Identificación Ágil de Microservicios Utilizando DDD y BDD

Nicolás Battaglia[1]

[1]Centro de Altos Estudios en Tecnología Informática (CAETI). Facultad de Tecnología Informática – Universidad Abierta Interamericana – Buenos Aires – Argentina

Nicolas.battaglia@uai.edu.ar

Resumen *La adopción de microservicios mejora la mantenibilidad y escalabilidad frente a arquitecturas monolíticas, aunque su diseño inicial es desafiante. Esta investigación propone un marco ágil que integra Desarrollo Guiado por Comportamiento y Diseño Guiado por el Dominio para identificar microservicios y definir arquitecturas escalables. Un análisis de la literatura y un experimento evidencian que combinar diseño estratégico y prácticas ágiles reduce la deuda técnica y mejora la escalabilidad.*

Abstract *Adopting microservices improves maintainability and scalability compared to monolithic architectures, though initial design remains challenging. This research proposes an agile framework combining Behavior-Driven Development and Domain-Driven Design to identify microservices and define scalable architectures. A literature review and an experiment show that integrating strategic design with agile practices reduces technical debt and enhances scalability.*

1. Introducción

El crecimiento de las arquitecturas basadas en microservicios (MS) se debe a sus ventajas frente a las arquitecturas monolíticas, especialmente al aprovechar la computación en la nube. Los MS facilitan la mantenibilidad y escalabilidad mediante el desarrollo, prueba y despliegue independiente de cada servicio, mejorando la flexibilidad y reduciendo la complejidad. Su modularidad permite que equipos distribuidos trabajen de forma autónoma, acelerando el desarrollo. Además, su tolerancia a fallos asegura que la falla de un servicio no afecte al sistema completo. Integrados con prácticas de DevOps, optimizan recursos y reducen costos, posicionándose como una solución escalable y rentable (De Lauretis, 2019). Sin embargo, el diseño de una arquitectura basada en microservicios no es trivial. Si las "fronteras" entre los microservicios no son las correctas, se pueden generar "malos olores" que con el tiempo generan problemas en la evolución y el mantenimiento (Taibi & Lenarduzzi, 2018). Al tomarse decisiones arquitecturales tempranamente (y con poca información) corremos el riesgo de hacerlo de manera equivocada dado que las decisiones de diseño arquitectónico son (como se afirma en (Fowler, 2012)) aquellas que son más difíciles de modificar durante el ciclo de vida del software. Este problema se agrava cuando se utilizan enfoques ágiles de desarrollo (Kruchten, 2010) dado que los mismos rechazan la utilización de períodos largos de análisis y diseño (por ejemplo, arquitectural) previos al desarrollo. No es de extrañar entonces que muchas compañías hayan decidido dar marcha atrás en sus migraciones a microservicios y volver a centrar sus sistemas críticos en arquitecturas monolíticas (Su et al., 2024).

Por estas razones (entre otras), se discute si en un sistema basado en MS es preferible diseñar los MS tempranamente o debe comenzarse como en un monolito para dividirse (luego, si fuera necesario) en MS (Cardoso, 2021). En (Lewis & Fowler, 2014), como un argumento razonable, se indica que no se debería comenzar con MS sino con un monolito modular y, solo cuando (y sí) se convierte en un problema, descomponerlo en MS (usando técnicas existentes y documentadas). Este problema incluso afecta a desarrolladores más experimentados (Fowler, 2015a). Varios autores han propuesto el uso de Domain-Driven Design (DDD) (Evans, 2004) como una estrategia adecuada para el descubrimiento de MS. Su uso podría reducir los malos olores mencionados anteriormente (Vural & Koyuncu, 2021). DDD propone basar el proceso de construcción de software en una comprensión profunda del modelo de negocios (en colaboración con los expertos del dominio) y de esa manera "descubrir" las abstracciones adecuadas, algunas de las cuales nos pueden ayudar a encontrar las fronteras entre MS. DDD aplicado en el contexto de las arquitecturas ágiles (Waterman, 2014) permitiría reducir el riesgo de tomar decisiones inadecuadas tempranamente. Además, la combinación de DDD con prácticas ágiles consolidadas como Test-Driven Development (Beck, 2022) permitirían asegurar que el diseño emergente (Blair et al., 2010) se mantenga correcto. No obstante, se ha reportado (Zhong et al., 2024) que no existe un soporte pragmático (en términos, por ejemplo, de heurísticas, reglas, etc) para guiar el reconocimiento ágil de microservicios en sistemas complejos. Los equipos de desarrollo enfrentan problemas para mantener consistencia entre modelos de dominio y su implementación. Además, la dificultad que existe para que desarrolladores y técnicos adquieran el conocimiento específico del dominio, valorando a las metodologías ágiles que alientan la participación de stakeholders en el proceso de desarrollo desde las etapas más tempranas. Adicionalmente, la naturaleza técnica de TDD, basada en un nivel de abstracción mucho más bajo (código) que el atacado por DDD, hace que exista un gap entre los conceptos apropiados para razonar sobre MS y los necesarios para aplicar esta técnica. Es en este punto que proponemos, entre otras contribuciones, la utilización sistemática de BDD (North & others, 2006) como un complemento a DDD para descubrir, testear y soportar la evolución de MS.

