre descubrimiento funcional (BDD) y modelado estratégico (DDD) puede ser más eficaz que su uso independiente.

Por último, las metodologías ágiles priorizan la entrega temprana y continua de valor (Beck et al., 2001) y, en este marco, las decisiones arquitectónicas deben tomarse de forma progresiva (Fowler, 2015c). La tensión entre diseño emergente y decisiones estructurales (como la partición en microservicios) y se agudiza en proyectos *greenfield*, donde no hay legado, pero tampoco certezas sobre el dominio completo (Kruchten, 2010). Algunos trabajos han propuesto enfoques híbridos o prácticas exploratorias como *Event Storming*, pero sin articular completamente la trazabilidad entre requisitos funcionales y decisiones arquitectónicas. En síntesis, el estado del arte muestra la necesidad de un enfoque metodológico que conecte de manera sistemática el descubrimiento funcional basado en ejemplos (BDD) con el diseño estratégico y evolutivo de microservicios guiado por el dominio (DDD), mediante procesos iterativos,

trazables y sostenibles. La tabla comparativa que se presenta a continuación sintetiza las principales características de los enfoques revisados y posiciona la propuesta integrada de esta tesis frente a ellos.

Para facilitar la comprensión de los enfoques revisados anteriormente, se sintetizan sus principales características en la siguiente tabla comparativa. La tabla contrasta los enfoques tradicionales de diseño de microservicios, como DDD aplicado de forma anticipada y las divisiones orientadas a equipos o el uso aislado de BDD, con la propuesta integrada de este trabajo. Esta comparación permite identificar claramente cómo la combinación de DDD y BDD en un marco iterativo colaborativo mejora aspectos críticos como la trazabilidad, la adaptabilidad y el alineamiento con el negocio.

Enfoque	Ventajas	Limitaciones
DDD tradicional	Visión estratégica del dominio	Requiere modelado anticipado
BDD aislado	Alta trazabilidad funcional	No considera estructura técnica o modularización
Event Storming	Descubrimiento visual e inclusivo	No define arquitectura ni límites formales
Propuesta integrada DDD + BDD	Diseño emergente, colaborativo y trazable	Requiere cultura organizacional madura

Tabla 3. Resumen de ventajas y desventajas de los diferentes enfoques

Ambas aproximaciones (DDD y BDD) se fundamentan en procesos altamente colaborativos que, si bien fomentan el alineamiento contextual, pueden carecer de mecanismos sistemáticos para validar o sugerir límites arquitectónicos cuando el dominio es amplio, ambiguo o distribuido.

Pese a los avances, la literatura muestra una falta de propuestas que combinen explícitamente la expresividad funcional de BDD con el modelado estratégico de DDD para habilitar una arquitectura emergente. En resumen, mostramos una tabla con los vacíos identificados

Categoría	Vacío Observado
Metodológico	No existen <i>frameworks</i> que combinen sistemáticamente BDD + DDD para delimitar microservicios en contextos ágiles.
Práctica ágil	Falta de guías concretas para usar escenarios de BDD como herramienta de diseño arquitectónico colaborativo.
<i>Greenfield</i>	La mayoría de las propuestas asumen existencia de código o servicios previos.
Participación interdisciplinaria	No hay mecanismos formales para integrar usuarios y stakeholders no técnicos en el descubrimiento de límites de servicios.
Validación empírica	Escasa evidencia empírica sobre la efectividad de las técnicas de identificación de microservicios en entornos ágiles reales.

Tabla 4. Categorización y resumen de los vacíos observados

En resumen, podemos plantear los siguientes problemas que motivan esta tesis

- Dificultad para identificar límites correctos de microservicios: la definición de los límites de un microservicio es una decisión arquitectónica crítica y altamente sensible. En entornos *greenfield*, donde no existe un sistema previo que funcione

como referencia, los equipos suelen depender de intuiciones técnicas o de interpretaciones subjetivas del dominio. Esto conduce con frecuencia a servicios mal segmentados, solapados o excesivamente granulares, afectando la cohesión interna, incrementando el acoplamiento y generando dificultades posteriores en la evolución del sistema.

- Falta de integración metodológica entre DDD y enfoques ágiles: DDD aporta un marco conceptual sólido para delimitar el dominio, pero su aplicación estricta requiere un modelado anticipado que entra en tensión con la naturaleza iterativa y emergente de los métodos ágiles. Por ello, en la práctica, los equipos no logran integrar adecuadamente el modelado del dominio dentro de los ciclos cortos de trabajo, provocando que DDD quede relegado, mal aplicado o desconectado del flujo ágil.
- Ausencia de trazabilidad entre requisitos funcionales y decisiones arquitectónicas: una de las mayores brechas en el diseño de microservicios es la falta de un mecanismo que conecte explícitamente el comportamiento funcional esperado con las decisiones arquitectónicas resultantes. Sin trazabilidad, los equipos pierden la capacidad de justificar por qué un servicio existe, cómo se comporta, o qué eventos y reglas motivaron su creación. Esto complica la validación, el mantenimiento y la gobernanza técnica a lo largo del tiempo.
- Riesgo de arquitectura accidental en proyectos *greenfield*: al no contar con información consolidada del dominio ni con evidencia funcional suficiente, los equipos suelen adoptar decisiones tempranas que quedan cristalizadas incluso cuando el dominio evoluciona. Esto deriva en una arquitectura accidental, en la que los componentes no reflejan el negocio real sino decisiones improvisadas tomadas en etapas iniciales y difíciles de revertir posteriormente.
- Carencia de guías prácticas para aplicar DDD de manera iterativa: aunque DDD es ampliamente recomendado para diseñar microservicios, la literatura propone lineamientos conceptuales, pero ofrece escaso soporte para una aplicación pragmática y sostenible dentro de ciclos ágiles. Los equipos carecen de herramientas que indiquen cómo derivar *bounded contexts*, cómo reaccionar a cambios, o cómo vincular patrones estratégicos de DDD con artefactos funcionales del *backlog*.
- Escasa participación del negocio y débil construcción del lenguaje ubicuo: el diseño arquitectónico suele quedar en manos del equipo técnico, con poca intervención de usuarios o expertos del dominio. Esto limita la riqueza semántica del modelo, genera inconsistencias conceptuales y favorece interpretaciones divergentes sobre reglas de negocio. La falta de un lenguaje ubicuo compartido profundiza la desconexión entre requerimientos y arquitectura.
- Dificultad para adaptar la arquitectura ante cambios del dominio: en sistemas complejos y dinámicos, los requisitos cambian de manera frecuente. Sin embargo, las arquitecturas tradicionales carecen de mecanismos formales para incorporar esos cambios sin romper la estructura existente. La falta de estrategias de refactorización arquitectónica contribuye a generar deuda técnica, servicios rígidos y una pérdida progresiva de coherencia.
- Fragmentación metodológica y abordajes aislados: la literatura presenta múltiples enfoques para identificar microservicios, pero la mayoría se centran en

partes aisladas del problema: heurísticas de dominio, análisis estructural, herramientas algorítmicas o prácticas ágiles desconectadas. Esta fragmentación metodológica conduce a que los equipos combinen técnicas de forma intuitiva, obteniendo resultados inconsistentes y poco reproducibles.

- Resultados inconsistentes en equipos heterogéneos: en organizaciones donde los equipos tienen niveles variados de experiencia, el diseño arquitectónico suele depender fuertemente del arquitecto más experimentado. Sin procesos estandarizados, los resultados son altamente variables: el mismo problema puede producir arquitecturas diferentes, contradictorias o de baja calidad según quién lo diseñe.
- Falta de mecanismos formales para fundamentar y registrar decisiones arquitectónicas: la mayoría de las decisiones arquitectónicas se toman verbalmente o quedan dispersas en documentos informales. Esto dificulta comprender por qué un servicio fue creado, qué alternativas se analizaron o cómo se justificó un patrón de integración. Sin documentación estructurada, los equipos nuevos pierden conocimiento histórico y enfrentan mayores riesgos de introducir inconsistencias.

Podemos, entonces, resumir el problema identificado con la siguiente pregunta de investigación que nos guiará en el desarrollo de la propuesta:

¿Cómo diseñar una metodología iterativa que permita identificar microservicios en contextos ágiles greenfield, integrando prácticas de descubrimiento funcional como BDD con principios estratégicos de DDD, y que asegure la trazabilidad funcional y la evolución arquitectónica coherente del sistema?

Capítulo 5. Ciclo de rigor

5.1. Trabajos Relacionados

A continuación, describimos diferentes trabajos que se relacionan con esta propuesta. En el anexo C, podrá ver una tabla comparativa de cada trabajo con la propuesta de esta tesis.

5.1.1. Partitioning microservices: A domain engineering approach

El enfoque propuesto por los autores (Josélyne et al., 2018) se basa en aplicar los principios de la ingeniería de dominios para abordar la identificación y segmentación de microservicios. La metodología parte de la elaboración de un modelo de dominio funcional, a partir del cual se busca identificar características comunes y variaciones dentro del sistema, empleando técnicas de análisis de línea de productos de software (SPL). El objetivo es encontrar una modularización adecuada que sirva como base para construir microservicios alineados con la estructura funcional del negocio. Desde la perspectiva conceptual, el aporte central de este enfoque radica en la reutilización sistemática de modelos de dominio como insumo para guiar decisiones arquitectónicas. Esta estrategia es especialmente útil en contextos donde se diseñan múltiples sistemas con variaciones similares, como en plataformas configurables o familias de productos. Sin embargo, al analizarlo desde la perspectiva de entornos ágiles *greenfield* (el foco de esta tesis), surgen limitaciones significativas. En primer lugar, la propuesta asume la existencia de una arquitectura de línea de productos o una visión anticipada del dominio, lo cual resulta poco compatible con procesos iterativos, exploratorios y colaborativos como los que se promueven en contextos de diseño emergente. En segundo lugar, el enfoque carece de mecanismos explícitos para garantizar la trazabilidad funcional entre requerimientos expresados por usuarios y servicios identificados, y tampoco contempla herramientas de validación del diseño desde el comportamiento del sistema.

A diferencia de esta propuesta, BDMA parte del comportamiento observable y narrativas funcionales expresadas en escenarios BDD, integrando técnicas de análisis semántico para guiar la identificación de contextos. Además, BDMA prioriza la validación continua y la refactorización estructural como parte del ciclo de diseño, elementos que no se abordan en el enfoque de ingeniería de dominios.

En términos de los criterios de evaluación en *Design Science Research* (DSR), la propuesta exhibe claridad en el problema, rigor en el diseño teórico y alineación con prácticas de modelado funcional, pero presenta una evaluación limitada y una aplicabilidad restringida a contextos no iterativos. Su utilidad como antecedente metodológico reside principalmente en su formalización estructurada del dominio, pero no ofrece una solución viable ni adaptable para ciclos ágiles con descubrimiento progresivo del conocimiento.

5.1.2. GreenMicro: identifying microservices from use cases in greenfield development

Uno de los trabajos más relevantes en el área de identificación automatizada de microservicios en contextos *greenfield* es la propuesta GreenMicro (Bajaj et al., 2022). Este enfoque combina modelos de casos de uso estructurados con técnicas de programación genética para derivar una segmentación óptima del sistema en microservicios. El proceso incluye el análisis de funcionalidades desde modelos UML, la generación de múltiples alternativas de partición mediante operadores evolutivos, y su evaluación en función de métricas arquitectónicas como cohesión, acoplamiento y tamaño balanceado.

Desde la perspectiva metodológica, GreenMicro representa un avance significativo en la automatización del diseño de servicios, particularmente cuando se dispone de una especificación formal y completa del sistema. Sin embargo, al analizar su aplicabilidad en entornos ágiles y de descubrimiento progresivo, se identifican limitaciones clave que restringen su adopción como estrategia de diseño iterativo.

En primer lugar, el enfoque parte de una lógica cerrada y estructurada del conocimiento del dominio, basada en casos de uso UML previamente definidos. Esto lo hace poco adecuado para contextos donde las funcionalidades emergen gradualmente a través de exploración, conversaciones y escenarios funcionales, como sucede en entornos ágiles *greenfield*. En contraposición, BDMA adopta un enfoque semántico, iterativo y colaborativo, donde la identificación de contextos y servicios se construye desde escenarios redactados con el negocio en lenguaje BDD y se guía por el lenguaje ubicuo.

En segundo lugar, GreenMicro no contempla la validación progresiva del diseño ni su evolución semántica, ya que la segmentación se realiza como un proceso de diseño inicial orientado a la optimización algorítmica. Esto contrasta con el enfoque de BDMA, que incorpora la validación funcional continua mediante especificación ejecutable y la posibilidad de refactorización arquitectónica a partir del comportamiento observado.

Finalmente, aunque GreenMicro aporta rigor en la aplicación de métricas estructurales, su alineación con los objetivos funcionales y semánticos del sistema es débil. BDMA, por su parte, privilegia la trazabilidad funcional, la participación de *stakeholders* y la alineación con el dominio como criterios centrales de calidad del diseño.

En resumen, si bien GreenMicro ofrece un marco valioso para la automatización de decisiones arquitectónicas en entornos bien especificados, su integración directa en una metodología como BDMA resultaría incompatible con los principios colaborativos, iterativos y semánticos que esta promueve. Por ello, se considera como un antecedente relevante a efectos comparativos, pero no como un componente integrable del marco propuesto.

5.1.3. GTMicro—microservice identification approach based on deep NLP transformer model for greenfield developments

Un aporte reciente es GTMicro (Bajaj et al., 2024), un enfoque orientado a proyectos *greenfield* que utiliza modelos de lenguaje basados en Transformers (BERT) para calcular

la similitud semántica entre descripciones de casos de uso, agrupándolos luego mediante *clustering* jerárquico para derivar candidatos a microservicios. El proceso parte de artefactos de requisitos (use cases), los convierte en *embeddings* vectoriales y genera una matriz de similitud sobre la cual se aplica una métrica de distancia y un algoritmo de agrupamiento. El enfoque es validado utilizando métricas arquitectónicas como Structural Modularity, Inter-Partition Call Percentage, Business Context Purity e Interface Number, mostrando mejoras sobre enfoques previos como Mono2Micro, CoGCN, FoSCI y MEM.

A diferencia de métodos colaborativos o guiados por el dominio, GTMicro se apoya de manera casi exclusiva en análisis textual automatizado y no incorpora participación del negocio ni mecanismos de validación funcional. Si bien constituye uno de los avances más relevantes en identificación automática de microservicios en contextos *greenfield*, su dependencia de artefactos textuales estructurados y su falta de trazabilidad hacia reglas de negocio limita su aplicabilidad en entornos ágiles. En este sentido, BDMA se posiciona como un enfoque complementario, que prioriza la comprensión del dominio, la co-creación de escenarios y la evolución controlada de la arquitectura.

5.1.4. Microservices backlog: A genetic programming technique for identification and evaluation of microservices from user stories

El trabajo propone un enfoque centrado en la identificación y evaluación de microservicios a partir de historias de usuario mediante un modelo denominado *Microservices Backlog* (Vera-Rivera et al., 2020). Esta propuesta se destaca por introducir una técnica de optimización basada en programación genética, que busca encontrar la segmentación más adecuada de funcionalidades en microservicios, guiada por métricas de calidad arquitectónica como cohesión, acoplamiento y granularidad.

Una de sus principales contribuciones es el intento de formalizar el análisis de historias de usuario como entrada funcional, lo cual representa un avance frente a propuestas que requieren modelos técnicos previos (como UML). A través de un proceso automatizado, el modelo genera configuraciones de microservicios y selecciona la más prometedora según los criterios definidos. Esto lo posiciona como un enfoque híbrido entre ingeniería de requisitos y diseño arquitectónico, con soporte cuantitativo.

Sin embargo, al analizarlo desde el marco de esta tesis, se identifican limitaciones conceptuales y metodológicas relevantes. En primer lugar, si bien parte de insumos funcionales (*user stories*), el proceso carece de un mecanismo de validación semántica o colaborativa con los actores del dominio, lo que dificulta garantizar que los servicios reflejen correctamente el comportamiento esperado. Además, el enfoque no promueve una construcción progresiva ni validación iterativa, sino que produce una segmentación óptima en un solo paso, sin soporte para la evolución funcional o la refactorización estructural.

Por el contrario, la propuesta BDMA no solo parte de narrativas funcionales colaborativas (escenarios BDD), sino que promueve la construcción de un lenguaje ubicuo, la validación continua mediante pruebas ejecutables y la iteración arquitectónica basada en feedback funcional. Así, BDMA responde no solo a los aspectos estructurales del diseño, sino también a la necesidad de mantener una trazabilidad

funcional completa y adaptabilidad al cambio, aspectos no cubiertos en el enfoque de Vera-Rivera.

Desde la perspectiva de DSR, el trabajo cumple con criterios como novedad técnica, evaluación empírica (mediante experimentación controlada), y contribución al conocimiento aplicado, pero presenta debilidades en los criterios de relevancia contextual para entornos ágiles y validación funcional del diseño.

5.1.5. Designing microservice-based applications by using a domain-driven design approach

El trabajo explora la combinación de DDD y BDD como marco conceptual para diseñar arquitecturas de microservicios. La propuesta se centra en cómo estas dos perspectivas pueden complementarse: DDD como base para modelar el dominio y establecer límites de contexto, y BDD como mecanismo para describir el comportamiento esperado desde el punto de vista funcional (Hippchen et al., 2017). El artículo describe cómo usar escenarios BDD como guía para generar modelos de dominio en DDD, promoviendo así una conexión más directa entre requerimientos funcionales y decisiones de diseño arquitectónico. El enfoque busca mejorar la trazabilidad y facilitar la alineación entre negocio y tecnología. Sin embargo, la propuesta carece de una metodología operativa definida: no se presentan fases, artefactos, actividades ni ejemplos sistemáticos que permitan su aplicación práctica en entornos reales. Se trata más de una visión teórica y estratégica, sin guías concretas para implementar la integración entre BDD y DDD. Tampoco contempla mecanismos de validación continua ni herramientas de apoyo algorítmico al diseño. En este sentido, BDMA toma como base conceptual el vínculo propuesto por Hippchen pero lo lleva a la práctica mediante una secuencia metodológica concreta, con cinco fases, artefactos asociados, técnicas colaborativas (*Event Storming*, *Example Mapping*), y validación funcional a través de especificación ejecutable. BDMA no solo articula DDD y BDD, sino que también los integra con mecanismos semánticos y con criterios de evolución arquitectónica, lo que permite mantener la coherencia del sistema en escenarios de cambio continuo. Desde la perspectiva de *Design Science Research* (DSR), el enfoque de Hippchen aporta valor conceptual y una justificación teórica sólida para la combinación de enfoques, pero carece de operatividad metodológica y evaluación empírica, lo cual limita su aplicabilidad directa en procesos de diseño iterativo.

5.1.6. A guide for microservices in greenfield projects

El trabajo propone una guía práctica para la adopción de microservicios en proyectos *greenfield* (Cardoso, 2021). La propuesta se basa en la experiencia industrial y en buenas prácticas extraídas de la literatura, organizadas en torno a cinco dimensiones clave:

- Visión del producto y contexto del negocio
- Diseño y granularidad de los servicios
- Prácticas de desarrollo e infraestructura
- Estrategias de despliegue
- Observabilidad y mantenimiento

Esta guía tiene como objetivo ayudar a equipos de desarrollo a tomar decisiones arquitectónicas coherentes en etapas tempranas del proyecto, evitando decisiones erróneas o prematuras. Si bien no propone una metodología detallada para la identificación de microservicios, Cardoso destaca la importancia de postergar las decisiones estructurales hasta haber comprendido adecuadamente el dominio funcional, y sugiere prácticas ágiles como *Event Storming* y uso de narrativas de usuario para guiar el diseño inicial.

En este sentido, la guía comparte varios principios fundamentales con la propuesta BDMA, en particular:

- El enfoque iterativo y exploratorio,
- El descubrimiento progresivo del dominio,
- La importancia de definir el lenguaje común del equipo,
- La necesidad de alinear el diseño arquitectónico con el conocimiento funcional.

Sin embargo, a diferencia de BDMA, la guía de Cardoso no presenta un proceso metodológico detallado ni artefactos operativos para implementar estas ideas, y se limita a ofrecer recomendaciones y patrones de decisión. Tampoco incluye mecanismos de validación funcional ni un modelo formal de trazabilidad.

Por ello, esta guía resulta valiosa como fundamento contextual para la tesis, especialmente dentro del ciclo de relevancia, ya que apoya la premisa de que los proyectos *greenfield* requieren enfoques adaptativos, colaborativos y con fuerte anclaje en el dominio. BDMA puede verse como una materialización metodológica de los principios propuestos por Cardoso, al traducirlos en fases, actividades y artefactos evaluables.

5.1.7. Microservices identification strategies: A review focused on Model-Driven Engineering and Domain Driven Design approaches

En este trabajo llevan a cabo una revisión sistemática de la literatura sobre estrategias y métodos de identificación de microservicios (Schmidt and Thiry, 2020), clasificando 41 publicaciones en función de criterios como la entrada utilizada, el tipo de técnica aplicada (manual, semiautomática, automática) y su orientación (estructural, funcional, organizacional). La revisión permite identificar tres grandes categorías de enfoques:

- Técnicas de minería y análisis estático: centradas en extraer microservicios a partir de sistemas existentes.
- Enfoques estructurados *model-driven*: basados en modelos como casos de uso, procesos BPMN o diagramas de clases.
- Técnicas funcionales o heurísticas: muchas de ellas sin metodología formalizada.

Una de las conclusiones más relevantes del estudio es que pocos trabajos proporcionan una guía práctica y operativa que combine principios funcionales, iterativos y validables, especialmente en entornos *greenfield* y ágiles. También señalan la ausencia de métodos que integren principios de DDD y BDD de manera sistemática.

Este panorama justifica y refuerza el desarrollo de la propuesta BDMA, que responde directamente a esta necesidad detectada por los autores: ofrece un enfoque estructurado, con entradas funcionales (escenarios BDD), técnicas semánticas de agrupamiento, participación colaborativa del negocio y validación iterativa. BDMA se alinea con los principios funcionales y tácticos sugeridos en la literatura, pero los operacionaliza con una metodología replicable, orientada a entornos de descubrimiento progresivo. Desde la óptica de *Design Science Research* (DSR), el trabajo de Schmidt y Thiry proporciona rigurosidad en la identificación del problema y permite justificar la relevancia práctica y teórica del artefacto propuesto en esta tesis.

5.1.8. Towards a UML profile for domain-driven design of microservice architectures

El trabajo propone un perfil UML específico que extiende la notación estándar para modelar conceptos clave de DDD (Rademacher et al., 2018). Su objetivo es formalizar el modelado arquitectónico basado en *bounded contexts*, agregados, entidades y servicios de dominio, de modo que los diseñadores puedan usar herramientas *Model-Driven* (MDD) para derivar arquitecturas de microservicios.

El perfil permite representar explícitamente:

- Los límites de contexto como paquetes UML estereotipados.
- Las relaciones de dependencia entre contextos
- Las unidades de consistencia (*aggregate roots*) y servicios de dominio.

El enfoque representa un avance relevante en la formalización del diseño estratégico de DDD, lo que facilita el uso de herramientas automáticas para generación de artefactos. Además, el uso de un lenguaje visual estandarizado puede ayudar a comunicar decisiones arquitectónicas a otros equipos técnicos.

Sin embargo, desde la perspectiva de la presente tesis, el trabajo tiene algunas limitaciones clave. En primer lugar, parte de una concepción estructuralista del diseño, donde el modelo de dominio se define anticipadamente, sin contemplar procesos iterativos de descubrimiento funcional. En segundo lugar, la propuesta no contempla validación funcional, ni participación del negocio en la identificación de los contextos, y mucho menos un mecanismo de trazabilidad desde los requisitos funcionales hasta los servicios diseñados.

En contraste, BDMA se enfoca en el diseño emergente y semántico, guiado por escenarios funcionales redactados en lenguaje natural (BDD) y validados mediante ejecución. Si bien no utiliza un perfil UML, BDMA prioriza la construcción colaborativa de lenguaje ubicuo, la identificación progresiva de servicios, y su validación continua mediante especificaciones ejecutables.

Desde el punto de vista de DSR, el enfoque del trabajo cumple con los criterios de diseño y contribución teórica (proporciona un artefacto formal útil para la comunidad), pero presenta limitaciones en cuanto a evaluación empírica, iteratividad y validación funcional del diseño.

5.1.9. Evolution of microservices identification in monolith decomposition: A systematic review

El trabajo presenta una revisión sistemática de literatura orientada a analizar las técnicas de extracción de microservicios a partir de sistemas existentes (enfoques *brownfield*) (Oumoussa and Saidi, 2024). Su estudio identifica, categoriza y evalúa más de 60 publicaciones que abordan el problema de migrar arquitecturas monolíticas hacia arquitecturas de microservicios, enfocándose principalmente en criterios como:

- tipo de entrada (código fuente, modelos, artefactos de documentación);
- técnica de extracción (clustering, análisis semántico, transformaciones modelo-modelo);
- nivel de automatización (manual, semiautomático, automático);
- métrica de evaluación (acoplamiento, cohesión, granularidad, rendimiento).

La revisión destaca que la mayoría de los enfoques estudiados se centran en criterios técnicos y estructurales, priorizando aspectos como acoplamiento o división basada en acceso a datos. En general, los autores concluyen que las técnicas actuales presentan limitaciones significativas en cuanto a relevancia semántica, participación del dominio, y soporte para evolución funcional.

Si bien el enfoque de Oumoussa y Saidi se concentra en contextos *brownfield*, sus hallazgos son directamente extrapolables como justificación de la necesidad de enfoques alternativos para entornos *greenfield*, donde no se cuenta con artefactos previos. La carencia de procesos que combinen entrada funcional, diseño progresivo y validación continua refuerza el vacío metodológico que BDMA busca cubrir.

En contraste con los métodos analizados en la revisión, BDMA se sitúa en el ámbito *greenfield*, utilizando narrativas funcionales colaborativas (escenarios BDD) como entrada principal. Además, introduce mecanismos semánticos asistidos para la agrupación conceptual (TF-IDF + HAC), promueve la construcción del lenguaje ubicuo y establece un marco de validación funcional basado en pruebas ejecutables. Por ello, BDMA no solo responde a un dominio diferente (descubrimiento vs. migración), sino que también avanza en los aspectos que Oumoussa y Saidi identifican como debilidades recurrentes: la ausencia de trazabilidad funcional y la escasa participación del negocio en los procesos de segmentación.

Desde la perspectiva de DSR, este trabajo cumple una función clave en el ciclo de relevancia y rigor, al ofrecer un diagnóstico estructurado del estado del arte que fundamenta la necesidad de propuestas metodológicas más integrales como la presentada en esta tesis.

5.1.10. Extraction of microservices from monolithic software architectures

El trabajo propone una técnica de extracción de microservicios desde sistemas monolíticos (*brownfield*) mediante agrupamiento jerárquico automático (HAC) aplicado sobre análisis semántico del código fuente (Mazlami et al., 2017). El enfoque se orienta a apoyar la migración de arquitecturas existentes, identificando componentes

funcionalmente cohesionados a partir de nombres de clases, métodos y estructuras del sistema legado.

La principal innovación técnica del trabajo reside en la aplicación de técnicas de Natural Language Processing para extraer características semánticas de las unidades de código, que luego son agrupadas jerárquicamente usando HAC. Las agrupaciones resultantes se proponen como candidatos a microservicios, priorizando cohesión semántica y separación lógica.

El enfoque representa un avance importante frente a técnicas puramente estructurales (como la segmentación por dependencias o llamadas), ya que permite incorporar información semántica en el proceso de segmentación. Sin embargo, presenta limitaciones en cuanto a:

- la dependencia total del código existente (no es aplicable en proyectos *greenfield*);
- la ausencia de validación funcional o participación de actores del negocio;
- la falta de trazabilidad hacia requerimientos funcionales o expectativas de comportamiento.

En contraste, BDMA toma el concepto de agrupamiento semántico (también usando HAC), pero lo aplica en entornos *greenfield*, utilizando como insumo principal escenarios funcionales BDD. BDMA introduce validación iterativa, lenguaje ubicuo y pruebas ejecutables, que permiten alinear los microservicios con el dominio desde su origen, no como producto derivado del código.

Por tanto, puede afirmarse que BDMA extiende las capacidades técnicas de Mazlami et al., pero integrándolas dentro de un proceso metodológico funcional, participativo y verificable.

5.1.11. Service Cutter: A systematic approach to service decomposition.

El trabajo presenta *Service Cutter* (Gysel et al., 2016), una herramienta para la segmentación automatizada de microservicios a partir de múltiples criterios funcionales y técnicos. El enfoque está diseñado para entornos *brownfield* y propone un modelo de entrada que combina artefactos como:

- entidades del dominio,
- uso compartido de datos,
- restricciones de confidencialidad,
- cambios concurrentes,
- llamadas frecuentes y otros atributos relevantes.

La herramienta aplica un algoritmo de particionamiento para sugerir grupos de entidades que podrían conformar un microservicio, maximizando la cohesión y minimizando el acoplamiento, de acuerdo con una métrica compuesta configurable.

El aporte de *Service Cutter* es significativo en tanto representa uno de los primeros enfoques sistemáticos y multivariable para la descomposición arquitectónica. Su ventaja

principal es la flexibilidad en la selección de criterios según el contexto del sistema, permitiendo adaptar el enfoque a distintas necesidades de partición.

Sin embargo, el enfoque presenta limitaciones importantes para su aplicación en entornos *greenfield*. En primer lugar, requiere un modelo estructurado de entrada, típicamente derivado de artefactos existentes (clases, bases de datos, entidades), lo cual lo hace inadecuado para proyectos sin sistemas previos. En segundo lugar, la herramienta no incorpora participación colaborativa ni validación funcional de los servicios sugeridos, lo cual dificulta alinear el diseño con la lógica del negocio.

Por el contrario, BDMA propone una estrategia orientada al descubrimiento, que parte de escenarios funcionales redactados en lenguaje natural (BDD) y utiliza mecanismos semánticos para sugerir grupos de términos o funcionalidades, pero siempre dentro de un marco participativo y validable. BDMA también emplea HAC para agrupamiento conceptual, pero lo integra en un proceso metodológico con fases, roles, artefactos y pruebas ejecutables. Desde la perspectiva de DSR, Service Cutter representa un artefacto de gran valor para contextos de reingeniería, mientras que BDMA se posiciona como un método de diseño emergente en dominios de exploración funcional.

5.1.12. Behavior-Driven Development of Microservice Applications

En este trabajo abordan el desafío de aplicar BDD al desarrollo de aplicaciones basadas en microservicios, identificando que hasta el momento había poca orientación sistemática sobre cómo combinar estas prácticas (Tuğlular et al., 2021). Los autores proponen un enfoque iterativo y jerárquico de pruebas, estructurado en tres niveles conforme a la pirámide de *testing*: pruebas de extremo a extremo (E2E), pruebas de componente y pruebas unitarias. Para cada nivel, se redactan escenarios BDD usando *Gherkin*, se implementan las funcionalidades necesarias y luego se desarrollan pruebas negativas que validan la robustez del comportamiento esperado. El enfoque fue validado mediante un caso práctico que incluye cinco microservicios desplegados de forma independiente. Se utilizó *Jenkins* para orquestación y *Cucumber* para automatización de pruebas, logrando una pirámide balanceada con una alta cobertura de pruebas unitarias y un número limitado pero significativo de pruebas E2E. Los autores concluyen que su método permite implementar microservicios alineados al comportamiento funcional esperado, favoreciendo calidad, trazabilidad y menor acoplamiento entre servicios.

Este enfoque está estrechamente alineado con los objetivos del marco BDMA, particularmente en cuanto a la integración de BDD como técnica de validación continua y especificación funcional colaborativa. Ambos enfoques comparten la premisa de que el diseño arquitectónico debe emerger del comportamiento esperado por el negocio, expresado en escenarios *Given–When–Then*. Sin embargo, el método de Tuğlular no incorpora elementos de DDD ni patrones estratégicos como *bounded contexts*, lo que lo distancia del enfoque propuesto en BDMA. Mientras BDMA propone una trazabilidad semántica desde el dominio al diseño técnico, el método de Tuğlular se focaliza en asegurar la calidad del comportamiento observable mediante pruebas.

5.2. Síntesis crítica

La revisión exhaustiva de los trabajos significativos vinculados a la identificación y diseño de microservicios ha permitido construir un panorama claro y fundado sobre el estado actual del conocimiento en el área. Este relevamiento ha sido clave para delimitar el espacio de diseño de la propuesta metodológica BDMA, así como para identificar las debilidades comunes que justifican su desarrollo desde una perspectiva de investigación en ingeniería del software basada en DSR.

En primer lugar, se constata que la mayoría de los enfoques se inscriben en contextos *brownfield*, focalizados en la extracción de microservicios desde sistemas existentes (Gysel et al., 2016; Mazlami et al., 2017; Vera-Rivera et al., 2020). Estos trabajos aportan soluciones algorítmicas o semiautomáticas basadas en artefactos preexistentes, como clases, casos de uso o estructuras de datos. Si bien estas estrategias son relevantes para procesos de reingeniería, suponen la existencia de sistemas maduros y no resultan aplicables a entornos *greenfield*, donde el sistema aún no fue diseñado y el conocimiento del dominio debe emerger desde el análisis funcional y la colaboración con expertos.

En segundo lugar, entre los enfoques que abordan el diseño de microservicios en contextos *greenfield* (Bajaj et al., 2022; Cardoso, 2021; Hippchen et al., 2017), se advierte una ausencia de procesos metodológicos operativos. Estas propuestas carecen de fases claramente estructuradas, roles definidos, artefactos intermedios y mecanismos formales de validación. Además, si bien muchos de ellos invocan a DDD o a BDD, lo hacen de forma parcial, conceptual o sin operacionalización concreta. La participación del negocio, el desarrollo del lenguaje ubicuo y la trazabilidad entre requisitos y servicios son tratados superficialmente o se delegan al juicio experto, sin mecanismos metodológicos que los garanticen. Otros trabajos recientes como GTMicro (Bajaj et al., 2024) avanzan hacia la automatización basada en NLP y Transformers, mostrando que el área evoluciona hacia enfoques data-driven. Sin embargo, estos métodos carecen de participación del negocio, trazabilidad semántica y validación continua, características centrales del marco BDMA.

Un tercer conjunto de hallazgos proviene de trabajos de revisión sistemática, que aportan rigor metodológico y evidencia empírica sobre la fragmentación del campo. En particular, (Schmidt and Thiry, 2020) clasifican los enfoques existentes según su entrada, técnica y nivel de automatización, y concluyen que son escasos los métodos que combinan fundamentos del dominio, técnicas de diseño y mecanismos de validación reproducible. Por su parte, (Oumoussa and Saidi, 2024) advierten que la mayoría de las técnicas de identificación ignoran la participación del dominio y carecen de trazabilidad semántica, lo que limita su aplicabilidad en entornos ágiles.

Frente a este panorama, la propuesta BDMA (*Behavior-Driven Microservice Architecture*) surge como una metodología que responde directamente a las brechas detectadas. Su diseño metodológico se apoya en una combinación de principios estratégicos (DDD), tácticas funcionales (BDD) y validación funcional mediante especificaciones ejecutables. BDMA está compuesta por fases iterativas que incluyen prácticas colaborativas (*Event Storming*), artefactos intermedios (mapa de dominio, lenguaje ubicuo, contratos de servicio), y un marco de validación funcional que permite comprobar los límites de contexto en forma progresiva. Desde la lógica del ciclo de diseño en *Design Science Research*, los trabajos analizados permiten:

- Definir el problema práctico y la motivación real del entorno (ciclo de relevancia).
- Establecer el fundamento científico sobre el cual se construye la propuesta (ciclo de rigor).
- Contrastar empírica y teóricamente los atributos diferenciales del artefacto propuesto (ciclo de evaluación).

En conjunto, estos hallazgos otorgan rigor científico, legitimidad académica y necesidad práctica a la construcción de BDMA, no como una evolución incremental de los métodos existentes, sino como una respuesta integrada, operativa y orientada al diseño funcional iterativo de microservicios en proyectos ágiles *greenfield*.

Capítulo 6. Ciclo de Diseño

Este capítulo presenta en detalle el marco *Behavior-Driven Microservice Architecture* (BDMA), describiendo sus fundamentos conceptuales, su estructura general y sus cinco fases operativas. El objetivo es ofrecer una visión clara y completa del método, de manera que pueda aplicarse y evaluarse de forma sistemática en capítulos posteriores.

BDMA surge como respuesta a vacíos metodológicos identificados en la literatura reciente sobre diseño de microservicios en proyectos ágiles *greenfield*. Asimismo, propone un proceso iterativo, colaborativo y trazable que integra prácticas de descubrimiento funcional y modelado estratégico, articulando tres principios centrales:

- Utilización de BDD como mecanismo temprano de descubrimiento de comportamiento, capturando reglas y escenarios del negocio mediante lenguaje natural estructurado (Hellesoy et al., 2017; Smart and Molak, 2023).
- Integración con DDD, de modo que dichos comportamientos se traduzcan en *bounded contexts* y posteriormente en microservicios coherentes y alineados al dominio (Brandolini, 2018; Evans, 2004; Rademacher et al., 2018).
- Aplicación en entornos ágiles y *greenfield*, permitiendo que la arquitectura emerja y evolucione conforme progresa la comprensión del dominio y cambian los requisitos (Ford et al., 2017; Taibi and Lenarduzzi, 2018).

Por otro lado, BDMA se apoya en los principios de la arquitectura evolutiva (Ford et al., 2017a) y concibe la arquitectura como un fenómeno dinámico y verificable. En este sentido, BDD actúa como un puente metodológico entre los requisitos expresados en lenguaje natural y la estructura técnica resultante, permitiendo derivar límites de contexto y decisiones arquitectónicas directamente desde escenarios funcionales (Chelimsky et al., 2010).

El resto del capítulo describe los componentes conceptuales que sustentan el marco y desarrolla cada una de sus cinco fases, especificando objetivos, actividades, entradas, salidas y artefactos que permiten su aplicación rigurosa y reproducible.

6.1. Componentes conceptuales del marco BDMA

Esta sección expone los tres componentes conceptuales (pilares) de BDMA: (1) *Domain-Driven Design*, (2) *Behavior-Driven Development* y (3) Arquitectura evolutiva/prácticas ágiles. A continuación, se propone una versión reducida para introducir al marco:

- ***Domain-Driven Design*** : Proporciona la base estratégica del método. DDD ofrece mecanismos para identificar unidades semánticas coherentes del dominio y construir un lenguaje ubicuo compartido entre negocio y desarrollo. En BDMA, DDD estructura el dominio definiendo límites conceptuales que reflejan responsabilidades de negocio, y facilita la transición a microservicios: cada *bounded context* puede evolucionar en un servicio autónomo y cohesivo. En resumen, los mapas de contexto y patrones de integración de DDD cimentan una arquitectura modular y evolutiva alineada con el dominio.
- ***Behavior-Driven Development***: Actúa como mecanismo funcional para capturar y validar el comportamiento esperado del sistema mediante escenarios en

lenguaje natural. Estos escenarios son especificaciones ejecutables que fomentan la colaboración entre roles técnicos y de negocio. En BDMA, BDD cumple una doble función: (1) sirve de fuente primaria de requisitos y (2) funciona como validación continua, ya que dichos escenarios se automatizan para verificar que la arquitectura implementada satisface los requisitos a medida que el sistema evoluciona. Al capturar reglas de negocio con ejemplos concretos, BDD garantiza trazabilidad funcional desde los escenarios hasta los servicios (cada escenario se refleja en contratos o pruebas de los microservicios correspondientes).

- **Arquitectura evolutiva y prácticas ágiles:** Constituye el tercer pilar, orientado a mantener la adaptabilidad del diseño. BDMA adopta principios de *evolutionary architecture*, concibiendo la arquitectura como algo adaptable bajo validación constante. En la práctica esto implica: (1) Construcción iterativa de la arquitectura; (2) Gobernanza ligera mediante *Architectural Decision Records* (ADR) para documentar decisiones críticas de forma explícita y versionada; (3) Integración con el ciclo ágil, incorporando las fases de BDMA en las actividades recurrentes de refinamiento, planificación e implementación del equipo. Gracias a estas prácticas, BDMA opera como un marco adaptativo acompañando la evolución del producto y del entendimiento del dominio conforme avanzan las iteraciones.

6.2. Diferencial metodológico

BDMA se distingue de los enfoques existentes en el diseño de microservicios por integrar, en un proceso operativo y reproducible, prácticas que tradicionalmente se aplican en forma aislada: el descubrimiento funcional guiado por comportamiento (BDD), el modelado estratégico del dominio (DDD) y los principios de arquitectura evolutiva. Mientras que gran parte de la literatura sobre identificación de microservicios se concentra en técnicas algorítmicas, recomendaciones heurísticas o procesos orientados a la reingeniería de sistemas existentes (Gysel et al., 2016; Mazlami et al., 2017; Oumoussa and Saidi, 2024), BDMA propone un método específicamente diseñado para proyectos *greenfield*, donde el dominio aún no está estabilizado y las decisiones arquitectónicas requieren validación continua. Asimismo, constituye así un marco metodológico innovador, orientado a la práctica y basado en evidencia funcional, que fortalece la autonomía y consistencia de los equipos ágiles en el diseño de soluciones escalables, trazables y alineadas con el dominio de negocio. Asimismo, es un método iterativo que transforma requisitos funcionales en decisiones arquitectónicas verificables.

6.3. El marco Behavior-Driven Microservices Architecture

Este apartado resume las cinco fases operativas del marco BDMA (Battaglia et al., 2025). Estas fases permiten transformar iterativamente escenarios funcionales en artefactos arquitectónicos verificables, fomentando una evolución controlada de los microservicios en proyectos *greenfield*. A continuación, se sintetiza cada fase con su objetivo, práctica clave y respaldo bibliográfico con un la tabla 5, que es acompañada con el método representado gráficamente en la ilustración 5.

Fase	Concepto	Fuente
1. Clasificación funcional de escenarios	Arquitectura evolutiva alineada a <i>features/epics</i>	(Ford et al., 2017)
	<i>Continuous Architecture</i> : decisiones orientadas a negocio	(Erder and Pureur, 2015; Woods et al., 2021)
2. Extracción colaborativa de elementos del dominio	BDD para describir comportamiento y dominio (Smart and Molak, 2023)	(Smart and Molak, 2023)
	Relación entre BDD y DDD	(Hippchen et al., 2017)
	Uso de BDD para extracción en contexto DDD	(Rademacher et al., 2018)
3. Delimitación de bounded contexts	Context Map, bounded contexts en DDD	(Evans, 2004)
	Event Storming como técnica colaborativa	(Brandolini, 2018)
	Aplicación de patrones estratégicos en arquitecturas distribuidas	(Vernon, 2013, 2016)
	Modelado formal de contextos y relaciones en microservicios	(Rademacher et al., 2018)
4. Derivación de interfaces y contratos API	Diseño de APIs contract-first y patrones de diseño para testabilidad	(Zimmermann et al., 2019, 2022)
	Especificación como prueba en arquitecturas evolutivas	(Ford et al., 2017)
	Mapeo de escenarios BDD	(Smart and Molak, 2023)
5. Validación continua y evolución estructural	ADR como técnica formal para registrar decisiones arquitectónicas	(Kopp et al., 2018)
	Validación continua de hipótesis arquitectónicas	(Ford et al., 2017)

	Evolución y versionado del mapa de contexto en <i>Continuous Architecture</i>	(Erder and Pureur, 2015; Woods et al., 2021)
--	--	---

Tabla 5. Resumen del marco BDMA

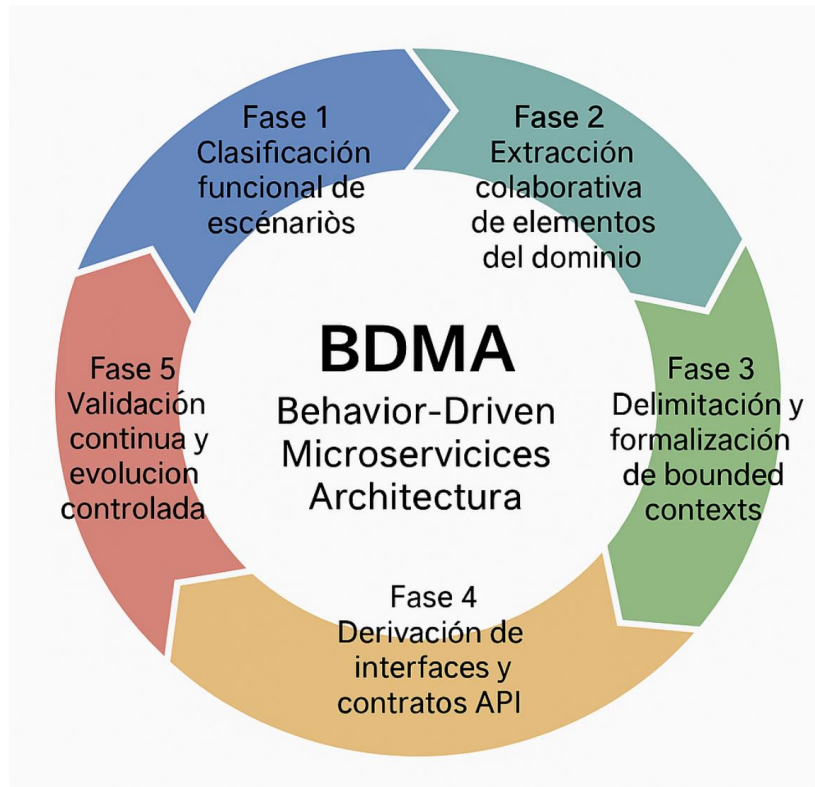


Ilustración 6. Marco BDMA

A continuación, describimos las 5 fases de BDMA de manera resumida para profundizar luego en secciones subsiguientes:

- Fase 1: Clasificación funcional de escenarios:** esta fase tiene como objetivo organizar el backlog funcional del sistema —expresado en escenarios BDD redactados en lenguaje Gherkin— en agrupaciones semánticamente coherentes que representen unidades funcionales relevantes como *features*, épicas o procesos de negocio. Esta clasificación permite descubrir límites funcionales naturales y anticipar áreas de responsabilidad que pueden derivar en futuros *bounded contexts*. Inspirada en los principios de arquitectura evolutiva (Ford et al., 2017) y *continuous architecture* (Erder and Pureur, 2015; Woods et al., 2021), la clasificación funcional se realiza como parte del refinamiento de backlog y constituye la base sobre la cual se estructuran las decisiones de diseño posteriores. Su aplicación temprana garantiza que la arquitectura se alinee con las necesidades del negocio.
- Fase 2: Extracción colaborativa de elementos del dominio:** una vez agrupados los escenarios, esta fase se enfoca en identificar, mediante prácticas colaborativas, los elementos fundamentales del dominio expresados en cada historia. El equipo técnico, junto con los expertos del negocio, lleva adelante

sesiones estructuradas de *Event Storming* (Brandolini, 2018) y *Example Mapping* (Wynne, 2015) para revelar actores, comandos, eventos de dominio y entidades involucradas. Estas sesiones permiten descomponer el comportamiento deseado en piezas semánticas reutilizables, estableciendo una base conceptual sólida para el modelado estratégico. El lenguaje ubicuo emerge como producto de esta colaboración, y los comandos y eventos extraídos alimentan directamente el diseño de contextos e interfaces. Además de fortalecer el entendimiento compartido del sistema, esta fase habilita la trazabilidad precisa entre escenarios funcionales y elementos arquitectónicos, dando lugar a un mapa intermedio del dominio centrado en el comportamiento.

- **Fase 3: Delimitación y formalización de *bounded contexts*:** con los elementos del dominio ya identificados, se procede a su agrupamiento en *bounded contexts* estratégicos. Estos contextos se definen como unidades semánticamente cohesivas que agrupan responsabilidades, reglas de negocio, entidades y eventos afines, promoviendo la alta cohesión interna y el bajo acoplamiento externo. Se aplican aquí patrones de diseño propuestos por DDD, como *context mapping* y relaciones de dependencia explícitas entre contextos. El resultado es un mapa de contextos actualizado que refleja tanto la estructura organizacional como los flujos de comunicación entre componentes. Esta delimitación formal constituye el corazón del diseño estratégico de microservicios, ya que define fronteras claras de autonomía, sincronización y evolución. La claridad obtenida en esta etapa permite construir una arquitectura sostenible en el tiempo, capaz de adaptarse sin perder coherencia.
- **Fase 4: Derivación de contratos e interfaces:** a partir de los contextos definidos en la etapa anterior, esta fase traduce los comandos y eventos del dominio en contratos técnicos concretos: API, *endpoints* o mensajes que expresan cómo cada servicio se comunica con otros o con el exterior. Se adopta un enfoque API-first, donde los contratos se diseñan antes de la implementación, y se asegura que cada interfaz esté motivada y validada por al menos un escenario BDD. Esta conexión funcional garantiza que el diseño técnico sea verificable desde el punto de vista del negocio. Las interfaces se modelan siguiendo buenas prácticas para microservicios (Jamshidi et al., 2018; Newman, 2021; Richardson, 2018) y se documentan usando estándares como OpenAPI o esquemas de eventos. A su vez, se mantiene la trazabilidad entre cada escenario, su operación técnica correspondiente y las pruebas automatizadas que lo validan. Esto permite que la arquitectura sea testeable de forma continua.
- **Fase 5: Validación continua y evolución arquitectónica:** la última fase sostiene el principio de que la arquitectura debe evolucionar al ritmo del dominio. A medida que se incorporan nuevos escenarios o cambian los existentes, el sistema es verificado mediante pruebas BDD automatizadas que actúan como especificaciones ejecutables. Cualquier desviación detectada impulsa la revisión del diseño, que puede implicar la modificación de contextos, la actualización de interfaces o la refactorización de componentes. Cada decisión arquitectónica significativa se documenta mediante registros ADR, asegurando transparencia y auditabilidad en la evolución del sistema (Ford et al., 2017; Fowler, 2015c). Además, se versionan el mapa de contextos, los contratos y los artefactos clave del marco, manteniendo una trazabilidad integral. Esta fase convierte a la

arquitectura en una práctica viva, en constante diálogo con el negocio, que aprende de la implementación, ajusta su estructura y se mantiene siempre alineada con los objetivos funcionales del sistema (Woods et al., 2021).