El objetivo general del trabajo doctoral es proponer un enfoque metodológico que integre Behavior-Driven Development (BDD) y Domain-Driven Design (DDD) para la identificación, diseño e implementación de microservicios. Este enfoque se centrará en evaluar el impacto de nuevos requisitos en la arquitectura, promoviendo la adaptabilidad y evolución del sistema, y previniendo la aparición temprana de malos olores o facilitando su eliminación de manera ágil. Este objetivo general se puede desglosar en objetivos específicos que aborden tanto la identificación de microservicios como la mantenibilidad: 1. Analizar las mejores prácticas actuales de BDD y DDD en el contexto de microservicios, con énfasis en cómo estas metodologías pueden mejorar la mantenibilidad y la capacidad de adaptación a nuevos requisitos. 2. Desarrollar un marco metodológico que combine BDD y DDD para la identificación de microservicios, priorizando la cohesión entre servicios y la facilidad de adaptación a cambios en los requisitos del negocio. 3. Proponer un conjunto de heurísticas de diseño que permitan adaptar las prácticas propuestas por DDD al mundo ágil que ayude al descubrimiento incremental de MS. 4. Evaluar el impacto de nuevos requisitos en la arquitectura de microservicios, midiendo cómo el enfoque propuesto mejora la capacidad de mantener y modificar los servicios sin introducir riesgos, complejidades innecesarias ni malos olores.

5.Comparar los resultados obtenidos con enfoques tradicionales de diseño de microservicios, destacando mejoras en la flexibilidad y la mantenibilidad del sistema a lo largo del tiempo. Este artículo se organiza de la siguiente manera: en la sección 2 abordamos el marco teórico de los temas relevantes en estudio. En la sección 3, describimos aspectos metodológicos del desarrollo de la tesis, los avances obtenidos hasta el momento y los próximos pasos. Por último, introducimos la propuesta sobre la cual se sustentará la tesis.

2. Preguntas de Investigación

A continuación, describimos las preguntas de investigación que motivan nuestro trabajo: 1. ¿Cómo puede el uso de Behavior-Driven Development (BDD) mejorar la identificación temprana de límites contextuales en arquitecturas de microservicios? 2. ¿Qué tan efectivo es combinar DDD con BDD para garantizar la cohesión y granularidad óptimas en el diseño de microservicios? 3.¿En qué medida la integración de DDD y BDD reduce la deuda técnica y mejora la mantenibilidad en sistemas basados en microservicios? 4.¿Qué diferencias significativas se observan en términos de refactorización y escalabilidad entre equipos que utilizan DDD y BDD frente a aquellos que no siguen metodologías específicas?

3. Conocimiento Actual del Problema

Como se dijo previamente, determinar cuáles son los límites de los microservicios es crucial para un buen diseño arquitectónico (Fowler, 2015a; SAID et al., 2024). De esa manera evitamos redundancia y acoplamiento reduciendo el costo de mantenimiento del sistema (Taibi et al., 2017). Como se mencionó en la sección 1, DDD ayuda a identificar y extraer los MS de un dominio mediante la identificación de "*bounded contexts*" que pueden ser interpretados como MS.