6.3.1. Vinculación de BDMA con el ciclo ágil

Cada fase del marco BDMA está alineada con el ciclo de vida de desarrollo ágil. A diferencia de un enfoque secuencial tradicional, BDMA no ejecuta sus fases una única vez ni de forma monolítica, sino que las integra de manera incremental en cada iteración (*sprint*) del proyecto. De este modo, en cada ciclo de desarrollo las actividades de BDMA se enfocan en los cambios o nuevas funcionalidades priorizadas, asegurando que la arquitectura evoluciona en sincronía con el producto. Cada fase está fundamentada en prácticas reconocidas tanto de DDD como de BDD, y busca traducir el conocimiento del dominio en decisiones arquitectónicas progresivas y trazables (Brandolini, 2018; Evans, 2004; Hellesoy et al., 2017). En términos generales, el proceso BDMA sigue las siguientes etapas en correspondencia con un *sprint* ágil típico:

- **Refinamiento de *backlog* (Antes del inicio del *sprint*):** Se aplica la **Fase 1: Clasificación funcional de escenarios**, donde los nuevos escenarios BDD identificados o aprobados para la próxima iteración se organizan y agrupan de forma coherente (por funcionalidad, *feature* o épica). Esta clasificación inicial prepara el terreno para entender qué partes del dominio serán abordadas en el *sprint*.
- **Planificación/Inicio del *sprint*:** Se activa la **Fase 2: Extracción colaborativa de elementos del dominio** sobre los escenarios priorizados. Durante esta etapa temprana del *sprint*, el equipo (incluyendo desarrolladores y expertos de negocio) analiza cada escenario BDD para identificar actores, acciones (comandos) y reacciones (eventos) relevantes, extrayendo elementos del dominio que serán la base del diseño técnico. Este análisis en conjunto asegura un entendimiento compartido y alimenta las siguientes fases con información semántica rica.
- **Diseño técnico inicial:** Conforme el equipo empieza a diseñar la solución del *sprint*, se emplea la **Fase 3: Delimitación y formalización de *bounded contexts***. Aquí se revisan y ajustan los **límites de contexto** existentes o se definen nuevos, en caso de que las funcionalidades introducidas por los escenarios requieran repensar la partición arquitectónica. No todos los *sprints* implicarán cambios drásticos en la arquitectura global; sin embargo, BDMA establece que durante esta fase se verifique si los nuevos elementos del dominio encajan en los contextos actuales o ameritan la creación/refactorización de algún contexto. Como resultado de Fase 3, se obtiene un *context map* actualizado que refleja las responsabilidades y relaciones (p. ej., dependencias o integraciones) entre los contextos delimitados.
- **Desarrollo/Implementación:** A medida que avanza la implementación de las historias de usuario del *sprint*, se desencadena la **Fase 4: Derivación de interfaces y contratos API cuando corresponde**. Es decir, si los escenarios BDD del *sprint* involucran la exposición de nuevas funcionalidades o la interacción entre contextos, se deriva la definición de contratos de servicio (interfaces API) para los microservicios involucrados. En esta fase se traducen los comandos y

eventos identificados en especificaciones de *endpoints*, mensajes o APIs REST/GraphQL, asegurando que cada contrato esté respaldado por un escenario BDD que prueba su comportamiento. Esta actividad garantiza que las **interfaces públicas** de los microservicios nazcan diseñadas para ser testeables y alineadas con necesidades del negocio.

- **Ejecución continua durante el sprint:** Paralelamente al desarrollo, se mantiene activa la **Fase 5: Validación continua y evolución controlada**. Esta fase consiste en la ejecución regular de los escenarios BDD como pruebas automatizadas, la detección temprana de impactos o regresiones y el registro de decisiones arquitectónicas en forma de ADRs (*Architectural Decision Records*). En la práctica, cada vez que un escenario BDD falla o evidencia un comportamiento no cubierto, el equipo reevalúa si hace falta ajustar el diseño (por ejemplo, dividir un servicio, agregar una nueva operación a un API, etc.) y deja constancia de dicha decisión. La Fase 5 opera de forma transversal a todo el sprint como un mecanismo de *gobernanza ligera* que mantiene la coherencia evolutiva del sistema.
- Al finalizar el sprint, BDMA sugiere realizar una consolidación de artefactos: verificar que el *context map* global esté actualizado con los cambios realizados, versionar los contratos de servicio que hubieran cambiado y cerrar los ADRs correspondientes a la iteración. Esta actividad de cierre asegura que la documentación arquitectónica y los artefactos intermedios reflejen el estado *real* de la arquitectura antes de comenzar el siguiente ciclo.

Cabe destacar que BDMA parte de la premisa de que el proyecto dispone de un *backlog* funcional ya compuesto por escenarios BDD validados (obtenidos a través de prácticas de descubrimiento de requisitos con el negocio). El marco no incluye la generación de esos escenarios en sus fases, sino que asume su existencia como insumo inicial. Esto permite enfocar las actividades en cómo aprovechar dichos escenarios para el diseño arquitectónico, sin prescribir una técnica específica de recopilación de requisitos (pueden obtenerse mediante *Example Mapping*, entrevistas, talleres BDD, etc., fuera del alcance de BDMA). Resumiendo, la estructura general de BDMA consta de cinco fases operativas que se repiten iterativamente en cada *sprint* según las necesidades impuestas por los cambios funcionales. Cada fase produce uno o más artefactos intermedios (mapa de agrupamiento funcional, lista de comandos/eventos, mapa de contextos, especificaciones de API, registros ADR) que sirven para mantener la trazabilidad y facilitar la validación continua entre los requisitos y la arquitectura. En la siguiente tabla 6 se describen detalladamente estas fases, incluyendo sus objetivos, entradas, salidas y el rol que desempeñan en el marco propuesto.

Etapas del ciclo ágil	Fase BDMA relacionada	Qué ocurre
Refinamiento del backlog	Fase 1 (clasificación funcional)	Se agrupan los nuevos escenarios BDD por <i>feature/epic</i>

Inicio del Sprint	Fase 2 (extracción de dominio)	Se analizan los escenarios priorizados, se identifican comandos, eventos y actores
Diseño técnico inicial	Fase 3 (bounded contexts)	Se ajustan o redefinen límites de contextos si hay cambios relevantes
Implementación	Fase 4 (contratos API)	Solo si el contexto impactado expone funcionalidades. Se derivan contratos desde los nuevos escenarios
Durante el Sprint	Fase 5 (validación continua)	Se ejecutan pruebas BDD, se detectan impactos, se registran ADRs y se actualizan artefactos arquitectónicos
Cierre / Revisión de Sprint	Consolidación de artefactos	Se valida que la arquitectura sigue representando al dominio funcional. Se versiona el Context Map

Tabla 6. Relación de BDMA con el ciclo ágil

6.4. El Marco BDMA en detalle

Para comprender mejor el proceso, se acompañará cada fase del marco BDMA con un ejemplo, partiendo de los siguientes escenarios BDD previamente identificados

Feature: Order management

Scenario: Creating a new order

Given a registered customer selects one or more products

And provides valid payment information

When the customer confirms the purchase

Then the system creates a new order with status "Created"

And decreases the stock of the selected products

And sends a confirmation message to the customer

Feature: Order management

Scenario: Canceling an existing order

Given a registered customer has a confirmed order

When the customer requests to cancel the order

Then the system changes its status to "Canceled"

And restores the stock for the canceled products

And sends a cancellation message to the customer

De esta forma, cada fase tendrá detallados los artefactos que genera y acompañados por un ejemplo de aplicación. A partir de aquí, cada fase se presenta con artefactos reproducibles que permiten la replicabilidad completa del método.

6.4.1. Fase 1: Clasificación funcional de escenarios

6.4.1.1. Objetivo

Organizar el backlog de escenarios BDD validados en agrupaciones coherentes que representan *epics* o procesos de negocio. Las entradas son los propios escenarios, redactados en lenguaje Gherkin (lenguaje específico de dominio utilizado para escribir escenarios de comportamiento legibles por humanos en BDD) y complementados con metainformación contextual. La salida principal es un mapa de agrupamiento funcional que actúa como artefacto de referencia para las fases posteriores. Esta actividad se fundamenta en los principios de *evolutionary architecture* (Ford et al., 2017) y *continuous architecture* (Woods et al., 2021), donde las decisiones iniciales se alinean a objetivos de negocio y guían la evolución arquitectónica.

6.4.1.2. Entradas

- Backlog funcional expresado en escenarios BDD redactados con lenguaje Gherkin (*Given-When-Then*)
- Metainformación opcional sobre los escenarios (*tags, epics, user stories* asociadas)

6.4.1.3. Salidas

- Grupos funcionales de escenarios candidatos a *bounded contexts* iniciales.
- Tabla de trazabilidad con la siguiente estructura
 - Epic / proceso
 - Escenario BDD asociado
 - Notas

6.4.1.4. Artefactos generados

- Mapa de agrupamiento funcional con la siguiente estructura (tabla 7)
 - Nombre del grupo
 - Escenarios
 - Actor principal
 - Eventos clave
 - Notas

Grupo funcional (candidato a Epic)	Escenarios incluidos	Actor principal	Eventos clave	Notas
Order lifecycle management	- Creating a new order - Canceling an existing order	Registered customer	- OrderCreated - OrderCanceled - StockDecreased - StockRestored - ConfirmationSent - CancellationSent	Proceso end-to-end del ciclo de vida de una orden; probablemente se divida luego en varios bounded contexts (Ordering, Inventory, Payment, Notification).

Tabla 7. Ejemplo artefacto fase 1 BDMA – mapa de agrupamiento funcional

6.4.2. Fase 2: Extracción colaborativa de elementos del dominio

6.4.2.1. Objetivo

Analizar escenarios agrupados para identificar actores, comandos, eventos y entidades clave. El conocimiento de dominio provisto por expertos complementa la interpretación de los escenarios, permitiendo construir un mapa de elementos del dominio por escenario. Este artefacto aporta trazabilidad explícita entre el comportamiento especificado y los elementos técnicos que lo implementarán. La fase se apoya en el uso de BDD como mecanismo de descripción de comportamiento (Smart and Molak, 2023) y en su integración con DDD para extraer modelos de dominio significativos (Hippchen et al., 2017; Rademacher et al., 2018).

6.4.2.2. Entradas

- Escenarios BDD organizados por agrupamientos funcionales (salida formal de la Fase 1)
- Mapa funcional preliminar con tags de features, epics o procesos
- Conocimiento contextual del negocio (proporcionado por PO o expertos)

6.4.2.3. Salidas

- Lista de comandos y eventos por contexto funcional.
- Mapa preliminar de relaciones entre entidades.
- Tabla de trazabilidad: escenario -> comando , evento o actor

6.4.2.4. Artefactos generados:

- Mapa de elementos del dominio por escenario (tabla 8)
 - Grupo funcional
 - Escenario
 - Actor
 - Comando (*When*)
 - Evento (*Then*)
 - Entidades (*Given*)

Grupo funcional	Escenario	Actor (Who)	Comando (When)	Eventos (Then)	Entidades (Given)
Order lifecycle management	Creating a new order	Registered customer	Customer confirms the purchase	- OrderCreated(status="Created") - StockDecreased(for selected products) - OrderConfirmationSent(to customer)	Customer registered Product selected PaymentInformation (valid) Order (nuevo, aún no persistido al inicio)

					StockItem (para cada producto)
Order lifecycle management	Canceling an existing order	Registered customer	Customer requests to cancel the order	OrderCanceled(status="Canceled") - StockRestored(for canceled products) - OrderCancellationSent(to customer)	- Customer (registered) - Order (confirmed) - Product (productos asociados a la orden) - StockItem (para cada producto)

Tabla 8. Ejemplo artefacto fase 2 BDMA – mapa de elementos del dominio

6.4.3. Fase 3: Delimitación y formalización de *bounded contexts*

Objetivo:

Agrupar comandos, eventos y entidades relacionados (extraídos de los escenarios BDD) en *bounded contexts* coherentes, explícitamente delimitados, que permitan separar responsabilidades, minimizar acoplamientos y sostener la evolución independiente de los servicios. El resultado es un *context map* que explicita las responsabilidades de cada *bounded context* y las relaciones entre ellos, utilizando patrones de interacción (Evans, 2004; Vernon, 2013, 2016). La técnica *Event Storming* (Brandolini, 2018) se integra aquí como herramienta colaborativa para descubrir dependencias y eventos relevantes.

6.4.3.1. Entradas

- Mapa de agrupamientos funcionales (artefacto de la Fase 1)
- Mapa de elementos del dominio por escenario (artefacto de la Fase 2)
- Conocimiento contextual de dependencias organizacionales y técnicas

6.4.3.2. Salidas

- Lista de bounded contexts con nombre, responsabilidad y elementos del dominio asignados
- Mapa de relaciones entre contextos
- Declaración de reglas de interacción (eventos compartidos, ACL, contratos esperados). La comunicación debe ser explícita, controlada (sin acoplamientos indeseados) y mantenible, por ejemplo:
 - Partnership
 - Relación activa de colaboracion
 - Ambos contextos evolucionan juntos
 - Requiere coordinación de equipos
 - Conformist
 - Un contexto acepta otro sin conformarlo
 - Ejemplo: reportes se adapta al modelo de ventas
 - Anti corruption layer (ACL)

- El contexto consumidor traduce el modelo del otro a su propia necesidad
- Evita acoplarse directamente a cambios ajenos

6.4.3.3. Artefactos generados

- Mapa formal de *bounded contexts* (tabla 9)
 - Nombre del context
 - Comandos
 - Eventos
 - Entidades
 - Requiere derivación de contrato api (fase 4)
 - Notas

Bounded Context	Responsabilidad	Comandos principales	Eventos publicados	Entidades clave	¿Requiere contrato API (F4)?
BC01. Ordering	Gestionar el ciclo de vida de órdenes	- CreateOrder - CancelOrder	- OrderCreated - OrderCanceled	- Order - OrderLine	Sí
BC02. Inventory	Gestionar stock de productos	- ReserveStock (opcional) - DecreaseStock - RestoreStock	- StockDecreased - StockRestored	- StockItem - Product (referencia)	Sí
BC03. Payment	Validar y registrar pagos	- AuthorizePayment - CapturePayment	- PaymentAuthorized - PaymentFailed	- Payment - PaymentMethod	Sí
BC04. Notification	Enviar mensajes al cliente	- SendOrderConfirmation - SendOrderCancellation	- OrderConfirmationSent - OrderCancellationSent	- NotificationTemplate - DeliveryChannel	Opcional (puede ser puramente event-driven)

Tabla 9. Ejemplo artefacto fase 3 BDMA – mapa de bounded contexts

Acompañamos el artefacto descrito anteriormente con el mapa de bounded context resultante

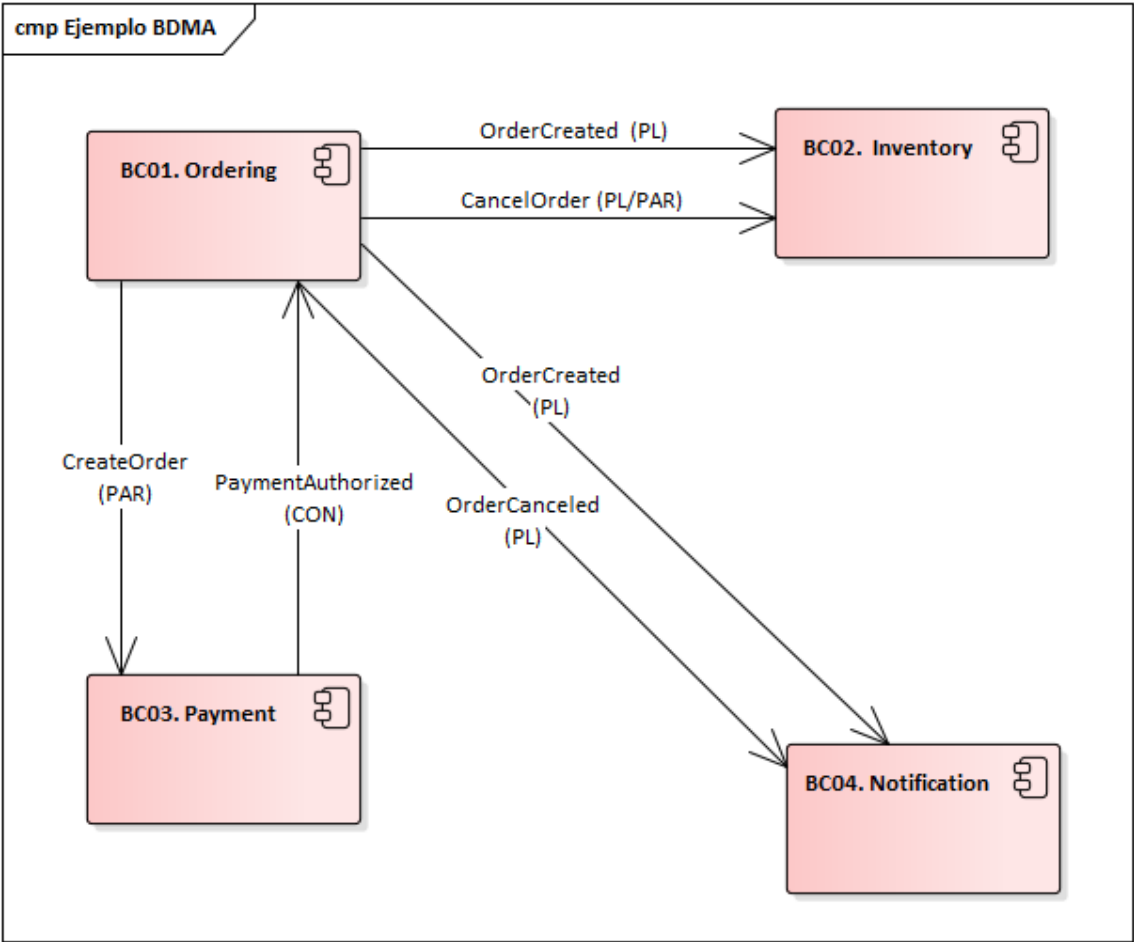


Ilustración 7. Mapa de bounded context con relaciones y patrones

- Mapa de interacciones entre contextos (tabla 10)
 - Contexto Origen
 - Evento
 - Contexto Destino
 - Tipo de relación
 - Medio de interacción
 - Notas

Contexto Origen	Evento / Comando	Contexto Destino	Tipo de relación (DDD)	Medio de interacción	Notas
Ordering	Command: CreateOrder	Payment	Partnership (colaboración estrecha para validar pago)	Síncrono (REST/gRPC)	Ordering llama a Payment para autorizar el pago antes de confirmar la orden.

Payment	Event: PaymentAuthorized	Ordering	Conformist (Ordering adopta el resultado de pago)	Async event (Message Bus) o respuesta síncrona	Orden solo pasa a “Created” si el pago fue autorizado.
Ordering	Event: OrderCreated	Inventory	Published Language	Async event (Message Bus)	Inventory escucha OrderCreated y aplica StockDecreased.
Ordering	Event: OrderCreated	Notification	Published Language	Async event	Notification envía confirmación al cliente.
Ordering	Command: CancelOrder	Inventory	Published Language / Partnership	Async event OrderCanceled o llamada directa	Inventory restaura stock asociado a la orden cancelada.
Ordering	Event: OrderCanceled	Notification	Published Language	Async event	Notification envía mensaje de cancelación al cliente.

Tabla 10. Ejemplo artefacto fase 3 BDMA – mapa de interacción entre contextos

- Identificación de contextos que requieren contratos API
 - Evaluar si cada contexto será consumido por otros o si publica eventos para indicar si requiere fase 4

6.4.4. Fase 4: Derivación de interfaces y contratos API

6.4.4.1. Objetivo

Traducir los comandos y eventos definidos en escenarios BDD, dentro de cada *bounded context*, en contratos de servicio (APIs) que sean verificables, trazables y diseñados para pruebas automatizadas. Esta fase garantiza que la arquitectura técnica respete las especificaciones funcionales y permita evolución segura. Esta fase se fundamenta en patrones de diseño de APIs (Zimmermann et al., 2019) y en la idea de la especificación como prueba en arquitecturas evolutivas (Ford et al., 2017). El artefacto generado asegura la continuidad entre las necesidades funcionales y las interfaces técnicas expuestas.

6.4.4.2. Entradas

- Mapa de elementos del dominio (Fase 2)
- Mapa de bounded contexts (Fase 3)
- Reglas de interacción entre contextos (artefacto 2)
- Escenarios BDD originales como fuente de trazabilidad

6.4.4.3. Salidas

- Especificación de contratos API por contexto.
- Trazabilidad documentada escenario > *endpoint* > prueba

6.4.4.4. Artefactos generados

- Especificación de contratos API con trazabilidad funcional según la siguiente estructura (tabla 11)
 - Contexto
 - Endpoint
 - Verbo
 - Comento o eventos asociados
 - Escenario BDD
 - Observaciones

Contexto	Endpoint/ Topic	Verbo Acción	Comando o eventos asociados	Escenario BDD	Observaciones
Ordering	/orders	POST	CreateOrder	<i>Creating a new order</i>	Crea una nueva orden en estado "Created" luego de pago autorizado. Publica OrderCreated al completarse.
Ordering	/orders/{orderId}/cancel	POST	CancelOrder	<i>Canceling an existing order</i>	Solicita la cancelación de una orden existente. Publica OrderCanceled si la cancelación es aceptada.
Ordering	order-events (topic)	EVENT (publish)	OrderCreated	<i>Creating a new order</i>	Evento de dominio que dispara la disminución de stock y el envío de la confirmación. Base para Inventory y Notification.
Ordering	order-events (topic)	EVENT (publish)	OrderCancelled	<i>Canceling an existing order</i>	Evento de dominio que dispara la restauración de stock y el envío del mensaje de cancelación.
Inventory	order-events (topic)	EVENT (consume)	OrderCreated → StockDecreased	<i>Creating a new order</i>	Inventory se suscribe a OrderCreated y ejecuta el comando interno DecreaseStock. La API

					puede ser interna al contexto.
Inventory	order-events (topic)	EVENT (consume)	OrderCancel ed → StockRestore d	<i>Canceling an existing order</i>	Inventory se suscribe a OrderCanceled y ejecuta RestoreStock para los productos asociados a la orden.
Inventory	/inventory/decrease (opcional)	POST	DecreaseStock	<i>Creating a new order</i>	Puede exponerse como endpoint sincrónico si Ordering decide no usar eventos para stock. Alternativa a event-driven.
Inventory	/inventory/restore (opcional)	POST	RestoreStock	<i>Canceling an existing order</i>	Igual que la fila anterior, pero para restaurar stock en caso de cancelación.
Payment	/payments/auth orize	POST	AuthorizePay ment	<i>Creating a new order</i>	Valida la payment information. Ordering puede requerir status = Authorized antes de crear la orden.
Payment	payment-events (topic, opcional)	EVENT (publish)	PaymentAuth orized, PaymentFail ed	<i>Creating a new order (implicito)</i>	Si se modela asincrónicamente, Ordering podría reaccionar a estos eventos antes de publicar OrderCreated.
Notificati on	order-events (topic)	EVENT (consume)	OrderCreate d → SendOrderC onfirmation	<i>Creating a new order</i>	Notification se suscribe a OrderCreated y genera el mensaje de confirmación al cliente.
Notificati on	order-events (topic)	EVENT (consume)	OrderCancel ed → SendOrderC ancellation	<i>Canceling an existing order</i>	Notification se suscribe a OrderCanceled y envía el mensaje de cancelación.
Notificati on	/notifications/or der-confirmation	POST	SendOrderC onfirmation	<i>Creating a new order</i>	Endpoint opcional para disparar notificaciones desde otros contextos o

					herramientas administrativas.
Notificación	/notifications/order-cancellation	POST	SendOrderCancellation	<i>Cancelling an existing order</i>	Equivalente para la notificación de cancelación.

Tabla 11. Ejemplo artefacto fase 4 BDMA – especificación de contratos API

6.4.5. Fase 5: Validación continua y evolución controlada

6.4.5.1. Objetivo

Garantizar que la arquitectura de microservicios evolucione en sincronía con los cambios funcionales del sistema, manteniendo la trazabilidad completa desde los escenarios BDD, y documentando cada decisión estructural mediante mecanismos formales que habiliten la revisión, refactorización y mantenimiento. Como salidas se producen bitácoras de decisiones arquitectónicas (ADR), informes de trazabilidad y versiones actualizadas del *context map*. Este registro asegura la auditabilidad del proceso y habilita la evolución incremental mediante el versionado de contratos y contextos. El fundamento se encuentra en los ADR como técnica formal de documentación (Kopp et al., 2018) y en el rol de la validación continua de hipótesis arquitectónicas (Erder and Pureur, 2015; Ford et al., 2017). De este modo, cada fase de BDMA no solo contribuye a la construcción de una arquitectura coherente y evolutiva, sino que además genera artefactos intermedios que fortalecen la trazabilidad entre requisitos funcionales y decisiones técnicas, superando la brecha metodológica identificada en la literatura.

6.4.5.2. Entradas

- Contratos API y relaciones de integración (Fase 4)
- Mapa de bounded contexts versionado (Fase 3)
- Escenarios BDD actualizados
- Historial de decisiones previas (bitácora ADR)

6.4.5.3. Actividades

6.4.5.4. Artefactos generados

- Bitácora de decisiones arquitectónicas (ADR Log)
 - Repositorio de decisiones versionado, accesible, con formato estándar
- Informe de trazabilidad evolutiva
 - Documento o herramienta que mapea:
 - Escenario BDD > Contrato API > Servicio > ADR
- Modelo de dominio y mapa de contexto versionado
 - Versión actual y coherente con el mapa de contexto
- Registro de contratos API ajustados o deprecados (tabla 11)
 - Contratos API (Fase 4), ADRs asociados a cambios

Ejemplo:

ADR-001 – Inventory como microservicio independiente

- **Contexto**
La creación y cancelación de órdenes impacta el stock de productos. El stock también será modificado por otros procesos (devoluciones, compras, ajustes manuales).
- **Decisión**
Se define un bounded context Inventory como un microservicio independiente (InventoryService).
 - OrderingService publicará eventos OrderCreated y OrderCanceled.
 - InventoryService consumirá estos eventos para ejecutar StockDecreased y StockRestored.
- **Consecuencias positivas**
- Permite escalar Inventory de forma independiente.
- Separa responsabilidades (lifecycle de órdenes vs gestión de stock).
- Facilita integrar futuros flujos logísticos.
- **Consecuencias negativas**
 - Introduce complejidad de consistencia eventual.
 - Requiere infraestructura de mensajería y observabilidad.

ADR-002 – Estilo de integración Ordering ↔ Payment

- **Contexto**
El escenario Creating a new order asume que el pago es válido antes de que la orden pase a estado "Created". Se necesita definir cómo Ordering y Payment colaboran.
- **Decisión**
La integración inicial será **sincrónica**, vía HTTP:
 - OrderingService invoca POST /payments/authorize en **PaymentService**.
 - Solo si el resultado es status = Authorized, se crea la orden y se publica OrderCreated.
- **Consecuencias positivas**
 - Flujo más simple de implementar y razonar.
 - La orden nunca queda en “limbo” sin tener claro el estado del pago.
- **Consecuencias negativas**
 - Ordering queda acoplado a la disponibilidad de Payment.
 - Aumenta la latencia de creación de órdenes.
 - Podrían aparecer cuellos de botella bajo alta carga.

ADR-003 – Notificaciones basadas en eventos de dominio

- **Contexto**
Los escenarios Creating a new order y Canceling an existing order requieren enviar mensajes al cliente (confirmación y cancelación).

- **Decisión**

Se define un microservicio **NotificationService** que se suscribe al topic order-events y reacciona a:

- OrderCreated → enviar mensaje de confirmación.
- OrderCanceled → enviar mensaje de cancelación.

- **Consecuencias positivas**

- Desacopla el envío de notificaciones del flujo transaccional de órdenes.
- Permite añadir nuevos canales (SMS, push, etc.) sin modificar Ordering.

- **Consecuencias negativas**

- Las notificaciones no son estrictamente parte de la transacción de negocio.
- Requiere monitoreo para evitar pérdida/duplicación de mensajes.

ADR-004 – Deprecación de endpoints síncronos de Inventory

Esta ya es una **decisión evolutiva**, a partir de la primera versión.

- **Contexto**

Inicialmente se definieron endpoints POST /inventory/decrease y POST /inventory/restore como alternativas síncronas. En la práctica, el equipo decidió estandarizar el uso de eventos para stock.

- **Decisión**

Los endpoints POST /inventory/decrease y POST /inventory/restore se marcan como **deprecated** a partir de la versión v1.1 del contrato.

- El mecanismo oficial de integración será solo event-driven vía OrderCreated y OrderCanceled.

- **Consecuencias positivas**

- Simplifica el modelo de integración.
- Reduce caminos alternativos que podían generar inconsistencias.

- **Consecuencias negativas**

- Sistemas externos ya integrados deberán migrar a la suscripción de eventos.
- Puede requerir un período de coexistencia con soporte a ambos enfoques.

Escenario o BDD	Contrato API / Topic	Servicio	ADRs relacionadas	Observaciones
Creating a new order	POST /orders	OrderingService	ADR-001, ADR-002, ADR-003	Creación de orden. Flujo depende de la autorización de pago (ADR-002) y dispara eventos para

				Inventory y Notification (ADR-001, ADR-003).
Creating a new order	EVENT OrderCreated en order-events	OrderingService	ADR-001, ADR-003	Evento de dominio publicado tras crear la orden. Base para ajuste de stock y envío de notificación.
Creating a new order	EVENT OrderCreated (consume)	InventoryService	ADR-001, ADR-004	Dispara StockDecreased. Inicialmente podía existir también POST /inventory/decrease (ADR-004 lo deprecia).
Creating a new order	EVENT OrderCreated (consume)	NotificationService	ADR-003	Dispara SendOrderConfirmation.
Creating a new order	POST /payments/authorize	PaymentService	ADR-002	Integración sincrónica para validar pago antes de crear la orden.
Canceling an existing order	POST /orders/{orderId}/cancel	OrderingService	ADR-001, ADR-003	Cancelación de la orden. Emite OrderCanceled que impacta stock y notificaciones.
Canceling an existing order	EVENT OrderCanceled en order-events	OrderingService	ADR-001, ADR-003	Evento de dominio para reflejar la cancelación.
Canceling an existing order	EVENT OrderCanceled (consume)	InventoryService	ADR-001, ADR-004	Dispara StockRestored. La opción sincrónica /inventory/restore pasa a deprecated (ADR-004).
Canceling an existing order	EVENT OrderCanceled (consume)	NotificationService	ADR-003	Dispara SendOrderCancellation.

Tabla 12. Artefacto fase 5 BDMA - informe de trazabilidad evolutiva

- Modelo de dominio V1
- **Order (aggregate root – Ordering context)**
 - OrderId
 - CustomerId
 - Status (Created, Canceled, ...)
 - Lines: List<OrderLine>
 - Eventos: OrderCreated, OrderCanceled
- **OrderLine**
 - ProductId
 - Quantity
 - UnitPrice
- **StockItem (aggregate root – Inventory context)**
 - ProductId
 - AvailableQuantity
 - Métodos: DecreaseStock, RestoreStock
- **Payment (aggregate root – Payment context)**
 - PaymentId
 - CustomerId
 - Amount, Currency
 - Status (Authorized, Failed)
- **Notification (concepto – Notification context)**
 - NotificationId
 - OrderId
 - CustomerId
 - Channel (Email, SMS, etc.)
 - Template

6.5. Beneficios de implementar BDMA

- Trazabilidad completa desde escenarios BDD hasta microservicios: BDMA asegura que cada servicio, contrato, evento y decisión arquitectónica pueda rastrearse hasta uno o más escenarios funcionales. Esto permite un vínculo directo entre el comportamiento esperado del sistema y su estructura técnica, facilitando auditoría, mantenimiento y validación continua.
- Mejora profunda en la comprensión del dominio: al iniciar el diseño desde escenarios funcionales y escenarios colaborativos, BDMA promueve conversaciones ricas entre expertos del negocio y técnicos. El análisis sistemático de comandos, eventos y entidades permite construir un entendimiento compartido y detallado del dominio, reduciendo ambigüedades y malentendidos.
- Reducción significativa de deuda técnica y malas particiones: la identificación de *bounded contexts* a partir de comportamientos reales evita fragmentaciones arbitrarias y duplicación de responsabilidades. Al mantener una conexión

explícita con el dominio, BDMA previene la aparición temprana de “*bad smells*”, favoreciendo servicios cohesivos, modulares y sostenibles en el tiempo.

- Adaptación controlada ante cambios de requisitos: BDMA incorpora mecanismos de evolución arquitectónica mediante pruebas ejecutables, artefactos versionados y registros de decisiones. Ante cambios funcionales, los equipos pueden evaluar el impacto de manera ordenada, refactorizar *bounded contexts*, ajustar contratos y mantener la coherencia general del sistema.
- Decisiones arquitectónicas fundamentadas con evidencia funcional: el uso explícito de escenarios BDD como insumo de diseño permite justificar cada servicio, *endpoint* y evento mediante evidencia verificable. Las decisiones dejan de ser intuitivas y pasan a estar respaldadas por criterios claros y por trazabilidad bidireccional.
- Integración efectiva entre DDD, BDD y arquitectura evolutiva: BDMA logra sintetizar prácticas normalmente aisladas en un proceso fluido: BDD aporta claridad funcional; DDD estructura el dominio y la modularización; y la arquitectura evolutiva asegura que el diseño se mantenga adaptable y verificado a lo largo del tiempo. El resultado es un marco coherente, iterativo y moderno.
- Mejor mantenibilidad y claridad estructural: los servicios diseñados con BDMA exhiben responsabilidades bien delimitadas, contratos explícitos y dependencias controladas. Esto facilita la lectura, el *debugging*, la refactorización y la incorporación de nuevas funcionalidades sin comprometer la integridad del sistema.
- Documentación estructurada pero liviana mediante ADRs: el uso sistemático de ADRs transforma las decisiones en conocimiento institucional perdurable, sin generar burocracia. Cada cambio arquitectónico queda registrado, contextualizado y justificado, ofreciendo transparencia y facilitando la incorporación de nuevos miembros al equipo.
- Alineación continua entre negocio y arquitectura: BDMA garantiza que la estructura técnica del sistema evolucione en la misma dirección que la lógica del negocio. La validación funcional permanente mediante escenarios ejecutables evita divergencias entre la implementación y las reglas del dominio.
- Arquitectura emergente pero controlada: El enfoque iterativo permite que la arquitectura crezca junto con el producto, pero siempre acompañada por validación y trazabilidad. Esto equilibra la flexibilidad del mundo ágil con la disciplina necesaria para sostener una arquitectura estable y de alta calidad.
- Artefactos intermedios claros, versionables y reproducibles: constituyen un sistema documental ordenado que acompaña toda la vida del proyecto. Permiten reproducir decisiones, recuperar información estratégica y mantener registros coherentes del progreso funcional y técnico.
- Mejora la calidad del modelo de dominio desde etapas tempranas: Al analizar cada escenario en detalle y derivar entidades, eventos y reglas, BDMA promueve la construcción progresiva de un modelo de dominio rico y exacto. Esto evita retrabajo posterior y reduce la necesidad de refactorizaciones estructurales.
- Reduce la ambigüedad semántica y fortalece el lenguaje ubicuo: El énfasis en escenarios colaborativos, discusiones guiadas por ejemplos y extracción

semántica refinada permite que todo el equipo utilice términos coherentes y precisos. Esto elimina interpretaciones divergentes y mejora la comunicación interna.

6.6. Cierre del ciclo de diseño

El diseño del marco metodológico BDMA se ha construido a partir de una visión iterativa, pragmática y contextualizada de los desafíos que enfrenta la identificación de microservicios en entornos ágiles *greenfield*. A través de cinco fases progresivas, la propuesta articula de forma operativa conceptos de DDD, BDD y técnicas de análisis semántico, permitiendo derivar decisiones arquitectónicas desde el comportamiento funcional esperado del sistema.

Cada fase cumple un rol específico y se apoya en artefactos concretos que aseguran la trazabilidad y la participación de los actores del negocio. Desde la exploración inicial del dominio hasta la refactorización arquitectónica guiada por validaciones funcionales, BDMA promueve una arquitectura emergente pero controlada, que se adapta a los cambios sin comprometer la coherencia semántica ni técnica del sistema.

A diferencia de otros enfoques existentes, como GreenMicro (Bajaj et al., 2022), que operan sobre especificaciones previas y modelos cerrados, BDMA fue diseñada para funcionar desde etapas tempranas del desarrollo, incluso en ausencia de requisitos completos. Su énfasis en el lenguaje ubicuo, la validación iterativa y la evolución progresiva del diseño, busca mejorar la alineación entre código y dominio, y facilitar la toma de decisiones en equipos multidisciplinarios.

Este capítulo ha documentado en detalle el proceso de diseño del artefacto propuesto, los principios que lo sustentan y los elementos que lo componen. En el capítulo siguiente, se abordará el ciclo de consolidación, donde se describirá el procedimiento de demostración y evaluación de la propuesta, validando su utilidad, aplicabilidad y alineación con los objetivos definidos.

Capítulo 7. Caso de aplicación del marco BDMA

Este capítulo documenta la aplicación del marco BDMA en un contexto realista: una plataforma distribuida de gestión editorial científica. El objetivo es evaluar el marco en un entorno que reproduce desafíos típicos de proyectos *greenfield* ágiles. La aplicación se realizó siguiendo iterativamente las cinco fases del marco BDMA, documentando artefactos, decisiones y aprendizajes.

Podrá accederse a una versión resumida de caso de aplicación en el siguiente enlace: <https://nbattaglia.github.io/bdma/>

7.1. Contexto

Una organización académica internacional quiere desarrollar una plataforma distribuida para la gestión integral de publicaciones científicas en revistas y congresos, abierta a investigadores de todo el mundo. La plataforma debe permitir a autores, revisores, editores y organizadores de eventos interactuar de manera asincrónica y conforme a reglas definidas por políticas editoriales dinámicas.

La organización está interesada en garantizar la escalabilidad del sistema, facilitar la evolución de las reglas editoriales sin afectar la estabilidad general y permitir que equipos de desarrollo distribuidos trabajen sobre distintas partes del sistema.

7.1.1. Requisitos funcionales

La plataforma contempla un conjunto integral de funcionalidades orientadas a la gestión del ciclo completo de publicaciones científicas. En primer lugar, se abordan las necesidades vinculadas a la gestión de autores y publicaciones, permitiendo el registro y autenticación de investigadores, así como la creación de perfiles personalizados con información de afiliación institucional, identificadores ORCID, áreas temáticas de interés y métricas de impacto asociadas. Los autores pueden enviar artículos tanto a congresos como a revistas científicas, con soporte para el versionado de los envíos y control automatizado de fechas límite.

En cuanto a la gestión de revisiones, la plataforma facilita el registro de revisores junto con sus especialidades, permitiendo a los editores realizar asignaciones de manera automática o manual. Se contemplan distintos tipos de revisión ciega (simple o doble), así como el acceso al historial completo de revisiones efectuadas sobre cada artículo.

Dentro del eje de gestión editorial y de conferencias, se habilita el alta de nuevas revistas y eventos científicos, junto con la configuración detallada de reglas editoriales, que incluyen la cantidad de revisores requeridos, fechas clave, criterios de aceptación y modalidad de revisión. La plataforma también permite la administración de los editores y comités científicos, incluyendo la emisión automática de cartas de aceptación y certificados personalizados.

Respecto de la gestión de métricas e informes, se contemplan funcionalidades para generar indicadores por autor, artículo, evento y revista, así como la exportación de dichos datos en formatos estándar como CSV, JSON o XML. Además, se incluyen

mecanismos para el envío de notificaciones automáticas, incluyendo un resumen mensual personalizado para cada usuario.

Por último, la plataforma incorpora capacidades de integración con sistemas externos, habilitando la sincronización de datos con fuentes como ORCID, Scopus y Google Scholar. También se prevé la validación automática de identificadores DOI y metadatos asociados a cada publicación, asegurando la consistencia y trazabilidad de los registros científicos.

7.2. Aplicación del marco BDMA

Este caso ilustra cómo BDMA integra descubrimiento funcional (escenarios BDD), análisis del dominio (comandos, eventos, entidades) y decisiones arquitectónicas (*bounded contexts*, contratos API) en un mismo flujo iterativo.

7.2.1. Fase 1: Clasificación funcional de escenarios

7.2.1.1. Objetivo

Organizar el backlog funcional (especificado mediante escenarios BDD validados) en agrupaciones coherentes que representen funcionalidades, procesos de negocio o epics. Esta organización inicial sirve como insumo para la delimitación de *bounded contexts* en fases posteriores. En el caso de estudio utilizamos un LLM (Chat GPT) para *clusterizar* los escenarios BDD.

7.2.1.2. Entradas

- Backlog funcional expresado en escenarios BDD redactados con lenguaje Gherkin (*Given-When-Then*). (Ver anexo A)
- Metainformación opcional sobre los escenarios (*tags*, *epics*, *user stories* asociadas) (Ver anexo A)

7.2.1.3. Artefactos Generados

7.2.1.3.1. Mapa de agrupamiento funcional

nombre	escenarios	Actor Principal	eventosClave (revisados)	notas
Submissions	envio-evento, envío-revista, deadline, versionado	Autor	EnvioRegistrado; EnvioRechazadoPorDeadline; VersionAdjuntada	Depende de políticas (fechas/tipo o revisión); fuente de eventos para Reviewing/Decisioning
Reviewing	alta-revisor, conflictos, auto-assign, manual-	Revisor / Editor	RevisorRegistrado; ConflictoRegistrado; RevisoresAsignados;	Ceguera: modelar como

	assign, ceguera-simple, aceptacion, declinacion, entrega, historial		RevisorAsignadoManual ; InvitacionAceptada; InvitacionDeclinada; RevisionRegistrada	atributo/fl ag en eventos de asignación (no evento propio). <i>historial</i> es consulta
Editorial Decisioning	decision, aceptacion-final	Editor	DecisionEmitida; ExpedienteCerrado ; CertificadoEmitido	Consolida revisiones + reglas (quórum/umbral); readiness para DOI/publicación
Venue Management	alta-revista, alta-evento, versionado-reglas, roles-editoriales	Administrador Editorial	VenueCreado; EventoCreado ; ReglasPublicadas; RolesAsignados	Fuente de verdad de políticas inmutables por versión
Identity & Access	registro-exitoso, autenticacion-ok, autenticacion-falla, bloqueo	Investigador	CuentaActivada; SesionIniciada ; AccesoDenegado; CuentaBloqueada	SesionIniciada puede exponerse también como AccesoConcedido/TokenEmitido (alias)
Profiles	perfil-basico, sincronizacion-metricas	Investigador	PerfilActualizado; MetricasSincronizadas	Aporta afiliación/áreas a asignación; guarda fuente/fecha de métricas

Metrics & Reporting	metricas-autor, metricas-revista, export, boletin	Analista / Sistema	ReporteGenerado; ExportacionLista; BoletinEnviado	Consume eventos de varios contextos
External Integrations	orcid, scopus-scholar, doi	Sistema	PerfilSincronizado; PublicacionesImportadas; DOIVerificado; MetadatosValidados	Normaliza/valida datos externos

Tabla 13. Mapa de agrupamiento funcional (Artefacto fase 1 BDMA)

7.2.1.4. Resultado de la Fase 1

Se identificaron 8 grupos funcionales coherentes que representan áreas de responsabilidad bien delimitadas. Cada grupo muestra alta cohesión interna en sus escenarios y separación clara de responsabilidades con otros grupos, siendo candidatos naturales para evolucionar hacia *bounded contexts* independientes en la Fase 3.

La realización de esta primera fase permitió evidenciar uno de los beneficios centrales de BDMA: la capacidad de organizar el comportamiento funcional del sistema de manera clara y comprensible desde etapas tempranas. A partir del análisis semántico de los escenarios redactados en lenguaje natural, surgieron agrupamientos funcionales coherentes que reflejan procesos reales del negocio y no divisiones técnicas arbitrarias. Esta etapa inicial no solo facilitó la comprensión colectiva del dominio, sino que también estableció un punto de partida sólido para las fases posteriores, demostrando que la claridad estructural en la arquitectura no depende del tamaño del problema, sino del método aplicado para descubrir el comportamiento del sistema.

7.2.2. Fase 2: Extracción colaborativa de elementos del dominio

7.2.2.1. Objetivo

Identificar, desde los escenarios BDD agrupados, los actores, comandos, eventos y entidades que estructuran el dominio funcional. Esta extracción es la base para construir *bounded contexts* significativos y modelos de dominio consistentes.

7.2.2.2. Entradas

- Escenarios BDD organizados por agrupamientos funcionales (salida de Fase 1)
- Mapa funcional preliminar con tags de *features*, *epics* o procesos
- Conocimiento contextual del negocio (proporcionado por PO o expertos)

7.2.2.3. Artefactos Generados

7.2.2.3.1. Mapa de Elementos del Dominio por Escenario

Grupo funcional	Escenario	Actor	Comando (When)	Evento (Then)	Entidades (Given)
-----------------	-----------	-------	----------------	---------------	-------------------

G01 – Identity & Access	registro-exitoso	Investigador	RegistrarInvestigador	CuentaActivada	Cuenta(Investigador), ORCID, EmailVerificación
G01 – Identity & Access	autenticacion-ok	Investigador	IniciarSesion	SesionIniciada (AccesoConcedido)	Cuenta, Credenciales
G01 – Identity & Access	autenticacion-falla	Investigador	IniciarSesion	AccesoDenegado	Cuenta, Credenciales
G01 – Identity & Access	bloqueo	Sistema	BloquearCuenta	CuentaBloqueada	Cuenta, PoliticaSeguridad, IntentosFallidos
G02 – Profiles	perfil-basico	Investigador	ActualizarPerfil	PerfilActualizado	Perfil, Afiliacion, AreasInteres, ORCID
G02 – Profiles	sincronizacion-metricas	Sistema	SincronizarMetricas	MetricasSincronizadas	Perfil, FuenteMetricas(Scopus/Scholar), ORCID
G03 – Submissions	envio-evento	Autor	RegistrarEnvioEvento	EnvioRegistrado	Envio, Autor, Evento, Version
G03 – Submissions	envio-revista	Autor	RegistrarEnvioRevista	EnvioRegistrado	Envio, Autor, Revista, Version
G03 – Submissions	deadline	Autor	RegistrarEnvio	EnvioRechazado PorDeadline	Envio, ReglasEditoriales(version), FechaLimite
G03 – Submissions	versionado	Autor	AdjuntarVersion	VersionAdjuntada	Envio, Version, Archivo, Metadatos

G04 – Reviewing	alta-revisor	Revisor	RegistrarRevisor	RevisorRegistrado	Revisor, Especialidades, Afiliacion
G04 – Reviewing	conflictos	Revisor	DeclararConflicto	ConflictoRegistrado	Revisor, Envio, Autor, RelacionConflicto
G04 – Reviewing	auto-assign	Editor/Sistema	EjecutarAutoAsignacion	RevisoresAsignados	Envio, RequisitosRevision(n), Especialidades, Conflictos
G04 – Reviewing	manual-assign	Editor	AsignarRevisorManual	RevisorAsignadoManual	Envio, Revisor, Editor
G04 – Reviewing	ceguera-simple	Sistema	AplicarPoliticaCeguera	RevisoresAsignados (anonimizados)	Envio, PoliticaCeguera, Revisor, Autor
G04 – Reviewing	aceptacion	Revisor	AceptarInvitacion	InvitacionAceptada	InvitacionRevision, Revisor, Envio
G04 – Reviewing	declinacion	Revisor	RechazarInvitacion	InvitacionDeclinada	InvitacionRevision, Revisor, Envio
G04 – Reviewing	entrega	Revisor	RegistrarRevision	RevisionRegistrada	Revision, Envio, Revisor, Recomendacion, Puntajes
G04 – Reviewing	historial	Editor	ConsultarHistorialRevisiones	N/A (consulta)	Envio, Revisiones
G05 – Editorial Decisioning	decision	Editor	CalcularDecisionEditorial	DecisionEmitida	Envio, Revisiones, PoliticasDecision(quorum/umbral)
G05 – Editorial Decisioning	aceptacion-final	Sistema	EmitirCertificado	ExpedienteCerrado;	Envio, Decision, DatosAutor

				CertificadoEmitido	
G06 – Venue Management	alta-revista	Admin Editorial	CrearRevista	VenueCreado (tipo=Revista)	Revista, PolíticasIniciales, FechasClave
G06 – Venue Management	alta-evento	Admin Editorial	CrearEvento	EventoCreado	Evento, PolíticasIniciales, FechasClave, Comité
G06 – Venue Management	versionado-reglas	Admin Editorial	PublicarVersionReglas	ReglasPublicadas	Venue(Revista/Evento), VersionReglas, Fechas
G06 – Venue Management	roles-editoriales	Admin Editorial	AsignarRolEditorial	RolesAsignados	Venue, Usuario, RolEditorial
G07 – Metrics & Reporting	metricas-autor	Analista	GenerarMetricasAutor	ReporteGenerado	Autor, Envios, Revisiones, Decisiones
G07 – Metrics & Reporting	metricas-revista	Analista	GenerarMetricasRevista	ReporteGenerado	Revista/Evento, Envios, Revisiones
G07 – Metrics & Reporting	export	Analista	GenerarExportacion	ExportacionLista	Dataset, Formato(CSV/JSON-LD/XML)
G07 – Metrics & Reporting	boletin	Sistema	EnviarBoletin	BoletinEnviado	Suscriptor, Periodo, ResumenMetricas
G08 – External Integrations	orcid	Sistema	SincronizarORCID	PerfilSincronizado	Perfil, ORCID

G08 – External Integrations	scopus-scholar	Sistema	ImportarPublicaciones	PublicacionesImportadas	Perfil, CuentaExterna(Scopus/Scholar), PublicacionExterna
G08 – External Integrations	doi	Sistema	ValidarDOI	DOIVerificado; MetadatosValidados	Publicacion, DOI, Metadatos

Tabla 14. Mapa de elementos del dominio por escenario (artefacto fase 2 BDMA)

7.2.2.4. Resultados de la fase 2

Se extrajeron exitosamente comandos (+35) y eventos (+28) de cada contexto funcional, estableciendo un vocabulario ubicuo claro. Los elementos extraídos muestran coherencia semántica dentro de cada grupo y diferenciación clara entre grupos, validando la clasificación de la Fase 1 y preparando la delimitación de *bounded contexts* en la Fase 3.

La aplicación de BDMA en esta fase permitió observar cómo la comprensión del dominio se profundiza cuando los escenarios BDD son analizados colaborativamente. El proceso de identificar actores, comandos, eventos y entidades derivó en un lenguaje compartido, preciso y libre de ambigüedades. La solidez conceptual lograda no surge de la simplicidad del caso, sino del mecanismo mismo de BDMA para transformar narrativas funcionales en elementos del dominio bien definidos. Esta extracción sistemática aporta uno de los beneficios esenciales del marco: un modelo de dominio robusto y fundamentado, que reduce inconsistencias y prepara una base semántica estable para la identificación de microservicios.

7.2.3. Fase 3: Delimitación y formalización de *bounded contexts*

7.2.3.1. Objetivos

Agrupar comandos, eventos y entidades relacionados (extraídos de los escenarios BDD) en *bounded contexts* coherentes, explícitamente delimitados, que permitan separar responsabilidades, minimizar acoplamientos y sostener la evolución independiente de los servicios.