La falta de heurísticas o guías para utilizar DDD para la detección temprana de MS en un enfoque ágil hacen que el éxito de este enfoque dependa exclusivamente de la experiencia del arquitecto (Kapferer & Zimmermann, 2020). Existe otro problema, no menor, que tiene que ver con la evolución de los requisitos y su impacto en el diseño de una arquitectura de MS. Si bien hay numerosos estudios que abordan el tema desde la perspectiva ágil, no identificamos en la literatura discusiones sólidas sobre las implicaciones de esta evolución y las consecuencias de tomar las decisiones de diseño en las etapas incorrectas. Nos referimos a entender cuando se viola el principio de "no lo vas a necesitar" (Fowler, 2015b) o sufrir la falacia del gran diseño por adelantado (Zimmermann, 2017). En este contexto, identificamos las arquitecturas ágiles como una conciliación entre la agilidad y la arquitectura (Waterman, 2014) permitiendo reducir el riesgo que genera dedicarle mucho tiempo a planificar y diseñar la arquitectura por adelantado. En los procesos ágiles, la fase inicial de planificación, conocida como "*sprint zero*" (Nord & Tomayko, 2006), define el tiempo necesario entre el inicio del proyecto y el desarrollo, tomando decisiones arquitectónicas clave que son difíciles de cambiar posteriormente. Este enfoque, denominado diseño "*upfront*", busca establecer una base arquitectónica sólida. Sin embargo, tras esta etapa, las decisiones arquitectónicas tienden a ser emergentes (Waterman, 2014), evolucionando con el desarrollo del software. Mientras una planificación prolongada puede retrasar la entrega de valor y reducir la agilidad, una arquitectura 100% emergente exige refactorizaciones continuas para mantener la coherencia, arriesgando que una arquitectura intencionada se vuelva

accidental (Booch, 2006). Por ello, el "*sprint zero*" debe equilibrar las decisiones iniciales y las que emerjan durante el desarrollo, aunque los métodos ágiles no definen claramente cuánto de la arquitectura debe decidirse al inicio. Por otro lado, Técnicas como *Test-Driven Development* (TDD) (Hellesoy et al., 2017) simplifican la refactorización del código en procesos ágiles, aunque suelen excluir a interesados como analistas y expertos del dominio. *Behavior-Driven Development* (BDD), en cambio, amplía el alcance al incluir a todos los involucrados en el proceso. Basado en prácticas ágiles como TDD [26] y *Domain-Driven Design* (DDD), BDD utiliza un lenguaje ubicuo propuesto por Evans (Evans, 2004) y adaptado por Dan North (North & others, 2006) para describir requisitos de forma clara y accesible. Este enfoque permite transformar escenarios definidos por analistas en pruebas de aceptación automatizadas, facilitando la comprensión entre técnicos y expertos del dominio (Tuglular et al., 2021). En entornos con requisitos cambiantes, BDD se presenta como una herramienta clave para construir características progresivamente y garantizar que el software cumpla con las expectativas del negocio (Smart & Molak, 2023).

3.1. Trabajos Relacionados

El desarrollo de aplicaciones basadas en microservicios (MS) utilizando Behavior-Driven Development (BDD) es un área poco explorada, a pesar de que BDD está ganando popularidad por su enfoque ágil y centrado en el dominio, facilitando el uso de Domain-Driven Design (DDD). Sin embargo, no existen mecanismos adecuados para identificar MS de manera temprana utilizando los principios ágiles de BDD a partir de escenarios descritos en lenguaje específico de dominio (Ma et al., 2018). Una búsqueda bibliográfica inicial reveló una escasez de estudios relacionados, lo que evidencia un vacío en la literatura y abre la oportunidad para nuevas propuestas.