7.2.3.2. Entradas

- Mapa de agrupamientos funcionales (artefacto de Fase 1)
- Mapa de elementos del dominio por escenario (artefacto de Fase 2)
- Conocimiento contextual de dependencias organizacionales y técnicas

7.2.3.3. Artefactos Generados

7.2.3.3.1. Mapa de Bounded Context (artefacto Fase 3 BDMA)

Contexto	Comandos	Eventos	Entidades	API
----------	----------	---------	-----------	-----

BC01 – Identity & Access	RegistrarInvestigador, IniciarSesion, BloquearCuenta	CuentaActivada, SesionIniciada, AccesoDenegado, CuentaBloqueada	Cuenta, Credenciales, PoliticaSeguridad	Sí
BC02 – Profiles	ActualizarPerfil, SincronizarMetricas	PerfilActualizado, MetricasSincronizadas	Perfil, Afiliacion, AreasInteres, FuenteMetricas	Sí (lectura)
BC03 – Submissions	RegistrarEnvioEvento, RegistrarEnvioRevisita, RegistrarEnvio, AdjuntarVersion	EnvioRegistrado, EnvioRechazadoPorDeadline, VersionAdjuntada	Envio, Version, Archivo, PoliticaEditorial	No (eventos)
BC04 – Reviewing	RegistrarRevisor, DeclararConflicto, EjecutarAutoAsignacion, AsignarRevisorManual, AceptarResultadoInvitacion, RechazarInvitacion, RegistrarRevision	RevisorRegistrado, ConflictoRegistrado, RevisoresAsignados, RevisorAsignadoManual, InvitacionAceptada, InvitacionDeclinada, RevisionRegistrada	Revisor, InvitacionRevision, Revision, PoliticasyAsignacion	No (eventos)
BC05 – Editorial Decisioning	CalcularDecisionEditorial, EmitirCertificado	DecisionEmitida, ExpedienteCerrado, CertificadoEmitido	PoliticasyDecision, Certificado, Expediente	No (eventos)
BC06 – Venue Management	CrearRevista, CrearEvento, PublicarVersionReglas, AsignarRolEditorial	VenueCreado, EventoCreado, ReglasPublicadas, RolesAsignados	Revista, Evento, VersionReglas, RolEditorial	Sí
BC07 – Metrics & Reporting	GenerarMetricasAutor, GenerarMetricasRevista,	ReporteGenerado, ExportacionLista, BoletinEnviado	Reporte, Formato, Filtro, Periodo	No

	GenerarExportacion, EnviarBoletin			
BC08 – External Integrations	SincronizarORCID, ImportarPublicaciones, ValidarDOI	PerfilSincronizado, PublicacionesImportadas, DOIVerificado, MetadatosValidados	FuenteExterna, Publicacion, DOI, Metadatos	No

Tabla 15. Mapa de bounded context (artefacto fase 3 BDMA)

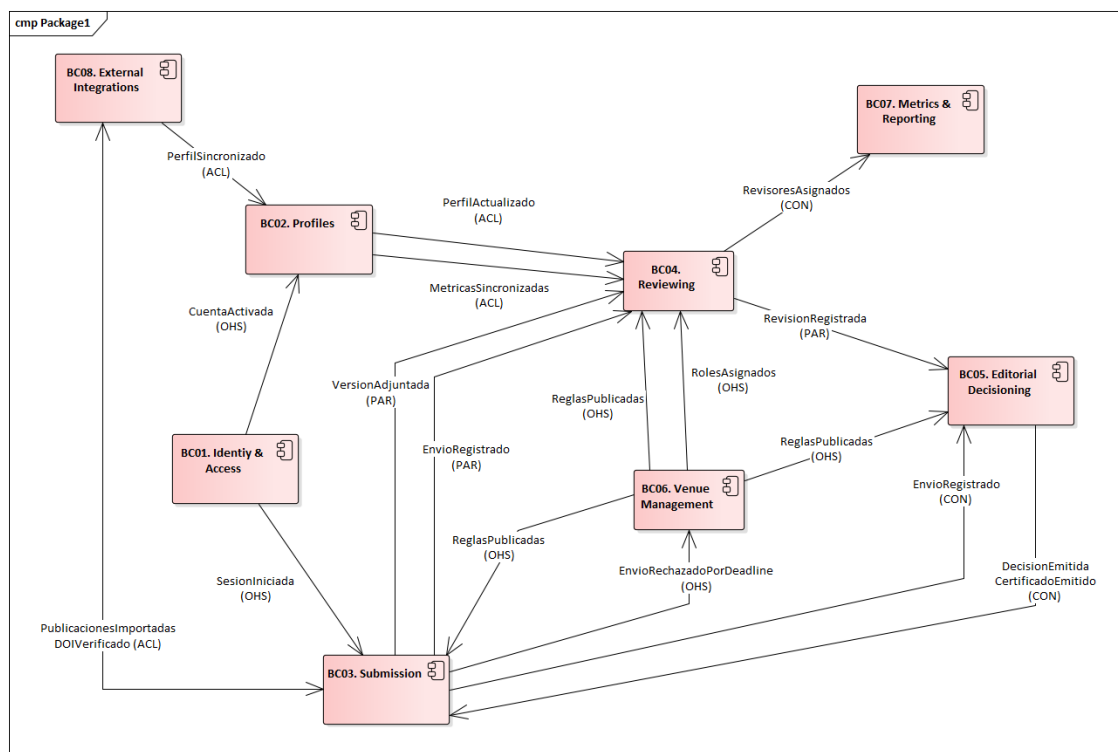


Ilustración 8. Mapa visual de bounded context del caso de estudio de BDMA

7.2.3.3.2. Mapa de Interacciones entre Contextos

Contexto Origen	Evento	Contexto Destino	Tipo de relación	Medio de interacción	Notas
BC01 Identity & Access	CuentaActivada	BC02 Profiles	OHS	Async · topic identity.cuentaActivada	Habilita creación/gestión de perfil
BC01 Identity & Access	SesionIniciada	BC03 Submissions	OHS	Async · topic identity.sesionIniciada	Autoriza operaciones de envío

BC02 Profiles	PerfilActualizado	BC04 Reviewing	ACL	Async · topic profiles.perfilActualizado	Afiliación/áreas para matching de revisores
BC02 Profiles	MetricasSincronizadas	BC04 Reviewing	ACL	Async · topic profiles.metricasSincronizadas	Ajusta ponderación de asignaciones
BC06 Venue Management	ReglasPublicadas	BC03 Submissions	OHS	Async · topic venue.reglasPublicadas	Políticas/fechas para envíos
BC06 Venue Management	ReglasPublicadas	BC04 Reviewing	OHS	Async · topic venue.reglasPublicadas	Criterios para revisión
BC06 Venue Management	ReglasPublicadas	BC05 Editorial Decisioning	OHS	Async · topic venue.reglasPublicadas	Umbrales/criterios de decisión
BC06 Venue Management	RolesAsignados	BC04 Reviewing	OHS	Async · topic venue.rolesAsignados	Asignación de editores/revisores
BC03 Submissions	EnvioRegistrado	BC04 Reviewing	Partnership	Async · topic submissions.envioRegistrado	Dispara proceso de revisión
BC03 Submissions	EnvioRegistrado	BC05 Editorial Decisioning	Conformist	Async · topic submissions.envioRegistrado	Sincroniza estado del envío
BC03 Submissions	VersionAdjuntada	BC04 Reviewing	Partnership	Async · topic submissions.versionAdjuntada	Nueva versión disponible para revisar

BC03 Submissions	EnvioRechazadoPorDeadline	BC06 Venue Management	OHS	Async · topic submissions.envioRechazadoPorDeadline	Notifica incumplimiento de regla
BC04 Reviewing	RevisionRegistrada	BC05 Editorial Decisioning	Partnership	Async · topic reviewing.revisionRegistrada	Insumo directo para decisión
BC04 Reviewing	RevisoresAsignados	BC07 Metrics & Reporting	Conformist	Async · topic reviewing.revisoresAsignados	Estadísticas de carga de trabajo
BC05 Editorial Decisioning	DecisionEmitida	BC03 Submissions	Conformist	Async · topic decisioning.decisionEmitida	Cierra ciclo del envío
BC05 Editorial Decisioning	CertificadoEmitido	BC03 Submissions	Conformist	Async · topic decisioning.certificadoEmitido	Emite certificado de aceptación/rechazo
BC08 External Integrations	PerfilSincronizado	BC02 Profiles	ACL	Async · topic external.perfilSincronizado	Actualiza datos ORCID/Scholar
BC08 External Integrations	PublicacionesImportadas	BC03 Submissions	ACL	Async · topic external.publicacionesImportadas	Carga antecedentes bibliográficos
BC08 External Integrations	DOIVerificado	BC03 Submissions	ACL	Async · topic external.doiVerificado	Valida metadatos/DOI
BC07 Metrics & Reporting	ReporteGenerado	—	Conformist	Async · topic metrics.reporteGenerado	Evento de auditoría/telemetría (sin

					consumidores de dominio en esta iteración)
--	--	--	--	--	--

Tabla 16. Mapa de interacciones entre contextos (artefacto fase 3 BDMA)

7.2.3.3.3. Identificación de contextos que requieren contratos API

Contexto	¿Consumido por otros?	¿Publica eventos?	Tipo de contrato requerido en F4	Consumidores principales
BC01 – Identity & Access	Sí	No	REST (Auth, OAuth2/JWT)	Todos (front, gateways, servicios)
BC02 – Profiles	Sí (solo lectura)	Sí (PerfilActualizado, MetricasSincronizadas)	REST (read-only) + Esquemas de eventos	BC04 (Reviewing), UI
BC03 – Submissions	No (principalmente asíncrono)	Sí (EnvioRegistrado, VersionAdjuntada, EnvioRechazadoPorDeadline)	Esquemas de eventos	BC04, BC05, BC07
BC04 – Reviewing	No (principalmente asíncrono)	Sí (RevisoresAsignados, RevisionRegistrada, etc.)	Esquemas de eventos	BC05, BC07
BC05 – Editorial Decisioning	No (asíncrono)	Sí (DecisionEmitida, ExpedienteCerrado)	Esquemas de eventos	BC03, BC07
BC06 – Venue Management	Sí	Sí (ReglasPublicadas)	REST (reglas/roles) + Esquemas de eventos	BC03, BC04, BC05, UI

BC07 – Metrics & Reporting	Opcional (descargas/export)	Sí (ReporteGenerado, ExportacionLista)	REST opcional (descarga/export) + Esquemas de eventos	UI, integraciones BI
BC08 – External Integrations	No (actúa como cliente)	Sí (PerfilSincronizado, PublicacionesImportadas, DOIVerificado)	Esquemas de eventos	BC02, BC03, BC07

Tabla 17. Identificación de contextos que requieren API (artefacto fase 3 BDMA)

7.2.3.4. Resultados de la Fase 3

Se delimitaron 8 *bounded contexts* con responsabilidades claras y bien definidas. El *context map* resultante muestra patrones de integración apropiados que minimizan el acoplamiento y maximizan la autonomía de los equipos. Las interacciones están explícitamente documentadas y siguen los patrones estratégicos de DDD.

En esta fase se hace evidente cómo BDMA contribuye a evitar particiones incorrectas y reduce la deuda técnica desde el inicio. La agrupación natural de los elementos identificados en *bounded contexts* coherentes muestra que los límites de los microservicios emergen del comportamiento del negocio y no de decisiones técnicas prematuras. La claridad de responsabilidades, la ausencia de solapamientos y la identificación temprana de relaciones entre contextos demuestran que BDMA conduce hacia una modularidad correcta incluso en dominios complejos. Este resultado no depende del tamaño del caso, sino de la trazabilidad semántica y funcional que el método garantiza.

7.2.4. Fase 4: Derivación de interfaces y contratos API

7.2.4.1. Objetivos

Traducir los comandos y eventos definidos en escenarios BDD, dentro de cada *bounded context*, en contratos de servicio (API) que sean verificables, trazables y diseñados para pruebas automatizadas. Esta fase garantiza que la arquitectura técnica respete las especificaciones funcionales y permita evolución segura.

7.2.4.2. Entradas

- Mapa de elementos del dominio (artefacto de Fase 2)
- Mapa de *bounded contexts* (artefacto de Fase 3)
- Reglas de interacción entre contextos
- Escenarios BDD originales como fuente de trazabilidad

7.2.4.3. Artefactos Generados

7.2.4.3.1. Especificación de contratos API con trazabilidad funcional

Contexto	Endpoint	Verbo	Comando / Evento	Escenario BDD	Observaciones
----------	----------	-------	------------------	---------------	---------------

Identity & Access	/auth/register	POST	Cmd: RegistrarCuenta → Ev: CuentaActivada	registro-exitoso	IdP externo (ORCID/email); 201/409; idempotencia por email.
Identity & Access	/auth/login	POST	Cmd: Autenticar → Ev: SesionIniciada / AccesoDenegado	autenticacion-ok / autenticacion-falla	Devuelve JWT; 200/401; rate-limit.
Identity & Access	/auth/lock/{accountId}	POST	Cmd: BloquearCuenta → Ev: CuentaBloqueada	bloqueo	Solo sistema/seguridad; 202; auditoría.
Identity & Access	/auth/me	GET	Consulta identidad	(derivado de autenticacion-ok)	Requiere Bearer; 200.
Profiles	/profiles/{researcherId}	PUT	Cmd: ActualizarPerfil → Ev: PerfilActualizado	perfil-basico	Valida ORCID opcional; 200/400; control de propiedad.
Profiles	/profiles/{researcherId}/metrics:sync	POST	Cmd: SincronizarMetricas → Ev: MetricasSincronizadas	sincronizacion-metricas	Fuentes ORCID/Scopus/Scholar; 202 async; guarda fuente/fecha.
Submissions	/venues/{venueId}/submissions	POST	Cmd: RegistrarEvento → Ev: EnvioRegistrado / EnvioRechazadoPorDeadline	envio-evento / envio-revista / deadline	Valida reglas versionadas; 201 o 422 (deadline).

Submissions	/submissions/{submissionId}/versions	POST	Cmd: AdjuntarVersion → Ev: VersionAdjuntada	versionado	Historial de versiones; 201; checksum.
Submissions	/submissions/{submissionId}	GET	Consulta envío	(apoya reviewing/ decision)	Incluye versión activa y estado; 200.
Reviewing	/reviewers	POST	Cmd: RegistrarRevisor → Ev: RevisorRegistrado	alta- revisor	Registra especialidades; 201.
Reviewing	/reviewers/{id}/conflicts	POST	Cmd: DeclararConflicto → Ev: ConflictoRegistrado	conflictos	Afecta matching; 201.
Reviewing	/submissions/{submissionId}/assignments:auto	POST	Cmd: AsignarRevisoresAuto → Ev: RevisoresAsignados	auto- assign	Usa políticas (n, ceguera); 202; determinística por seed.
Reviewing	/submissions/{submissionId}/assignments	POST	Cmd: AsignarRevisorManual → Ev: RevisorAsignadoManual	manual- assign	Auditoría obligatoria; 201.
Reviewing	/assignments/{assignmentId}/accept	POST	Cmd: AceptarInvitación → Ev: InvitacionAceptada	aceptación	200; expira por fecha límite.
Reviewing	/assignments/{assignmentId}/decline	POST	Cmd: DeclinarInvitación → Ev: InvitacionDeclinada	declinación	200; disparar reasignación.

Reviewing	/reviews	POST	Cmd: RegistrarRevision → Ev: RevisionRegistrada	entrega	Adjunta recomendación/score; ceguera por política; 201.
Reviewing	/submissions/{submissionId}/reviews	GET	Consulta historial	historial	Solo editor/autorizados; ocultar identidades según ceguera; 200.
Editorial Decisioning	/submissions/{submissionId}/decision	POST	Cmd: EmitirDecision → Ev: DecisionEmitida	decision	Aplica quórum/umbral; registra versión de reglas; 200.
Editorial Decisioning	/submissions/{submissionId}/acceptance-letter	POST	Cmd: EmitirCertificado → Ev: CertificadoEmitido	aceptacion-final	Genera carta/certificado; 201; repositorio de docs.
Venue Management	/venues	POST	Cmd: CrearVenue → Ev: VenueCreado	alta-revista / alta-evento	Tipo={revista,event}; 201.
Venue Management	/venues/{venueId}/policies	POST	Cmd: PublicarReglas → Ev: ReglasPublicadas	versionado-reglas	Inmutable por versión; 201; fecha de vigencia.
Venue Management	/venues/{venueId}/roles	POST	Cmd: AsignarRoles → Ev: RolesAsignados	roles-editoriales	Asigna editores/comités; 201.
Metrics & Reporting	/reports/authors	GET	Consulta KPIs autor → Ev: (ReporteGenerado)	metricas-autor	Params authorId, periodo; 200; cacheable.

Metrics & Reporting	/reports/venues	GET	Consulta KPIs venue → Ev: (ReporteGenerado)	metricas-revista	Params venueId, periodo; 200.
Metrics & Reporting	/exports	POST	Cmd: GenerarExportacion → Ev: ExportacionLista	export	CSV/JSON-LD/XML; 202 async + URL descarga/callback.
Metrics & Reporting	/digests/monthly	POST	Cmd: EnviarBoletin → Ev: BoletinEnviado	boletin	Idempotencia por periodo+destinatario; 202.

Tabla 18. Especificación de *contratos API* (artefacto fase 4 BDMA)

7.2.4.4. Resultado de la Fase 4

Se derivaron +25 contratos API siguiendo principios *contract-first*, con trazabilidad completa hacia los escenarios BDD originales. Cada *endpoint* está diseñado para ser verificable mediante pruebas automatizadas y sigue patrones REST consistentes. La distribución de *endpoints* refleja la complejidad relativa de cada *bounded context*.

La traducción de los comandos y eventos del dominio en contratos API verificables deja en evidencia otro beneficio clave de BDMA: la capacidad de diseñar interfaces precisas, alineadas al negocio y sostenibles en el tiempo. Cada contrato surge como una consecuencia lógica de un escenario funcional, lo que evita la proliferación de *endpoints* sin propósito claro o inconsistentes entre sí. Esta alineación directa entre necesidad funcional y exposición técnica fortalece la mantenibilidad del sistema y asegura que la arquitectura del caso pueda evolucionar de manera controlada, independientemente del tamaño o la complejidad del dominio.

7.2.5. Fase 5: Validación continua y evolución controlada

7.2.5.1. Objetivos

Garantizar que la arquitectura de microservicios evolucione en sincronía con los cambios funcionales del sistema, manteniendo la trazabilidad completa desde los escenarios BDD, y documentando cada decisión estructural mediante mecanismos formales que habiliten la revisión, refactorización y mantenimiento.

7.2.5.2. Entradas

- Contratos API y relaciones de integración (artefacto de Fase 4)
- Mapa de *bounded contexts* versionado (artefacto de Fase 3)
- Escenarios BDD actualizados
- Historial de decisiones previas (bitácora ADR)

7.2.5.3. Artefactos Generados

7.2.5.3.1. Bitácora de decisiones arquitectónicas (ADR Log)

ADR-001 – Definición de arquitectura inicial de la plataforma académica

- **Estado:** *Aceptado.*
- **Fecha:** 2025-08-11
- **Contexto:** La plataforma académica se encuentra en etapa inicial. Se definieron escenarios BDD, grupos funcionales, bounded contexts, relaciones inter-contexto y contratos API iniciales como baseline arquitectónico.
- **Decisión:** Adoptar la arquitectura resultante de la aplicación de las fases 1–4 de BDMA como **versión inicial**.
- **Consecuencias:**
 - Esta versión será la referencia para evaluar cambios en futuras iteraciones de la Fase 5.
 - Todos los bounded contexts, modelos de dominio y contratos API definidos se consideran vigentes en versión 1.0.
- **Artefactos relacionados:**
 - Context Map v1.0
 - Modelos de dominio v1.0
 - Contratos API v1.0

7.2.5.3.2. Informe de trazabilidad evolutiva

Escenario BDD	Bounded Context	Comando (When)	Evento (Then)	Endpoint / Contrato API	ADR Asociado
registro-exitoso	BC01 – Identity & Access	RegistrarCuenta	CuentaActivada	POST /auth/register	ADR-001
autenticacion-ok	BC01 – Identity & Access	Autenticar	SesionIniciada	POST /auth/login	ADR-001
autenticacion-falla	BC01 – Identity & Access	Autenticar	AccesoDenegado	POST /auth/login	ADR-001
bloqueo	BC01 – Identity & Access	BloquearCuenta	CuentaBloqueada	POST /auth/lockouts/evaluate	ADR-001
perfil-basico	BC02 – Profiles	ActualizarPerfil	PerfilActualizado	PUT /profiles/{researcherId}	ADR-001

sincronizac ion- metricas	BC02 – Profiles	SincronizarMetrica s	MetricasSincronizad as	POST /profiles/{researcherId}/metrics/sync	ADR-001
envio- evento	BC03 – Submissions	RegistrarEnvioEve nto	EnvioRegistrado	POST /venues/{venueId}/submissions	ADR-001
envio- revista	BC03 – Submissions	RegistrarEnvioRevi sta	EnvioRegistrado	POST /journals/{journalId}/submissions	ADR-001
deadline	BC03 – Submissions	IntentarRegistrarE nvio	EnvioRechazadoPor Deadline	POST /venues/{venueId}/submissions	ADR-001
versionado	BC03 – Submissions	AdjuntarVersion	VersionAdjuntada	POST /submissions/{submissionId}/versions	ADR-001
alta-revisor	BC04 – Reviewing	RegistrarRevisor	RevisorRegistrado	POST /reviewers	ADR-001
conflictos	BC04 – Reviewing	DeclararConflicto	ConflictoRegistrado	POST /reviewers/{reviewerId}/conflicts	ADR-001
auto-assign	BC04 – Reviewing	AsignarRevisoresA utomaticamente	RevisoresAsignados	POST /submissions/{submissionId}/assignments:auto	ADR-001
manual- assign	BC04 – Reviewing	AsignarRevisorMa nual	RevisorAsignadoMan ual	POST /submissions/{submissionId}/assignments	ADR-001
ceguera- simple	BC04 – Reviewing	ConfigurarCeguera	(flag aplicado)	POST /submissions/{submissionId}/assignments	ADR-001
aceptacion	BC04 – Reviewing	AceptarInvitacion	InvitacionAceptada	POST /assignments/{assignmentId}/accept	ADR-001

declinacion	BC04 – Reviewing	DeclinarInvitacion	InvitacionDeclinada	POST /assignments/{assignmentId}/decline	ADR-001
entrega	BC04 – Reviewing	RegistrarRevision	RevisionRegistrada	POST /submissions/{submissionId}/reviews	ADR-001
historial	BC04 – Reviewing	ConsultarHistorial	—	GET /submissions/{submissionId}/reviews	ADR-001
decision	BC05 – Editorial Decisioning	CalcularDecisionEditorial	DecisionEmitida	POST /submissions/{submissionId}/decision	ADR-001
aceptacion -final	BC05 – Editorial Decisioning	EmitirCertificado	CertificadoEmitido	POST /submissions/{submissionId}/acceptance-letter	ADR-001
alta-revista	BC06 – Venue Management	CrearRevista	VenueCreado	POST /venues/journals	ADR-001
alta-evento	BC06 – Venue Management	CrearEvento	EventoCreado	POST /venues/conferences	ADR-001
versionado -reglas	BC06 – Venue Management	PublicarVersionReglas	ReglasPublicadas	POST /venues/{venueId}/rulesets	ADR-001
roles- editoriales	BC06 – Venue Management	AsignarRolEditorial	RolesAsignados	POST /venues/{venueId}/roles	ADR-001
metricas- autor	BC07 – Metrics & Reporting	GenerarMetricasAutor	ReporteGenerado	GET /reports/authors	ADR-001
metricas- revista	BC07 – Metrics & Reporting	GenerarMetricasRevista	ReporteGenerado	GET /reports/venues	ADR-001

export	BC07 – Metrics & Reporting	GenerarExportacion	ExportacionLista	POST /exports	ADR-001
boletin	BC07 – Metrics & Reporting	EnviarBoletin	BoletinEnviado	POST /digests/monthly	ADR-001
orcid	BC08 – External Integrations	SincronizarORCID	PerfilSincronizado	POST /integrations/orcid /sync	ADR-001
scopus-scholar	BC08 – External Integrations	ImportarPublicaciones	PublicacionesImportadas	POST /integrations/scholar-scopus/sync	ADR-001
doi	BC08 – External Integrations	ValidarDOI	DOIVerificado	POST /integrations/doi/validate	ADR-001

Tabla 19. Informe de trazabilidad evolutiva (atfacto fase 5 BDMA)

7.2.5.3.3. Modelo de dominio y mapa de contexto versionado

Versión inicial (v1.0) – Fecha: 2025-08-11

- **Modelos de dominio**
- **BC01 – Identity & Access**
 - Entidades: Cuenta, Credenciales, PoliticaSeguridad.
 - Eventos clave: CuentaActivada, SesionIniciada, AccesoDenegado, CuentaBloqueada.
 - Comandos: RegistrarInvestigador, IniciarSesion, BloquearCuenta.
- **BC02 – Profiles**
 - Entidades: Perfil, Afiliacion, AreasInteres, FuenteMetricas.
 - Eventos clave: PerfilActualizado, MetricasSincronizadas.
 - Comandos: ActualizarPerfil, SincronizarMetricas.
- **BC03 – Submissions**
 - Entidades: Envio, Version, Archivo, PoliticaEditorial.
 - Eventos clave: EnvioRegistrado, EnvioRechazadoPorDeadline, VersionAdjuntada.
 - Comandos: RegistrarEnvioEvento, RegistrarEnvioRevista, AdjuntarVersion.
- **BC04 – Reviewing**
 - Entidades: Revisor, InvitacionRevision, Revision, PoliticasAsignacion.
 - Eventos clave: RevisorRegistrado, ConflictoRegistrado, RevisoresAsignados, RevisionRegistrada.
 - Comandos: RegistrarRevisor, DeclararConflicto, EjecutarAutoAsignacion, AsignarRevisorManual, RegistrarRevision.
- **BC05 – Editorial Decisioning**
 - Entidades: PoliticasDecision, Certificado, Expediente.

- Eventos clave: DecisionEmitida, CertificadoEmitido, ExpedienteCerrado.
- Comandos: CalcularDecisionEditorial, EmitirCertificado.
- **BC06 – Venue Management**
 - Entidades: Revista, Evento, VersionReglas, RolEditorial.
 - Eventos clave: VenueCreado, EventoCreado, ReglasPublicadas, RolesAsignados.
 - Comandos: CrearRevista, CrearEvento, PublicarVersionReglas, AsignarRolEditorial.
- **BC07 – Metrics & Reporting**
 - Entidades: Reporte, Formato, Filtro, Periodo.
 - Eventos clave: ReporteGenerado, ExportacionLista, BoletinEnviado.
 - Comandos: GenerarMetricasAutor, GenerarMetricasRevista, GenerarExportacion, EnviarBoletin.
- **BC08 – External Integrations**
 - Entidades: FuenteExterna, Publicacion, DOI, Metadatos.
 - Eventos clave: PerfilSincronizado, PublicacionesImportadas, DOIVerificado, MetadatosValidados.
 - Comandos: SincronizarORCID, ImportarPublicaciones, ValidarDOI.

Relación entre BC (derivados de fase 3)

Origen	Destino	Tipo de relación	Medio de interacción
BC01 – Identity & Access	Todos	OHS (Open Host Service)	API Auth + JWT
BC02 – Profiles	BC04 – Reviewing	ACL	API interna
BC03 – Submissions	BC04 – Reviewing	Event Publisher / Partnership	Mensajería/eventos
BC03 – Submissions	BC05 – Editorial Decisioning	Event Publisher / Partnership	Mensajería/eventos
BC04 – Reviewing	BC05 – Editorial Decisioning	Event Publisher	Mensajería/eventos
BC05 – Editorial Decisioning	BC03 – Submissions	Event Subscriber	Mensajería/eventos
BC06 – Venue Management	BC03, BC04, BC05	OHS + Event Publisher	API y eventos
BC07 – Metrics & Reporting	Todos	Conformist	Lectura directa de proyecciones
BC08 – External Integrations	BC01, BC02, BC03	ACL	Adaptadores de datos externos

Tabla 20. Relaciones entre bounded contexts (artefacto fase 5 BDMA)

7.2.5.4. Resultado de la Fase 5

Se estableció un proceso de validación continua que garantiza la evolución sincronizada de la arquitectura con los cambios funcionales. La trazabilidad completa desde escenarios BDD hasta ADR permite mantener la coherencia estructural y facilita la toma de decisiones informada. El sistema está preparado para evolucionar de manera controlada y documentada a lo largo del ciclo de vida del producto.

La última fase del caso de estudio refleja con claridad cómo BDMA habilita una arquitectura evolutiva y gobernada. La validación continua mediante escenarios BDD ejecutables, el registro explícito de decisiones arquitectónicas y el versionado del mapa de contextos permiten que la solución evolucione sin comprometer su integridad conceptual. Este enfoque demuestra que los cambios pueden ser incorporados sin reestructuraciones disruptivas, lo cual es uno de los beneficios más significativos del marco. La capacidad de mantener coherencia y trazabilidad durante la evolución arquitectónica no se debe al tamaño del problema, sino a los mecanismos formales que BDMA incorpora.

7.3. Evidencia de los beneficios de BDMA en el caso de estudio

A continuación, describimos cada uno de los beneficios identificados y documentados en el capítulo 6, con el objetivo de comprender el impacto y validez con un caso real

- Trazabilidad completa desde escenarios BDD hasta microservicios: en el caso de estudio, cada elemento arquitectónico —como comandos, eventos, contextos delimitados o contratos de servicio— surgió directamente de los escenarios BDD redactados al inicio. Esto permitió mantener un hilo conductor claro entre lo que el sistema debía hacer y cómo la arquitectura se organizaba para cumplirlo. La trazabilidad fue natural y emergió de la metodología, sin esfuerzo adicional, demostrando que BDMA garantiza una correspondencia directa entre comportamiento funcional y diseño técnico, sin que ello dependa de la escala del dominio.
- Mejora en la comprensión del dominio: el análisis colaborativo de los escenarios de comportamiento permitió que los participantes identificaran y acordaran rápidamente los conceptos centrales del negocio, los actores relevantes y las responsabilidades principales. Esta comprensión compartida se mantuvo a lo largo de todo el diseño, evitando ambigüedades o interpretaciones divergentes. El fenómeno no se debió al tamaño del caso, sino a que BDMA obliga a explorar primero el comportamiento real del sistema antes de pensar en microservicios.
- Reducción de deuda técnica y particiones incorrectas: a medida que el caso avanzaba, los límites de los microservicios surgieron de manera estable y coherente. No aparecieron servicios redundantes, solapados o excesivamente granulares; por el contrario, cada servicio mostró una responsabilidad clara y bien acotada. Esto revela que BDMA ayuda a establecer límites correctos desde el principio y reduce la posibilidad de malos olores arquitectónicos, independientemente de la complejidad del dominio.
- Adaptación controlada ante cambios de requisitos: cuando se introdujeron ajustes o escenarios nuevos, la arquitectura pudo evolucionar sin necesidad de replantear toda la estructura. Los cambios se absorbieron ajustando límites,

contratos o reglas específicas, siempre con un registro explícito de las decisiones tomadas. El diseño se mantuvo coherente, validado por los escenarios existentes. Esto confirma que BDMA facilita una evolución ordenada del sistema incluso frente a dominios cambiantes, mostrando que el beneficio proviene del método y no del tamaño del problema.

- Decisiones arquitectónicas fundamentadas con evidencia funcional: durante el diseño, cada decisión tuvo una justificación funcional explícita. Los participantes podían explicar por qué un servicio se comportaba de determinada manera, qué escenario motivaba su existencia o por qué se eligió un estilo de integración. La arquitectura no surgió de preferencias personales, sino de necesidades funcionales concretas derivadas de los escenarios. Esto demuestra que BDMA permite tomar decisiones fundamentadas desde etapas tempranas.
- Integración efectiva entre DDD, BDD y arquitectura evolutiva: en la práctica, el caso mostró cómo BDMA entrelaza de manera natural el descubrimiento funcional de BDD, el modelado estratégico de DDD y los principios de la arquitectura evolutiva. La metodología guía al equipo desde la comprensión del comportamiento, pasando por el análisis del dominio, hasta la definición de los contextos y contratos. Esta integración no depende del tamaño del proyecto; se evidencia porque BDMA estructura el camino para llegar a ella.
- Mayor mantenibilidad y claridad estructural: el resultado final mostró un conjunto de microservicios fáciles de entender, con roles bien definidos y sin responsabilidades superpuestas. La claridad de la estructura permitió visualizar con facilidad cómo nuevos escenarios podrían incorporarse sin comprometer los componentes existentes. Esta mantenibilidad es inherente a la forma en la que BDMA conduce la identificación iterativa de responsabilidades y no está condicionada por el tamaño del dominio.
- Documentación estructurada y gobernanza ligera: el uso de decisiones arquitectónicas registradas de manera sintética permitió que cada cambio quedara documentado sin generar burocracia adicional. Esto facilitó comprender la evolución del diseño y preservó el razonamiento detrás de cada elección técnica. La documentación fue lo suficientemente ligera como para no entorpecer el flujo de trabajo, reflejando uno de los principios clave de BDMA.
- Alineación continua entre negocio y arquitectura: a lo largo del caso, cada paso del diseño mantuvo una relación clara con los objetivos del negocio. La estructura arquitectónica se mantuvo fiel a la lógica funcional expresada en los escenarios, evitando desviaciones típicas en las que la arquitectura evoluciona de manera desconectada del dominio. Esta alineación ocurrió naturalmente por la secuencia que BDMA propone (primero el comportamiento, luego el dominio, luego los servicios), demostrando que no depende de la magnitud del sistema.
- Arquitectura emergente pero controlada: la arquitectura no se definió de antemano ni se improvisó. Fue emergiendo de la sucesiva interpretación y validación de los escenarios, pero siempre guiada por mecanismos que permitieron corregir o refinar decisiones sin perder coherencia. El equilibrio entre libertad y control es una propiedad central de BDMA, evidenciado en el caso al evitar tanto el diseño rígido como la arquitectura accidental.

- Artefactos reproducibles y versionables: el caso mostró cómo BDMA conduce a producir artefactos intermedios claros —agrupaciones funcionales, elementos del dominio, contextos, contratos y decisiones— que pueden revisarse, versionarse y utilizarse como insumos en iteraciones posteriores. Esta reproducibilidad no depende del tamaño del proyecto sino del diseño metodológico, que asegura coherencia entre cada artefacto.
- Adecuado para equipos con diversidad de experiencia: la implementación del caso involucró participantes con diferentes niveles de experiencia, y aun así lograron una arquitectura consistente. BDMA brindó el marco necesario para que todos trabajaran con la misma estructura de razonamiento, evitando disparidades en las decisiones arquitectónicas. Esto muestra que el beneficio proviene de la guía explícita del proceso y no de la complejidad moderada del dominio.
- Mejora del modelo de dominio desde etapas tempranas: desde los primeros pasos del caso, el modelo del dominio surgió nítido y sin contradicciones. Las entidades, sus relaciones y los eventos asociados se fueron refinando a través del análisis de los escenarios, generando un modelo robusto desde el comienzo. Esta mejora es consecuencia del enfoque iterativo de BDMA, no del tamaño del problema analizado.
- Reducción de ambigüedad semántica y fortalecimiento del lenguaje ubicuo: el caso permitió trabajar con un vocabulario compartido que se sostuvo en todas las fases del diseño. Términos funcionales, nombres de eventos, acciones y estados se mantuvieron coherentes entre los escenarios, los contextos y las interfaces, reduciendo ambigüedades semánticas. Esto no surgió espontáneamente: fue consecuencia del proceso que BDMA impone para construir el lenguaje del sistema desde el comportamiento.

Capítulo 8. Ciclo de consolidación

El presente capítulo tiene como objetivo consolidar la evaluación del marco metodológico propuesto, denominado BDMA, mediante una estrategia que contempla la evaluación empírica de su aplicación en contextos de uso reales o simulados.

8.1. Estrategia de evaluación

De acuerdo con (Hevner, 2007) la calidad de un artefacto de diseño no se determina solo por su formulación teórica, sino por su capacidad para resolver un problema real, su utilidad y su coherencia con el conocimiento científico existente. Bajo esta premisa, la evaluación de BDMA se centra en analizar tanto la efectividad del proceso como la completitud estructural del método, integrando criterios técnicos, funcionales y epistemológicos. La estrategia adoptada compara la aplicación de BDMA con enfoques no estructurados, observando su impacto en la identificación y diseño de microservicios, así como en la alineación con el dominio, la validación funcional y la experiencia de los participantes. Para ello, se emplea un diseño experimental en el que un grupo de estudiantes resuelve un mismo problema primero sin guía metodológica y luego siguiendo la documentación de BDMA, complementado con rúbricas técnicas y encuestas de percepción (Runeson and Höst, 2009; Wohlin et al., 2012).

La evaluación de un artefacto requiere integrar también la perspectiva de los usuarios que lo aplican. Por este motivo se incluyen encuestas de percepción, que permiten relevar dimensiones no observables directamente en los entregables técnicos, como claridad conceptual, facilidad de aprendizaje, utilidad percibida y satisfacción general. Estas variables son fundamentales para estimar la adoptabilidad y sostenibilidad del método en contextos reales. Este enfoque se fundamenta en el *Method Evaluation Model* (Moody, 2003), un modelo teórico para validar métodos de diseño de sistemas de información y en lineamientos de evaluación de artefactos propuestos por (Gregor and Hevner, 2013; Venable et al., 2016a, 2016b). Para garantizar una evaluación integral, se definió un conjunto de cinco dimensiones que funcionan como marco analítico común:

- **Metodológica:** estructura, claridad y coherencia interna del método, incluyendo fases, artefactos y roles (March and Smith, 1995; Prat et al., 2015; Venable et al., 2016a).
- **Dominio y colaboración:** grado en que el método integra el conocimiento del dominio y la participación de los actores relevantes (Evans, 2004; Hevner et al., 2004).
- **Validez funcional:** capacidad del método para producir resultados correctos, completos y verificables frente a los requisitos (Prat et al., 2015; Venable et al., 2016a).
- **Escalabilidad y adaptabilidad:** aptitud del método para evolucionar sin perder coherencia ni generar deuda técnica, en línea con diseño resiliente y ciclos de cambio (Jamshidi et al., 2018).
- **Perceptual:** experiencia de los usuarios respecto de claridad, facilidad de uso y utilidad, complementando la evaluación técnica con factores humanos y organizacionales (Moody, 2003)

Estas dimensiones constituyen un marco de evaluación consistente, fundamentado y alineado con los objetivos de la tesis, permitiendo valorar la propuesta BDMA de manera integral.

8.2. Evaluación de la aplicación del método BDMA

La estrategia de evaluación se centra en la *utilidad práctica* del método en contexto real o simulado, alineada con el criterio de *utilidad* (March and Smith, 1995) se orienta a analizar los efectos concretos de la aplicación del marco BDMA en un escenario de diseño de microservicios, en comparación con enfoques no sistemáticos. En este contexto, el foco se traslada a los resultados obtenidos y a la eficacia práctica del método en manos de los usuarios. Para ello, se propone un diseño experimental de carácter descriptivo-comparativo en la que un grupo de estudiantes resuelve la identificación de microservicios de un escenario provisto y sin ninguna guía metodológica (solo basándose en su experiencia previa) y luego, aplicando íntegramente el marco BDMA. Este enfoque comparativo permite identificar diferencias atribuibles a la aplicación del método, bajo condiciones similares de contexto, desafío y disponibilidad de información. La evaluación contempla dos dimensiones principales:

- Calidad técnica del diseño de microservicios, mediante una rúbrica específica que analiza segmentación, cohesión, trazabilidad al dominio, validación funcional y definición de contratos.
- Percepción de los participantes, a través de una encuesta estructurada que indaga sobre claridad, utilidad, dificultad y nivel de satisfacción respecto del método utilizado.

Esta estrategia permite recabar datos cuantitativos y cualitativos para comparar el desempeño entre grupos y explorar el potencial del marco BDMA desde una perspectiva pragmática. Desde un enfoque epistemológico, esta instancia responde al principio de utilidad práctica del artefacto (Venable et al., 2016a) uno de los pilares de la investigación basada en diseño. La capacidad del método para producir soluciones de mayor calidad y facilitar el proceso de diseño en contextos reales constituye una evidencia clave de su valor como contribución científica y técnica

8.2.1. Objetivo

El objetivo de esta evaluación es analizar la efectividad del marco BDMA como metodología de identificación de microservicios, a partir de su aplicación en un entorno experimental controlado, y en comparación con enfoques no sistemáticos. A través de esta instancia se busca no solo validar empíricamente la utilidad práctica del método BDMA, sino también recoger evidencia que permita valorar sus contribuciones diferenciales en contextos de diseño ágil del tipo *greenfield*. Las cinco dimensiones se operacionalizan de la siguiente manera:

- **Metodológica:** Evalúa la fidelidad con la que los participantes implementaron las fases del método BDMA, incluyendo el uso de artefactos intermedios, la trazabilidad de las decisiones de diseño y la coherencia interna del proceso seguido.
- **Dominio y colaboración:** Analiza el grado de alineación entre los microservicios diseñados y los elementos del dominio, la incorporación de narrativas

funcionales, y el modo en que los participantes tradujeron necesidades del negocio en decisiones arquitectónicas.

- **Validez funcional:** Considera si la solución generada cubre adecuadamente los requerimientos funcionales, tanto en la versión inicial como tras la incorporación de un cambio de requisitos. Evalúa la relación entre escenarios BDD, contratos y estructura de servicios.
- **Escalabilidad y adaptabilidad:** Examina la capacidad del diseño para responder a modificaciones de manera coherente y sin pérdida de calidad arquitectónica, a través del análisis del impacto del cambio y la modularidad de la solución adaptada.
- **Perceptual:** Releva la experiencia subjetiva de los participantes en cuanto a la claridad, utilidad, dificultad y satisfacción con el uso del método, brindando indicios sobre su potencial de adopción en entornos reales o formativos.

8.2.2. Protocolo de la evaluación

El experimento se estructura bajo un diseño con el propósito de comparar los efectos del uso del marco BDMA frente a enfoques no sistemáticos en el diseño de microservicios. A diferencia de una evaluación centrada exclusivamente en los productos obtenidos, este protocolo incorpora también la observación del proceso de implementación metodológica, así como la adaptación ante cambios de requisitos.

Metodología

- Tipo: Intra-sujetos
- Orden: Fijo (P1→P2)
- P1: Arquitectura SIN BDMA
- P2: Arquitectura CON BDMA

Control Interno

- Mismo escenario de dominio en P1 y P2
- Mismos equipos de trabajo
- Entregables comparables (rúbricas)
- Cuestionario anónimo post-experiencia

La intervención se realiza en un contexto académico, con estudiantes de posgrado que abordan la resolución de un mismo desafío técnico bajo condiciones controladas.

Se conforman grupos con características similares que ejecutaran el experimento en dos etapas, a partir de una consigna común diseñada como un ejercicio tipo KATA que simula un escenario *greenfield* para lograr la identificación de los microservicios involucrados:

- **Etapla inicial:** abordará el problema sin una metodología explícita, usando sus propios criterios o enfoques profesionales.
- **Etapla experimental:** abordará el problema siguiendo una guía metodológica de BDMA

La recolección de datos se realizará mediante:

- Rúbricas y encuestas de percepción (ver anexo B).
- Análisis documental de artefactos producidos (mapas, escenarios, diseños).

- Registro de sesiones (en caso de observación o entregas en plataformas).

El análisis se basará en una triangulación metodológica, contrastando:

- Evaluación técnica de los entregables.
- Análisis cualitativo de la experiencia del usuario.
- Comparación entre el desempeño de los grupos ante el cambio de requisitos.

Este protocolo permite articular una mirada integral que contempla tanto la calidad del producto, como la fidelidad metodológica al marco BDMA y la experiencia subjetiva de quienes lo aplican. De este modo, se garantiza una evaluación coherente con el enfoque de investigación basada en diseño, triangulando evidencias que permiten valorar la efectividad, utilidad y aplicabilidad práctica del método propuesto.

En resumen, el protocolo debe cumplir los siguientes pasos

1. Presentación del caso base (Kata).
2. Etapa Inicial: Resolución y presentación de resultados Iniciales sin usar ninguna guía metodológica
3. Etapa Final (experimental): Resolución y presentación de resultados utilizando el marco BDMA.

8.2.3. Instrumentos de evaluación

La evaluación de la aplicación del método BDMA en un entorno experimental se apoya en un enfoque mixto (cuantitativo-cualitativo) y en la triangulación de fuentes e instrumentos. La estrategia adoptada busca no solo valorar los resultados técnicos obtenidos por los participantes, sino también analizar cómo fue implementado el método, cómo se adaptaron las soluciones ante un cambio de requisitos y qué percepción tuvieron los usuarios sobre la experiencia metodológica.

Este enfoque permite captar evidencia sobre los efectos de aplicar BDMA en términos de resultados técnicos, fidelidad metodológica, alineación con el dominio y percepción de los participantes. Desde la perspectiva cuantitativa, se utilizaron la rúbrica y encuesta detallados en el anexo B

8.2.4. Amenazas a la validez

La validación preliminar del marco presenta limitaciones que pueden afectar la generalización de los resultados.

- Validez interna: la comparación se realizó en un entorno académico controlado, lo que reduce el impacto de variables externas, pero puede introducir sesgos de aprendizaje o motivación.
- Validez externa: la muestra estuvo compuesta por 28 estudiantes de posgrado, por lo que los resultados no necesariamente se extrapolan a equipos industriales distribuidos.
- Validez de constructo: las métricas aplicadas (cuestionario Likert) evalúan percepción y comprensión, pero no miden directamente la calidad arquitectónica del software.
- Validez de conclusión: el tamaño reducido de la muestra impide análisis estadísticos robustos.

Para mitigar estas amenazas, se empleó un diseño de comparación interna (antes /

después) y se documentaron todos los artefactos generados.

8.2.5. Resultados Obtenidos

8.2.5.1. Perfil de participantes

28 estudiantes de posgrados tecnológicos (maestría y doctorado). Perfiles profesionales diversos; mayoría de la industria TI. Roles: análisis/diseño, arquitectura, desarrollo, gestión de proyectos y analista funcional.

8.2.5.2. Conocimiento Previo (Escala 0-5)

Área de Conocimiento	Media (n=28)
Métodos Ágiles (Scrum, Lean)	3.71
Patrones de Software	3.43
UML	3.32
Arquitecturas Monolíticas	3.03
Domain-Driven Design (DDD)	2.86
Behavior-Driven Development (BDD)	2.82
Diseño de Microservicios	2.79
TDD/ATDD	2.54
SOA	2.50
Implementación de Microservicios	2.36

Tabla 21. Conocimientos previos de los participantes de la evaluación inicial de BDMA

8.2.5.3. Resultados Cuantitativos

Medias altas y consistentes con baja dispersión (4.46-4.75); amplio consenso sobre beneficios de BDMA.

Pregunta	Enunciado	Media	Desv. Est.	Min-Max
Q1	El enfoque sistemático me ayudó a entender mejor el dominio	4.75	0.44	4-5
Q2	Los escenarios BDD me permitieron razonar mejor sobre comportamiento	4.68	0.47	4-5
Q3	La guía paso a paso facilitó las decisiones arquitectónicas	4.57	0.69	3-5
Q4	Pude identificar mejor los microservicios con el enfoque estructurado	4.46	0.74	3-5
Q5	Esta forma de trabajar mejora la mantenibilidad de la solución	4.50	0.63	3-5

Q6	La comparación me ayudó a reflexionar sobre diseño arquitectónico	4.57	0.50	4-5
-----------	---	-------------	------	-----

Tabla 22. Resultados cuantitativos de la evaluación inicial de BDMA

8.2.5.4. Resultados Cualitativos

Ventajas identificadas

- Mayor orden y claridad en el proceso de diseño
- Mejor comprensión del dominio y razonamiento del comportamiento mediante BDD
- Reflexión y justificación más sólidas de decisiones arquitectónicas
- Trazabilidad completa desde requisitos hasta implementación
- Documentación estructurada con ADRs y artefactos versionados

Dificultades y mejoras sugeridas

- Curva de aprendizaje inicial (escenarios BDD y flujo por fases)
- Carga cognitiva por comparar enfoques en paralelo
- Más ejemplos de escenarios BDD
- Más práctica en derivación de contratos e interfaz

8.2.5.5. Análisis de los datos y conclusiones

A partir de los resultados del experimento inicial y de los vacíos metodológicos identificados en la literatura, se ejecutó el experimento con el propósito de evaluar preliminarmente el aporte de BDMA en un entorno controlado. Mientras que un estudio anterior (Battaglia, 2025) permitió observar las dificultades de diseño cuando los equipos carecen de una guía estructurada, este experimento exploratorio (Battaglia et al., 2025) incorpora explícitamente el marco BDMA para analizar su impacto en la comprensión del dominio, la identificación de microservicios y la fundamentación de decisiones arquitectónicas. Su objetivo es obtener evidencia temprana sobre la utilidad del método propuesto, identificar fortalezas y posibles limitaciones, y avanzar en la validación empírica del artefacto conforme al enfoque de *Design Science Research*. Los hallazgos derivados de este estudio complementan el análisis conceptual previo y contribuyen a consolidar la formulación del problema identificado previamente.

Con el objetivo de observar empíricamente las dificultades en la identificación de microservicios y evaluar preliminarmente el aporte de BDMA, se llevó a cabo un experimento pedagógico con 28 estudiantes de posgrado de la materia Diseño y Arquitectura de Software. A todos se les planteó el diseño de una plataforma de gestión de publicaciones científicas. Los participantes trabajaron en equipos y elaboraron dos diseños arquitectónicos para el mismo problema: primero, una solución sin un método estructurado (aplicando solo conocimientos generales para delimitar servicios y diseñar entidades) y luego una solución siguiendo las fases de BDMA. Tras completar ambas partes, se recogió feedback mediante un cuestionario Likert y preguntas abiertas. Este diseño intra-sujetos (misma persona antes y después) es válido para exploración, aunque se reconoce que pudieron aprender del primer intento. Los resultados, sin embargo, muestran diferencias significativas a favor de BDMA, lo que sugiere que los beneficios superaron la posible ventaja de conocimiento del dominio: en promedio,

BDMA obtuvo 4.5/5 en comprensión de dominio, identificación de servicios y mantenibilidad percibida, contra valores mucho más bajos sin BDMA (diferencias observables al comparar las dos entregas). Cualitativamente, los estudiantes indicaron que “la parte 2 con BDMA les permitió comprender mejor el dominio, razonar sobre el comportamiento esperado mediante escenarios BDD, contar con una guía paso a paso para fundamentar decisiones, e identificar microservicios con mayor precisión”. Muchos destacaron la mayor claridad y orden que aportó BDMA al proceso de diseño, en contraste con la incertidumbre de abordar el problema sin dicha guía. Como contrapartida, se mencionó la curva de aprendizaje inicial de BDMA como una dificultad menor, si bien todos los equipos lograron aplicar el marco en su segunda iteración. En síntesis, este experimento preliminar evidencia la brecha inicial (diseños ad-hoc poco alineados) y muestra el potencial de BDMA para mejorar la calidad y trazabilidad del diseño incluso en equipos noveles.

En resumen, los resultados evidencian que BDMA proporciona beneficios tangibles en el diseño de arquitecturas de microservicios, especialmente en la comprensión del dominio, razonamiento sobre comportamiento y toma de decisiones arquitectónicas. El enfoque sistemático de 5 fases reduce la complejidad cognitiva y mejora la calidad perceptual del diseño, validando su aplicabilidad tanto en contextos académicos como su potencial para entornos profesionales.

Capítulo 9. Conclusión y trabajos futuros

9.1. Conclusión

La presente tesis desarrolló y validó el marco *Behavior-Driven Microservice Architecture* (BDMA) como respuesta a la problemática identificada en el diseño de microservicios en entornos ágiles *greenfield*. A modo de conclusión, se sintetizan las contribuciones clave y resultados obtenidos:

- Integración innovadora de enfoques: BDMA constituye un método original que integra DDD con BDD dentro de un proceso iterativo guiado por principios de arquitectura evolutiva. Esta articulación interdisciplinaria (uno de los aportes centrales) no había sido abordada previamente de forma sistemática en la literatura. La tesis demuestra que es posible combinar exitosamente prácticas de descubrimiento funcional (BDD) con modelado estratégico (DDD) para lograr un diseño de microservicios más alineado con el dominio de negocio.
- Mecanismos de trazabilidad y alineación continua: a lo largo del trabajo se evidenció que BDMA mejora la trazabilidad entre requisitos del negocio y decisiones de arquitectura. Cada escenario BDD se transforma en artefactos arquitectónicos (mapas de contexto, contratos de API, registros de decisión), garantizando un rastro explícito desde las necesidades funcionales hasta los servicios implementados. Esta contribución se reflejó en las evaluaciones: el marco permitió una coherencia semántica mayor entre servicios y aumentó la capacidad de adaptación ante cambios de requisitos. En otras palabras, BDMA impacta positivamente la calidad de la arquitectura al mantenerla alineada con el comportamiento esperado del sistema en todo momento.
- Validación y aplicabilidad práctica: la construcción teórica de BDMA fue complementada con una validación exploratoria mediante un caso de estudio. Los resultados preliminares mostraron que equipos ágiles pueden adoptar el marco sin una curva de aprendizaje prohibitiva, y que BDMA efectivamente facilita la identificación iterativa de *bounded contexts* y microservicios. Se observaron mejoras en indicadores como la comprensión compartida del dominio en el equipo, la detección temprana de inconsistencias y la reducción de deuda técnica en fases iniciales. Si bien la evaluación fue acotada, ofrece evidencia inicial del impacto práctico del marco en situaciones controladas, asentando bases para futuras implementaciones a mayor escala.

En suma, la tesis logró su objetivo principal: diseñar y validar un marco metodológico replicable que llena un vacío en la ingeniería de software actual. BDMA aporta una guía estructurada, apoyada en fundamentos rigurosos y verificada en escenario experimental, para que equipos de desarrollo puedan descubrir, diseñar y evolucionar arquitecturas de microservicios basadas en comportamientos de negocio. Las conclusiones respaldan la idea de que una mejor integración entre prácticas de requisitos (BDD) y diseño estratégico (DDD) sí redundan en arquitecturas más adaptables y coherentes con el dominio, especialmente en entornos ágiles de alta incertidumbre.

A pesar de los logros alcanzados, es importante reconocer aquellos aspectos de esta investigación que quedaron fuera del alcance, las cuales abren oportunidades para mejoras y trabajo futuro:

- **Alcance de la validación:** La evaluación de BDMA se realizó a través de un estudio exploratorio limitado (un caso de uso principal y feedback de participantes en entorno controlado). Si bien los resultados fueron positivos, aún no se ha comprobado el marco en proyectos industriales de larga duración. Esto significa que carecemos de datos sobre el rendimiento de BDMA en escenarios de alta complejidad, con múltiples equipos o durante periodos extensos de evolución de un producto. La generalización de los hallazgos, por lo tanto, está acotada y requiere cautela.
- **Requisitos de adopción:** BDMA propone un conjunto de actividades y artefactos que suponen cierto nivel de madurez en los equipos de desarrollo. Por ejemplo, demanda fuerte involucramiento del experto de negocio en varias fases y disciplina en el mantenimiento de artefactos como los ADR. Esto podría presentar desafíos en organizaciones con cultura menos colaborativa o con poca experiencia en DDD/BDD. La tesis identificó que la curva de aprendizaje y el cambio cultural asociado son factores para considerar: en contextos distintos, la adopción de BDMA podría requerir entrenamiento adicional o adaptaciones. En resumen, el marco no es una “solución *plug-and-play*” universal, sino que debe implementarse con apoyo gerencial y posiblemente ajustes según el entorno.
- **Automatización y herramientas de soporte:** Si bien BDMA contempla el uso de técnicas automáticas —por ejemplo, modelos de lenguaje (LLM) para asistir la delimitación de contextos— gran parte del proceso continúa realizándose de forma manual o apoyado en herramientas genéricas. Actualmente no existe un software específico que orqueste de manera integrada todas las fases del marco, lo que introduce variabilidad en su aplicación y depende en mayor medida de la pericia del equipo para vincular correctamente las salidas y entradas entre fases. Esta decisión fue deliberada, priorizando principios metodológicos sobre herramientas concretas; no obstante, constituye un área clara de mejora. Una línea natural de evolución consiste en incorporar técnicas de procesamiento del lenguaje natural (NLP) y métodos de *clusterización* asistida para automatizar parcialmente la clasificación funcional de escenarios y la identificación inicial de elementos del dominio. Trabajos recientes como (Bajaj et al., 2022, 2024) evidencian el potencial de estos enfoques en contextos *greenfield*. Su integración permitiría avanzar hacia una versión “híbrida” de BDMA que, manteniendo la validación humana y la trazabilidad funcional, complementa las fases tempranas del marco con sugerencias algorítmicas que mejoren la consistencia, reproducibilidad y escalabilidad del proceso.

A pesar de estas limitaciones, ninguna invalida los resultados obtenidos; más bien señalan consideraciones prácticas y científicas para extender la investigación. Reconocer estos límites permite delinear con mayor precisión las líneas futuras de trabajo, orientadas a robustecer y ampliar el alcance de BDMA.

9.2. Trabajos futuros

Derivado de las conclusiones y limitaciones, se proponen las siguientes líneas de trabajo futuro para continuar y profundizar los aportes de esta tesis:

- Evaluación en entornos industriales y proyectos a gran escala: Como paso inmediato, resulta fundamental aplicar BDMA en casos de estudio más amplios y reales, por ejemplo, en colaboración con empresas de desarrollo de software. Un estudio longitudinal en un entorno productivo permitiría medir el impacto del marco en condiciones de alta complejidad (múltiples equipos, dominio en evolución constante, presión de entregas, etc.). Se podrían recoger métricas cuantitativas (tiempo de entrega por microservicio, incidencias de reprocesos, nivel de cohesión/acoplamiento logrado) para validar empíricamente los beneficios observados de manera cualitativa. Esta línea fortalecería la evidencia sobre la utilidad y escalabilidad de BDMA en la industria.
- Desarrollo de herramientas y automatización del marco: Para facilitar la adopción, un trabajo futuro valioso es construir herramientas software que den soporte a BDMA. Esto podría incluir extensiones de gestión de requisitos que conviertan automáticamente escenarios BDD en artefactos (por ejemplo, generación asistida de mapas de contexto a partir de narrativa, o software que aplique *clustering* semántico sobre los escenarios y sugiera *bounded contexts* candidatos). Incluso es factible explorar integración de inteligencia artificial para procesar descripciones de escenarios en lenguaje natural y proponer agrupamientos iniciales o nombres de contextos. Al dotar a BDMA de herramientas especializadas, se reduciría la carga manual y se mejorarían la reproducibilidad y consistencia del proceso. Este esfuerzo también habilitaría a equipos menos experimentados a seguir el marco con mayor confianza.
- Adaptación de BDMA a otros contextos y extensiones metodológicas: Si bien BDMA se concibió para entornos *greenfield*, surge la inquietud de evaluar su aplicabilidad en contextos *brownfield* o con equipos distribuidos geográficamente. Un posible trabajo futuro es ajustar las fases del marco para proyectos donde ya existe una base de código (¿pueden los escenarios BDD aplicarse a funcionalidades existentes para refactorizar hacia microservicios?). Asimismo, podría investigarse la incorporación de otros enfoques complementarios: por ejemplo, integrar *Event Storming* de manera más profunda en la fase de descubrimiento, o combinar BDMA con principios de Continuous Delivery/DevOps para asegurar que la evolución arquitectónica se refleje inmediatamente en ciclos de despliegue. Estas adaptaciones/extensiones probarían la flexibilidad del marco y podrían generar versiones especializadas de BDMA para distintos tipos de proyectos.
- Consideración de requisitos no-funcionales: La tesis se centró fuertemente en requisitos funcionales (escenarios BDD) para guiar la arquitectura. Esto cubre la alineación funcional, pero las arquitecturas de microservicios también están fuertemente influidas por requisitos no-funcionales (rendimiento, seguridad, consistencia de datos, etc.). Un área para reforzar es cómo incorporar esos aspectos en BDMA. Se espera incluir RNF en futuras revisiones del marco

- Formalización y refinamiento de artefactos intermedios: Otra línea interesante es trabajar en la estandarización de los artefactos que produce BDMA (mapas de contexto, ADR, tablas de trazabilidad, etc.). Futuras investigaciones podrían proponer metamodelos o perfiles UML para representar dichos artefactos de forma unificada, facilitando su comprensión y comparación. También se podría profundizar en métricas de calidad asociadas a estos artefactos (por ej., grado de cohesión semántica de un *bounded context* derivado con BDMA vs. uno obtenido por métodos tradicionales). Esto contribuiría a dar más rigor al método y a enlazarlo con marcos de medición de arquitectura de software.
- Automatización parcial de las fases tempranas del marco BDMA: particularmente, en el agrupamiento semántico de escenarios BDD y la detección inicial de elementos de dominio. En esta tesis, dichas actividades se realizan de manera colaborativa, pero pueden beneficiarse de técnicas de NLP, embeddings y modelos Transformer como los utilizados. La incorporación de algoritmos de clusterización asistida podría fortalecer la consistencia del proceso, reducir la variabilidad entre equipos y proporcionar sugerencias iniciales para las fases de análisis funcional y delimitación de *bounded contexts*. Este avance permitiría explorar un BDMA híbrido, manteniendo la validación humana y la trazabilidad funcional, pero potenciando la eficiencia y escalabilidad del método en dominios complejos.

En conclusión, BDMA abre un camino fértil de investigación en el ámbito del diseño de arquitecturas de microservicios guiadas por el comportamiento. Las conclusiones de la tesis confirman la validez y el potencial de este marco en entornos ágiles *greenfield*, al tiempo que los trabajos futuros delineados buscan amplificar su impacto, superar sus limitaciones actuales e inspirar aplicaciones prácticas innovadoras. Se espera que la comunidad académica e industrial tome estos resultados como base para continuar explorando la convergencia de métodos de diseño de software centrados en el dominio y el comportamiento, avanzando así hacia arquitecturas más adaptativas, trazables y alineadas con las necesidades reales del negocio.

Anexo A. Escenarios BDD

Utilizados en el capítulo 7

A continuación, detallamos la lista completa de los escenarios BDD utilizados en el caso de estudio del capítulo 7

Feature: Registro y autenticación de investigadores

Como Investigador

Quiero registrarme y autenticarme

Para gestionar mis envíos y revisiones

@registro-exitoso

Scenario: Registro exitoso con ORCID y verificación de email

Given existe el proveedor de identidad "ORCID"

And no existe una cuenta previa con email "ana@uni.edu"

When el Investigador se registra con ORCID válido y email "ana@uni.edu"

And confirma el email recibido

Then la cuenta "ana@uni.edu" queda en estado "Activa"

And el perfil se crea con ORCID vinculado

@autenticacion-ok

Scenario: Autenticación exitosa con credenciales válidas

Given existe la cuenta "ana@uni.edu" en estado "Activa" con password "s3cret!"

When intenta autenticarse con email "ana@uni.edu" y password "s3cret!"

Then el acceso es concedido

And se registra auditoría de inicio de sesión

@autenticacion-falla

Scenario: Autenticación rechazada por contraseña incorrecta

Given existe la cuenta "ana@uni.edu" en estado "Activa" con password "s3cret!"

When intenta autenticarse con email "ana@uni.edu" y password "wrong"

Then el acceso es denegado por credenciales inválidas

@bloqueo

Scenario: Bloqueo por múltiples intentos fallidos

Given existe la cuenta "mike@uni.edu" en estado "Activa" con password "pass"

When intenta autenticarse 5 veces con password incorrecto

Then la cuenta "mike@uni.edu" queda en estado "Bloqueada"

And se notifica al usuario por email

Feature: Gestión de perfil de investigador

Como Investigador

Quiero mantener mi perfil con afiliación, áreas y métricas

Para facilitar asignaciones y reportes

@perfil-basico

Scenario: Completar afiliación y áreas de interés

Given el Investigador "Ana" tiene un perfil inicial sin afiliación ni áreas
When actualiza afiliación a "Universidad X" y áreas ["IA", "NLP"]
Then el perfil de "Ana" muestra afiliación "Universidad X" y áreas ["IA", "NLP"]

@sincronizacion-metricas

Scenario: Importar métricas desde ORCID/Google Scholar

Given el perfil "Ana" tiene ORCID vinculado
When sincroniza métricas de citaciones e índice-h desde "Google Scholar"
Then el perfil exhibe métricas actualizadas con fuente "Google Scholar" y fecha de actualización

Feature: Envío y versionado de artículos

Como Autor
Quiero enviar y versionar artículos
Para postular a revistas y conferencias respetando deadlines

@envio-evento

Scenario: Envío válido a evento antes del deadline

Given existe el Evento "IC-2026" con fecha límite de envío "2026-05-01"
And soy Autor con perfil completo
When envío el Artículo "A-101" a "IC-2026" en fecha "2026-04-20"
Then se registra el Envío "E-1" en estado "Enviado" con versión 1
And recibo comprobante de envío con sello de tiempo

@envio-revista

Scenario: Envío válido a revista con revisión doble ciego

Given existe la Revista "SJ-ML" con tipo de revisión "Doble ciego"
And soy Autor con perfil completo
When envío el Artículo "A-202" a "SJ-ML" en fecha "2026-03-10"
Then se registra el Envío "E-2" en estado "Enviado" con versión 1
And se verifica que el archivo no contenga datos identificatorios

@deadline

Scenario: Rechazo por envío fuera de fecha

Given existe el Evento "IC-2026" con fecha límite de envío "2026-05-01"
And soy Autor con perfil completo
When envío el Artículo "A-102" a "IC-2026" en fecha "2026-05-10"
Then el sistema rechaza el envío por deadline excedido con motivo "Fecha límite superada"

@versionado

Scenario: Nuevo versionado de un envío en curso

Given existe el Envío "E-1" del Artículo "A-101" en estado "Enviado" y versión 1
When adjunto una nueva versión con cambios menores
Then el Envío "E-1" queda en versión 2
And el historial conserva versiones [1, 2]

Feature: Registro de revisores y especialidades

Como Revisor
Quiero registrar mis especialidades y conflictos
Para recibir asignaciones apropiadas

@alta-revisor

Scenario: Registro de revisor con especialidades

Given no existe el Revisor "r1"
When registro al Revisor "r1" con especialidades ["IA", "NLP"]
Then el Revisor "r1" queda "Activo" con especialidades ["IA", "NLP"]

@conflictos

Scenario: Declaración de conflicto de interés

Given el Revisor "r1" está "Activo"
When declara conflicto con el Autor "Ana" por coautoría reciente
Then el conflicto queda registrado y bloquea asignaciones con "Ana" por 2 años

Feature: Asignación de revisores y manejo de ceguera

Como Editor
Quiero asignar revisores automáticamente o manualmente
Para iniciar la evaluación cumpliendo reglas y anonimato

@auto-assign

Scenario: Asignación automática según especialidad y conflictos

Given existe el Envío "E-1" con tema "NLP"
And candidatos Revisores ["r1:IA", "r2:SE", "r3:NLP", "r4:IA"]
And el Revisor "r2" tiene conflicto con el Autor de "E-1"
And las reglas editoriales exigen "3" revisores
When ejecuto la asignación automática para "E-1"
Then quedan asignados 3 revisores especialistas sin conflictos
And el Autor no ve las identidades de los revisores
And los Revisores no ven la identidad ni afiliación del Autor

@manual-assign

Scenario: Asignación manual con validación de conflictos

Given existe el Envío "E-1" pendiente de asignación
And el Editor selecciona "r2", "r3" y "r4"
When confirma la asignación manual
Then el sistema impide asignar "r2" por conflicto registrado
And propone candidatos alternativos compatibles

@ceguera-simple

Scenario: Aplicación de ceguera simple en revista configurada

Given la Revista "SJ-DS" tiene tipo de revisión "Ciego simple"
And existe el Envío "E-3" a "SJ-DS"
When asigno revisores para "E-3"
Then los Revisores no ven la identidad del Autor
And el Autor puede ver las identidades de los Revisores solo después de la decisión final

Feature: Ciclo de revisión por pares

Como Revisor
Quiero aceptar/declinar invitaciones y cargar evaluaciones
Para contribuir a la decisión editorial

@acceptacion

Scenario: Revisor acepta invitación

Given el Revisor "r3" fue invitado a revisar el Envío "E-1"

When "r3" acepta la invitación dentro de 3 días
Then la asignación de "r3" pasa a estado "Aceptada"
And se habilita el formulario de revisión

@declinacion

Scenario: Revisor declina por conflicto sobrevenido

Given el Revisor "r3" fue invitado a revisar el Envío "E-1"
When "r3" declara conflicto y declina la invitación
Then su asignación pasa a "Declinada"
And se notifica al Editor para reasignación

@entrega

Scenario: Entrega de revisión con recomendación

Given el Revisor "r3" tiene asignación "Aceptada" sobre "E-1"
When envía su revisión con recomendación "Mayor revisión"
Then la revisión queda registrada con sello de tiempo y versión
And el historial del Envío "E-1" muestra la revisión de "r3"

@historial

Scenario: Consulta del historial completo de revisiones por artículo

Given existen revisiones de "r1", "r3" y "r4" para el Envío "E-1"
When consulto el historial de revisiones de "E-1"
Then veo todas las revisiones con autores anónimos, fechas y recomendaciones

Feature: Decisión editorial y comunicaciones

Como Editor
Quiero emitir decisiones y cartas
Para cerrar el proceso conforme a reglas

@decision

Scenario: Emisión de decisión basada en reglas y quorum

Given el Envío "E-1" tiene 3 revisiones con recomendaciones ["Aceptación", "Mayor revisión", "Aceptación"]
And las reglas exigen minRevisiones=3, quorum="2 de 3", umbral=">=Aceptación"
When calculo la decisión editorial para "E-1"
Then la decisión es "Aceptado condicional"
And se genera carta de decisión con condiciones y plantilla aprobada

@acceptacion-final

Scenario: Emisión de certificado tras aceptación final

Given el Envío "E-1" está en estado "Aceptado"
And el Autor subió la versión final y transfirió copyright
When cierro el expediente editorial
Then se emite certificado de aceptación para el Autor
And se programa la publicación con fecha estimada

Feature: Gestión de métricas e informes

Como Analista
Quiero generar métricas y exportarlas
Para seguimiento por autor, artículo, evento y revista

@metricas-autor

Scenario: Métricas por autor y evento

Given existen envíos y decisiones históricas
When genero métricas para el Autor "Ana" en el Evento "IC-2026"
Then obtengo métricas {envios:2, aceptados:1, tasaAceptacion:"50%"} con periodo consultado

@metricas-revista

Scenario: Métricas por revista

Given existen decisiones en la Revista "SJ-ML" durante 2026
When genero el informe de tasa de aceptación mensual para "SJ-ML"
Then obtengo una serie temporal con aceptación, tiempos de revisión y desviación estándar

@export

Scenario Outline: Exportación de reportes a formatos estándar

Given existe un reporte consolidado "R-1"
When exporto el reporte "R-1" en formato "<formato>"
Then descargo un archivo con extensión "<extension>" y esquema válido

Examples:

formato	extension
CSV	.csv
JSON-LD	.jsonld
XML	.xml

@boletin

Scenario: Envío de notificación mensual personalizada

Given el Autor "Ana" está suscripto al resumen mensual
And existen métricas del último mes para "Ana"
When corre el job mensual
Then se envía un email a "Ana" con su resumen y enlaces a detalles

Feature: Integraciones externas y validación de metadatos

Como Sistema
Quiero sincronizar perfiles y validar metadatos
Para mantener calidad e interoperabilidad

@orcid

Scenario: Sincronización de datos de perfil desde ORCID

Given el perfil "Ana" tiene ORCID vinculado
And existen cambios en ORCID para nombre y afiliación
When sincronizo datos desde ORCID
Then el perfil se actualiza preservando cambios locales confirmados
And se registra la fuente y fecha de sincronización

@scopus-scholar

Scenario: Importación de publicaciones desde Scopus y Google Scholar

Given el perfil "Ana" tiene identificadores válidos en Scopus y Google Scholar
When importo sus publicaciones recientes
Then las nuevas publicaciones se agregan como referencias con DOI

@doi

Scenario: Validación de DOI y metadatos al registrar versión final

Given el Envío "E-1" está en versión final con DOI "10.1000/xyz123"
And los metadatos requeridos son {titulo, autores, afiliaciones, resumen, palabrasClave}
When valido el DOI y los metadatos
Then el DOI queda marcado "Válido" y los metadatos "Completo"
And el registro queda listo para indexación

Anexo B. Rúbrica y encuesta de percepción del experimento exploratorio

Dimensión	Criterio	Nivel Bajo (1)	Nivel Medio (2)	Nivel Alto (3)
Trazabilidad funcional	¿Se puede rastrear claramente cada funcionalidad desde los escenarios hasta la arquitectura?	No hay vínculo explícito	Vínculos parciales o informales	Trazabilidad clara y completa
Claridad de la arquitectura	¿La arquitectura es comprensible, coherente y bien justificada?	Ambigua o confusa	Parcialmente comprensible	Clara, bien comunicada y justificada
Modularidad y contextos	¿Los Bounded Contexts están bien delimitados?	Mal definidos	Delimitación parcial	Lógica y correctamente delimitados
Derivación de contratos	¿Los contratos/interfases surgen del análisis de los escenarios?	Poco claros o ausentes	Contratos presentes pero poco fundamentados	Derivación explícita desde BDD
Eventos y modelo de dominio	¿Los eventos y entidades representan fielmente el dominio?	Incompletos o irrelevantes	Representación parcial del dominio	Representación fiel, relevante y coherente

Justificación de decisiones	¿Se explican y fundamentan las decisiones arquitectónicas?	Sin justificación	Justificación superficial	Justificación sólida basada en el método
Mantenibilidad esperada	¿La arquitectura propuesta favorece el cambio y evolución futura?	No	Parcialmente	Claramente sí, con mecanismos explícitos

Tabla 23. Rúbrica y encuesta de percepción del experimento exploratorio.

La encuesta de percepción se conformó con los siguientes campos

- Formación de base
- Área de desarrollo profesional (Academia / Industria IT / Otros.....)
- Dominio de desarrollo profesional
- Tipo de actividad
 - Análisis y diseño de software
 - Arquitectura de software
 - Desarrollo de software
 - Testing
 - Gestión de proyectos
 - Otros...
- Cuanto (del 0 al 5) conoce y cuanta experiencia (NADA, MUY POCO, MEDIA, MUCHA) sobre los siguientes conceptos
 - UML y Modelización
 - Patrones de software (diseño, análisis, etc)
 - Métodos ágiles (Scrum / Lean / Crystal / etc)
 - Métodos ágiles (TDD / ATDD)
 - Métodos ágiles (BDD)
 - DDD
 - Arquitecturas Monolíticas (layered, clean, etc)
 - SOA
 - Microservicios (diseño)
 - Microservicios (implementación)
- El enfoque sistemático BDMA me ayudó a entender mejor el dominio (del 1 al 5)
- El uso de escenarios BDD me permitió razonar mejor sobre el comportamiento esperado. (del 1 al 5)
- La guía paso a paso facilitó la toma de decisiones arquitectónicas (del 1 al 5)
- Pude identificar mejor los microservicios usando el enfoque estructurado (del 1 al 5)

- Considero que esta forma de trabajar ayuda a mejorar la mantenibilidad de la solución. (del 1 al 5)
- ¿Qué ventajas encontraste al aplicar el enfoque BDMA? (abierta)
- ¿Qué dificultades o limitaciones experimentaste durante ese proceso? (abierta)
- ¿Qué cambiarías del enfoque propuesto? (abierta)
- ¿En qué aspectos te resultó más útil trabajar con escenarios BDD? (abierta)
- ¿Qué aprendizajes te llevás de esta experiencia para tu práctica profesional? (abierta)

Anexo C. Resumen comparativo de los trabajos relacionados con BDMA

A continuación, detallaremos un cuadro comparativo de cada trabajo relacionado con el marco BDMA. Estos datos sirvieron como insumo para un análisis crítico en el capítulo 5.

Partitioning microservices: A domain engineering approach

Criterio	(Josélyne et al., 2018)	BDMA
Entrada principal	Modelo de dominio funcional derivado de ingeniería de productos	Escenarios BDD redactados colaborativamente
Enfoque técnico	Análisis estructurado de variabilidad del dominio (ingeniería de líneas de productos)	Asistencia algorítmica al análisis semántico, con validación humana iterativa
Técnica de identificación	Análisis de características y modularización en función de SPL	Recomienda LLM
Participación del negocio	Baja: no contempla stakeholders en el proceso de partición	Alta: colaboración en Event Storming, lenguaje ubicuo y validación de escenarios
Iteratividad	Baja: enfoque planificado con diseño anticipado	Alta: diseño emergente y refactorizable
Validación del diseño	Limitada: no contempla validación funcional automatizada	Validación funcional ejecutable con BDD
Trazabilidad funcional	Parcial: no establece conexión clara entre requerimientos y servicios	Completa: desde escenarios funcionales hasta microservicios y sus interfaces
Lenguaje ubicuo	No contemplado	Central en toda la metodología
Adaptabilidad al cambio	Limitada: orientado a contextos estables o variantes conocidas	Alta, con soporte para refactorización semántica y validación continua
Entorno de aplicación	Dominios con alta reutilización (familias de productos)	Entornos ágiles <i>greenfield</i> con descubrimiento progresivo

GreenMicro: identifying microservices from use cases in greenfield development

Criterio	(Bajaj et al., 2022)	BDMA
----------	----------------------	------

Entrada principal	Casos de uso UML estructurados	Escenarios BDD redactados colaborativamente
Enfoque técnico	Automatización completa del diseño mediante algoritmos evolutivos	Asistencia algorítmica al análisis semántico, con validación humana iterativa
Técnica de identificación	Programación genética multiobjetivo	Agrupamiento semántico asistido (TF-IDF + HAC)
Participación del negocio	Nula o mínima	Alta, durante todo el proceso
Iteratividad	Baja: enfoque de diseño inicial cerrado	Alta: diseño emergente y refactorizable
Validación del diseño	Evaluación automatizada por métricas técnicas	Validación funcional ejecutable con BDD
Trazabilidad funcional	Parcial: no explícitamente vinculada a requisitos	Completa: desde escenarios hasta servicios
Lenguaje ubicuo	No contemplado	Central en toda la metodología
Adaptabilidad al cambio	Limitada	Alta, con soporte para refactorización semántica
Entorno de aplicación	Greenfield estructurado (alta especificación inicial)	Greenfield ágil, con descubrimiento progresivo

Microservices backlog: A genetic programming technique for identification and evaluation of microservices from user stories

Criterio	(Vera-Rivera et al., 2020)	BDMA
Entrada principal	Historias de usuario formales	Escenarios BDD colaborativos
Enfoque técnico	Automatización algorítmica por programación genética	Asistencia semántica + validación iterativa humana
Técnica de identificación	Generación y evaluación de particiones funcionales	Recomienda LLM
Participación del negocio	Nula o mínima	Alta: Event Storming, lenguaje ubicuo, validación
Iteratividad	Baja: proceso en un solo paso	Alta: diseño emergente, validación y refactorización continua
Validación del diseño	Métricas de calidad arquitectónica	Pruebas funcionales ejecutables (BDD)

Trazabilidad funcional	Parcial: no explícitamente mantenida	Completa: desde escenarios hasta servicios y contratos
Lenguaje ubicuo	No contemplado	Central en la metodología
Adaptabilidad al cambio	Limitada	Alta, con soporte para evolución semántica y estructural
Entorno de aplicación	Requisitos funcionales definidos (user stories)	Entornos ágiles con descubrimiento progresivo

Designing microservice-based applications by using a domain-driven design approach

Criterio	(Hippchen et al., 2017)	BDMA
Entrada principal	Escenarios funcionales (BDD) y conceptos DDD	Escenarios BDD redactados colaborativamente
Enfoque técnico	Integración conceptual de BDD y DDD	Proceso operativo con fases, artefactos, roles y validación
Técnica de identificación	No definida: enfoque propositivo	Agrupamiento semántico + revisión colaborativa
Participación del negocio	Implícita, no metodológicamente estructurada	Alta: Event Storming, lenguaje ubicuo, validación
Iteratividad	Enunciada, pero sin mecanismos concretos	Alta: diseño emergente, validación y refactorización continua
Validación del diseño	No especificada	Validación funcional ejecutable con BDD
Trazabilidad funcional	Sugerida, pero no operacionalizada	Completa: desde escenarios hasta servicios y contratos
Lenguaje ubicuo	Mencionado como concepto	Central en toda la metodología
Adaptabilidad al cambio	Limitada: no definida en la propuesta	Alta, con soporte para evolución semántica y estructural
Entorno de aplicación	Visión para entornos ágiles	Metodología concreta para entornos ágiles <i>greenfield</i>

A guide for microservices in greenfield projects

Criterio	(Cardoso, 2021)	BDMA
-----------------	-----------------	-------------

Entrada principal	Recomendaciones generales, visión del negocio, y principios prácticos	Escenarios BDD redactados colaborativamente y términos del dominio
Enfoque técnico	Guía heurística estructurada en dimensiones clave (visión, diseño, despliegue)	Metodología iterativa con fases, roles, artefactos, entradas y salidas
Técnica de identificación	No definida formalmente; sugiere Event Storming como práctica	LLM Recomendado
Participación del negocio	Recomendada pero no sistematizada	Alta, desde la Fase 1: Event Storming, lenguaje ubicuo, validación colaborativa
Iteratividad	Sugerida en la filosofía, no instrumentalizada	Alta: validación continua y refactorización integrada
Validación del diseño	No contemplada formalmente	Validación funcional ejecutable mediante escenarios BDD
Trazabilidad funcional	Implícita en la guía, per	

Towards a UML profile for domain-driven design of microservice architectures

Criterio	(Rademacher et al., 2018)	BDMA
Entrada principal	Modelado de conceptos DDD en UML	Escenarios funcionales (BDD) y lenguaje ubicuo
Enfoque técnico	Formalización estructural de DDD (profile UML)	Metodología operativa, iterativa y semántica
Técnica de identificación	Modelado previo de bounded contexts y agregados	Agrupamiento semántico + validación iterativa funcional
Participación del negocio	Nula: enfoque técnico para arquitectos	Alta: Event Storming, narrativa funcional, validación
Iteratividad	Baja: diseño anticipado y estructural	Alta: evolución de diseño con base funcional
Validación del diseño	No contemplada	Validación funcional ejecutable (BDD)
Trazabilidad funcional	No contemplada formalmente	Completa: desde narrativa hasta servicios
Lenguaje ubicuo	Implícito en los modelos, no guiado por colaboración	Central, construido colaborativamente desde Fase 1
Adaptabilidad al cambio	Limitada a reestructuración de modelos UML	Soportada por validación y refactorización progresiva
Entorno de aplicación	Proyectos con enfoque model-driven y alto nivel técnico	Greenfield ágil con descubrimiento progresivo

Microservices identification strategies: A review focused on Model-Driven Engineering and Domain Driven Design approaches

Criterio	(Schmidt and Thiry, 2020)	BDMA
Entrada principal	Revisión sistemática de métodos existentes	Escenarios funcionales (BDD), términos del dominio
Enfoque técnico	Clasificación y análisis de enfoques (teórico)	Metodología operativa con fases y artefactos concretos
Técnica de identificación	N/A (identifica categorías: minería, MDE, heurística)	Agrupamiento semántico asistido + validación colaborativa
Participación del negocio	Identificada como carencia frecuente	Alta, estructurada desde el inicio del proceso
Iteratividad	Reclamada como necesidad en el estado del arte	Incorporada explícitamente en el diseño metodológico
Validación del diseño	Poca presencia en los métodos revisados	Validación funcional ejecutable con BDD
Trazabilidad funcional	Identificada como debilidad general	Asegurada mediante especificaciones y contratos
Lenguaje ubicuo	No mencionado explícitamente	Central y emergente en el proceso BDMA
Adaptabilidad al cambio	Demandada como futura línea de trabajo	Soportada mediante refactorización semántica y controlada

Evolution of microservices identification in monolith decomposition: A systematic review

Criterio	(Oumoussa and Saidi, 2024)	BDMA
Enfoque técnico	Revisión sistemática (SLR) sobre técnicas de extracción (<i>brownfield</i>)	Propuesta metodológica operativa para diseño (<i>greenfield</i>)
Entrada principal	Código fuente, modelos técnicos, artefactos existentes	Escenarios funcionales redactados en BDD, lenguaje ubicuo
Técnica de identificación	Clustering, análisis estático, transformaciones M2M	Agrupamiento semántico asistido (TF-IDF + HAC) con validación funcional
Participación del negocio	Ausente en la mayoría de enfoques analizados	Alta: Event Storming, narrativas colaborativas, revisión funcional
Iteratividad	Poca presencia en las técnicas relevadas	Alta: validación continua, rediseño progresivo

Validación del diseño	Basada en métricas técnicas (cohesión, acoplamiento, granularidad)	Basada en ejecución funcional (BDD) y feedback de negocio
Trazabilidad funcional	No contemplada explícitamente	Trazabilidad completa desde narrativa funcional hasta servicios
Lenguaje ubicuo	No abordado en los enfoques estudiados	Construcción y uso sistemático a lo largo del proceso
Adaptabilidad al cambio	No tratada o limitada a refactorización estructural	Incorporada mediante refactorización semántica y evolución iterativa
Entorno de aplicación	Migración de monolitos existentes (<i>brownfield</i>)	Descubrimiento de servicios en nuevos sistemas (<i>greenfield</i>)

Extraction of microservices from monolithic software architectures

Criterio	(Mazlami et al., 2017)	BDMA
Enfoque técnico	Extracción semántica automatizada (<i>brownfield</i>)	Diseño funcional y colaborativo (<i>greenfield</i>)
Entrada principal	Código fuente del sistema legado	Escenarios funcionales BDD redactados colaborativamente
Técnica de identificación	Clustering semántico automático con HAC	Recomienda LLM
Participación del negocio	Inexistente	Alta: Event Storming, lenguaje ubicuo, validación funcional
Iteratividad	No contemplada (proceso automático en un solo paso)	Alta: iteraciones sobre diseño, refinamiento y validación
Validación del diseño	No incorporada (basado en métrica técnica)	Validación funcional ejecutable con BDD
Trazabilidad funcional	No contemplada	Trazabilidad completa desde narrativa hasta servicio
Lenguaje ubicuo	No considerado	Central en la metodología
Adaptabilidad al cambio	Baja: depende del código existente	Alta: evolución semántica y funcional integrada
Entorno de aplicación	Migración de sistemas existentes (<i>brownfield</i>)	Diseño desde cero en entornos ágiles (<i>greenfield</i>)

Service Cutter: A systematic approach to service decomposition (Gysel et al., 2016)

Criterio	Gysel et al. (2016)	BDMA
Enfoque técnico	Segmentación algorítmica basada en atributos	Diseño funcional colaborativo y validado
Entrada principal	Artefactos técnicos (entidades, llamadas, restricciones)	Narrativas funcionales (BDD), lenguaje ubicuo
Técnica de identificación	Algoritmo multivariable de particionamiento configurable	Recomienda LLM
Participación del negocio	Nula: enfoque técnico para arquitectos	Alta: Event Storming, co-creación de escenarios, validación funcional
Iteratividad	No contemplada	Alta: fases con revisión continua y rediseño
Validación del diseño	Métricas estructurales (cohesión, acoplamiento)	Validación funcional ejecutable con BDD
Trazabilidad funcional	No contemplada	Trazabilidad completa desde requerimiento hasta contrato de servicio
Lenguaje ubicuo	No considerado	Construido explícitamente desde el inicio del proceso
Adaptabilidad al cambio	Moderada: recalcula agrupamientos con nuevos datos	Alta: evolución semántica guiada por validación funcional
Entorno de aplicación	Reingeniería de sistemas existentes (<i>brownfield</i>)	Diseño iterativo desde cero (<i>greenfield</i>)

Behavior-Driven Development of Microservice Applications

Criterio	(Tuglular et al., 2021)	BDMA
Enfoque técnico	Desarrollo guiado por comportamiento (BDD) aplicado a microservicios con énfasis en testing	Diseño de microservicios desde escenarios BDD, integrando DDD para el modelado de dominio
Entrada principal	Escenarios BDD escritos en Gherkin (nivel funcional)	Escenarios BDD + conocimiento del dominio (event storming, modelos contextuales)
Técnica de identificación	Derivación de servicios a partir de necesidades funcionales y escenarios de prueba	Identificación de bounded contexts y agregados mediante DDD estratégico y táctico

Participación del negocio	Implícita: colaboración en redacción de escenarios Gherkin	Alta: co-creación de escenarios, mapeo del dominio, validación iterativa con expertos del negocio
Iteratividad	Fuerte: ciclo de pruebas en tres niveles (E2E, componente, unidad)	Fuerte: evolución continua de contexto, contratos y escenarios validables
Validación del diseño	Ejecución progresiva de pruebas con herramientas como Cucumber y Jenkins	Validación semántica cruzada entre dominio, escenario y arquitectura, soportada por pruebas BDD
Trazabilidad funcional	Media: de escenarios a pruebas	Alta: desde requerimientos de negocio hasta arquitectura y servicios implementados
Lenguaje ubicuo	Parcial: expresado en Gherkin pero sin integración al diseño de dominio	Central: usado en modelado colaborativo, escenarios y contratos de servicio
Adaptabilidad al cambio	Moderada: buen soporte de pruebas, pero sin gestión explícita de evolución del dominio	Alta: soporta descubrimiento progresivo del dominio, refactorización de contexto y contratos
Entorno de aplicación	Proyectos de microservicios con enfoque en pruebas ágiles	Proyectos greenfield ágiles, con foco en arquitectura emergente y alineación semántica

Bibliografía

- Adžić, G. (2011). *Specification by Example*. Manning Publications Co., Greenwich, CT.
- Author. (2011). *ISO/IEC/IEEE 42010:2011 Systems and software engineering—Architecture description*.
- 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.
- Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice*. Addison-Wesley Professional.
- Battaglia, N. (2025). Identificación Ágil de Microservicios Utilizando DDD y BDD. *XXVIII Ibero-American Conference on Software Engineering (CIBSE 2025 Doctoral Symposium)*.
- Battaglia, N., Garcia, 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*, 1506–1510.
- Battaglia, N., Rossi, G., & Fernández., A. (2025). Behavior-Driven Microservice Architecture: Un Marco Metodológico para la Identificación Iterativa de Microservicios en Proyectos Ágiles Greenfield. *Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería Del Conocimiento IIISIC 2025*.
- Beck, K. (2003). *Test driven development: By example*. Addison-Wesley Professional.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., & others. (2001). *Manifesto for agile software development*.
- Beck, K., & Fowler, M. (2001). *Planning extreme programming*. Addison-Wesley Professional.
- Binamungu, L. P., Embury, S. M., & Konstantinou, N. (2018). Maintaining Behaviour-Driven Development Specifications: Challenges and Opportunities. *Proc. of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 175–184. <https://doi.org/10.1109/SANER.2018.8330207>
- Biswas, S. (2022). *BDD and Domain Driven Design: How to go hand in hand with BDD and DDD*. <https://medium.com/@soubhik.biswas/bdd-and-domain-driven-design-55d64bf1e776>
- Blair, S., Watt, R., & Cull, T. (2010). Responsibility-driven architecture. *IEEE Software*, 27(2), 26–32.
- Booch, G. (2006). The accidental architecture. *IEEE Software*, 23(3), 9–11.

- Brandolini, A. (2018). *Introducing EventStorming*. Leanpub.
- Cardoso, J. P. S. (2021). *A guide for microservices in greenfield projects (Tesis de master)*. Instituto Politecnico do Porto (Portugal).
- Chelimsky, D., Astels, D., Helmkamp Zach Dennis, & others. (2010). *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf.
- Cockburn, A. (2005). *The Hexagonal (Ports & Adapters) Architecture*. <https://alistair.cockburn.us/hexagonal-architecture>
- Dragičević, Z., & Bošnjak, S. (2019). Agile architecture in the digital era: Trends and practices. *Strategic Management*, 24(2), 12–33.
- 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. (2017). *Building Evolutionary Architectures*. O'Reilly Media.
- Fowler, M. (2003). Who needs an architect? *IEEE SOFTWARE*, 20(5), 11–13.
- Fowler, M. (2012a). Is Design Dead? In *Refactoring: Improving the Design of Existing Code* (2nd ed., pp. 75–90). Addison-Wesley.
- Fowler, M. (2012b). *Patterns of enterprise application architecture*. Addison-Wesley.
- Fowler, M. (2015a). *Microservice Premium*. <https://martinfowler.com/bliki/MicroservicePremium.html>
- Fowler, M. (2015b). *Microservice Trade-Offs*. <https://martinfowler.com/articles/microservice-trade-offs.html>
- Fowler, M. (2015c). *Monolith First*. <https://martinfowler.com/bliki/MonolithFirst.html>
- Fowler, M. (2015d). *YAGNI*. <https://martinfowler.com/bliki/Yagni.html>
- Giardino, C., Frattini, A., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., & Abrahamsson, P. (2016). Software development in startup companies: The greenfield startup model. *IEEE Transactions on Software Engineering*, 42(6), 585–604.
- Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 37(2), 337–355.
- Gysel, M., Korff, A., & Zimmermann, O. (2016). Service Cutter: A systematic approach to service decomposition. *European Conference on Service-Oriented and Cloud Computing*, 185–200. https://doi.org/10.1007/978-3-319-44482-6_12
- Hellesoy, A., Tooke, S., & Wynne, M. (2017). *The cucumber book: behaviour-driven development for testers and developers*. The Pragmatic Bookshelf.
- Hevner, A. R. (2007). A three-cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2), 87–92.

- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 75–105.
- Highsmith, J. (2002). *Agile Software Development Ecosystems*. Addison-Wesley.
- Hippchen, B., Giessler, P., Steinegger, R., Schneider, M., & Abeck, S. (2017). Designing microservice-based applications by using a domain-driven design approach. *International Journal on Advances in Software*, 10(3 & 4), 432–445.
- Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied software architecture*. Addison-Wesley.
- Irshad, M., Britto, R., & Petersen, K. (2021). Adapting Behavior Driven Development (BDD) for large-scale software systems. *Journal of Systems and Software*, 177, 110944.
- Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Newman, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3), 24–35.
- 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.
- Kapferer, S., & Zimmermann, O. (2020). Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling. *MODELSWARD*, 299–306.
- Kitchenham, B., & Charters, S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering. (Technical Report EBSE 2007-001)*. Keele University & Durham University.
- Kopp, O., Armbruster, A., & Zimmermann, O. (2018). Markdown Architectural Decision Records: Format and Tool Support. *Proceedings of the 10th ZEUS Workshop*, 55–62.
- Kruchten, P. (1995). Architectural Blueprints—The “4+1” View Model of Software Architecture. *IEEE Software*, 12(6), 42–50.
- Kruchten, P. (2010). Agility and Architecture: A Mismatch? *IEEE Software*, 27(2), 44–46.
- Lewis, J., & Fowler, M. (2014). *Microservices - a definition of this new architectural term*. <https://martinfowler.com/articles/microservices.html>
- March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4), 251–266.
- Martin, R. C. (2012). *Clean architecture: A craftsman’s guide to software structure and design*. Prentice Hall.
- Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. *2017 IEEE International Conference on Web Services (ICWS)*, 524–531. <https://doi.org/10.1109/ICWS.2017.59>
- Moody, D. L. (2003). The method evaluation model: a theoretical model for validating information systems design methods. *Proceedings of the 11th European Conference on Information Systems (ECIS 2003)*, 1327–1336.
- Newman, S. (2021). *Building microservices*. “O’Reilly Media, Inc.”

- Nord, R. L., & Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *IEEE Software*, 23(2), 47–53.
- North, D. (2006). Introducing BDD. *Better Software*, 12, 7.
- Oumoussa, I., & Saidi, R. (2024). Evolution of microservices identification in monolith decomposition: A systematic review. *IEEE Access*, 12, 23389–23405.
- Peffers, K., Rothenberger, M., Kuechler, B., & Tuunanen, T. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3), 45–77.
- Prat, N., Comyn-Wattiau, I., & Akoka, J. (2015). A taxonomy of evaluation methods for information systems artifacts. *Journal of Management Information Systems*, 32(3), 229–267.
- Rademacher, F., Sachweh, S., & Zündorf, A. (2018). Towards a UML profile for domain-driven design of microservice architectures. *Software Engineering and Formal Methods: SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers 15*, 230–245.
- Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.
- Richardson, C. (2018). *Microservices Patterns: With examples in Java*. Manning Publications.
- Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2), 131–164.
- Sangabriel-Alarcón, J., Ocharán-Hernández, J. O., Limón, X., & Cortés Verdín, K. (2025). Domain-Driven Design in Microservices-Based Systems Development: A Systematic Literature Review and Thematic Analysis. *Programming and Computer Software*, 50(8), 742–770. <https://doi.org/10.1134/S0361768824700749>
- 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.
- Shaw, M., & Garlan, D. (1996). *Software architecture: Perspectives on an emerging discipline* (Vol. 101). Prentice Hall.
- Skelton, M., & Pais, M. (2019). *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. Addison-Wesley.
- Smart, J. F., & Molak, J. (2023). *BDD in Action: Behavior-driven development for the whole software lifecycle*. Simon and Schuster.
- Solís, C., & Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development. *Proc. of the 37th Annual EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 383–387. <https://doi.org/10.1109/SEAA.2011.76>

- Su, R., Li, X., & Taibi, D. (2024). From Microservice to Monolith: A Multivocal Literature Review. *Electronics*, 13(8), 1452.
- Taibi, D., & Lenarduzzi, V. (2018). On the definition of microservice bad smells. *IEEE Software*, 35(3), 56–62.
- Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5), 22–32.
- Tuglular, T., Coşkun, D. E., Gülen, Ö., Okluoğlu, A., & Algan, K. (2021). Behavior-Driven Development of Microservice Applications. *International Journal of Computers*, 15, 130–137. <https://doi.org/10.46300/9108.2021.15.20>
- 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.
- Venable, J., Pries-Heje, J., & Baskerville, R. (2016a). A comprehensive framework for evaluation in design science research. *Design Science Research in Information Systems. Advances in Theory and Practice*, 423–438. <https://doi.org/https://doi.org/10.1007/s41870-024-01766-5>
- Venable, J., Pries-Heje, J., & Baskerville, R. (2016b). FEDS: a framework for evaluation in design science research. *European Journal of Information Systems*, 25(1), 77–89.
- 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. *Proceedings of the 17th IEEE International Conference on Services Computing (SCC 2020)*, 85–102.
- Vernon, V. (2013). *Implementing Domain-Driven Design*. Addison-Wesley Professional.
- Vernon, V. (2016). *Domain-driven design distilled*. Addison-Wesley Professional.
- Vural, H., & Koyuncu, M. (2021). Does domain-driven design lead to finding the optimal modularity of a microservice? *IEEE Access*, 9, 32721–32733.
- Waterman, M. G. (2014). *Reconciling agility and architecture: a theory of agile architecture (Doctoral dissertation)*. Open Access Te Herenga Waka-Victoria University of Wellington.
- Waterman, M., Noble, J., & Allan, G. (2015). How much up-front? A grounded theory of agile architecture. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 347–357.
- Wieringa, R. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in 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.

- Wynne, M. (2015). Introducing Example Mapping. *Cucumber Blog*.
<https://cucumber.io/blog/bdd/example-mapping-introduction/>
- Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domain-driven design for microservices: An evidence-based investigation. *IEEE Transactions on Software Engineering*, 50(6), 1425–1449.
- Zimmermann, O. (2017). Microservices tenets: Agile approach to service development and deployment. *Computer Science-Research and Development*, 32, 301–310.
- Zimmermann, O., Stocker, M., Lübke, D., Pautasso, C., & Zdun, U. (2019). *Introduction to microservice API patterns (MAP)*.
- 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.

Acrónimos y Abreviaturas

A

ACL – Anti-Corruption Layer

Patrón de DDD para proteger el modelo de un contexto frente a otro.

ADR – Architectural Decision Record

Registro liviano, versionado y justificable de decisiones arquitectónicas.

API – Application Programming Interface

Interfaz de programación que define operaciones y contratos de servicio.

ATDD – Acceptance Test-Driven Development

Pruebas de aceptación antes del desarrollo.

B

BDD – Behavior-Driven Development

Metodología para especificar comportamiento con escenarios Given–When–Then.

BC – Bounded Context

Unidad estratégica de DDD que agrupa conceptos coherentes del dominio.

BDMA – Behavior-Driven Microservice Architecture

Marco metodológico propuesto en esta tesis.

C

CSV – Comma-Separated Values

Formato tabular simple para exportación de datos.

D

DDD – Domain-Driven Design

Enfoque de diseño guiado por el dominio.

DOI – Digital Object Identifier

Identificador persistente de publicaciones científicas.

DSR – Design Science Research

Metodología de investigación basada en diseño.

E

EDA – Event-Driven Architecture

Arquitectura basada en eventos asincrónicos.

ETL – Extract, Transform, Load

Proceso para integrar datos, implícito en sincronizaciones externas (Scopus/Scholar).

F

FI – Fitness Function

Funciones que validan automáticamente atributos arquitectónicos.

G

Gherkin

DSL de BDD (Given–When–Then).

H

HAC – Hierarchical Agglomerative Clustering

Algoritmo de agrupamiento usado para clusterización semántica en BDMA.

HTTP – HyperText Transfer Protocol

Protocolo para servicios REST.

J

JSON – JavaScript Object Notation

Formato de datos ligero.

JSON-LD – JSON for Linked Data

Extensión de JSON para datos enlazados.

JWT – JSON Web Token

Token estándar para autenticación distribuida.

L

LLM – Large Language Model

Modelo de lenguaje de gran escala (utilizado para asistencia semántica en Fase 1).

M

MDE – Model-Driven Engineering

Ingeniería guiada por modelos.

MVP – Minimum Viable Product

Producto mínimo viable (referencia en desarrollo greenfield).

O

OHS – Open Host Service

Patrón de DDD que define una interfaz estable para consumidores externos.

P

PO – Product Owner

Rol ágil relevante en las sesiones colaborativas del marco BDMA.

REST – Representational State Transfer

Estilo arquitectónico para APIs sobre HTTP.

S

SOA – Service-Oriented Architecture

Arquitectura orientada a servicios tradicional.

SQL – Structured Query Language

Lenguaje estándar para bases de datos relacionales.

T

TDD – Test-Driven Development

Práctica de escribir pruebas antes del código.

TF-IDF – Term Frequency–Inverse Document Frequency

Métrica estadística para análisis semántico en Fase 1.

U

UAI – Universidad Abierta Interamericana

Institución del autor.

UML – Unified Modeling Language

Lenguaje de modelado.

URL – Uniform Resource Locator

Dirección de un recurso web.

V

XML – eXtensible Markup Language

Formato estructurado para intercambio de datos.