Entre los trabajos relevantes, (Tuglular et al., 2021) propone un método de BDD para MS que comienza con pruebas a nivel de sistema y microservicio, integrando pruebas negativas tras cubrir las historias de usuario, validado con cinco microservicios. El artículo (Ma et al., 2018) sugiere usar algoritmos de procesamiento del lenguaje natural para recuperar MS en contextos específicos, entrenados con tokens de documentos en sintaxis Gherkin de BDD. En (Rahman & Gao, 2015) se presenta una arquitectura reutilizable para pruebas de aceptación automáticas en MS con BDD, abordando la reusabilidad, el desacoplamiento de pruebas del código fuente y la facilidad de auditoría, aunque identifica desafíos como mantenibilidad y testing de caja negra. Finalmente, (Lima et al., 2021) destaca que BDD permite a usuarios no técnicos escribir casos de prueba, facilitando la detección de fallos de negocio no identificados en pruebas unitarias de los servicios.

4. Estado del proyecto

Este trabajo adopta la metodología Design Science Research (DSR) como enfoque central, dado su enfoque en el desarrollo de artefactos innovadores y su validación rigurosa en entornos prácticos. Hevner et al (Hevner et al., 2004) destacan que la DSR se compone de tres ciclos principales: el ciclo de relevancia, el ciclo de rigor y el ciclo central de diseño. Estos ciclos permiten garantizar que el artefacto no solo sea innovador y técnicamente sólido, sino también ´util y aplicable en contextos reales. Para desarrollar el ciclo de relevancia, el primer punto que se trabajó fue en la realización de un mapeo sistemático sobre el descubrimiento de microservicios utilizando metodologías ágiles

(Battaglia et al., 2024) y, posteriormente, un experimento práctico con estudiantes de posgrado. 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: Pl: ¿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? y 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 la 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 asi 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 versen sobre la creación de arquitecturas basadas en MS desde su concepción ya que, en general, trabajan sobre la descomposición de monolitos previos.

Por otro lado, se realizó un experimento para comparar enfoques en la identificación y diseño de microservicios (MS). Participaron 26 estudiantes 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 y alineación entre objetivos técnicos y de negocio.

4.1. Próximos pasos

Para abordar los ciclos de diseño y evaluación, nos apoyamos en el análisis de la literatura y estudios realizados hasta el momento que nos permiten identificar que BDD puede utilizarse para describir y analizar requisitos, identificando objetivos de negocio y buscando características que ayuden a alcanzarlos asi como colaborar en la identificación y mantenimiento de MS. En base a lo anteriormente expuesto buscaremos, en primera medida, comprender que sucede actualmente en la industria en términos de uso de BDD y TDD para identificar MS y, a partir a los aportes de (Irshad et al., 2021; Smart & Molak, 2023), propondremos un marco inicial de trabajo iterativo para la identificación de MS utilizando DDD y BDD tanto para la incorporación de nuevas características como para el refinamiento de los MS existentes. Esta propuesta se basa en la definición estratégica del dominio con DDD, alcance y propósito, modelado funcional y definición de escenarios utilizando BDD/Gherkin. Por último, se realiza el diseño arquitectónico, modelado de datos y definición de interfaces. En términos concretos, buscamos combinar los beneficios de las arquitecturas ágiles con la naturaleza de DDD y BDD en uso de lenguajes específicos de dominio y la caracterización temprana de microservicios con límites apropiados.

5. Conclusiones

En este trabajo presentamos los aspectos más importantes de la investigación doctoral sobre enfoques ágiles basados en DDD y BDD para especificación de microservicios. La integración de métodos sistemáticos como Domain-Driven Design (DDD) con metodologías ágiles ofrece un enfoque efectivo para abordar la complejidad inherente al diseño de microservicios (MS). Este enfoque iterativo y evolutivo maximiza la capacidad de respuesta ante cambios, asegurando que las soluciones desarrolladas reflejen las necesidades reales del negocio. Además, la combinación de DDD con Behavior-Driven Development (BDD) no solo facilita la identificación y diseño de microservicios desde las primeras etapas del desarrollo, sino que también mejora la mantenibilidad y escalabilidad del sistema. Los resultados de nuestro experimento con estudiantes de posgrado demuestran que el uso de estas metodologías reduce la deuda técnica y mejora la capacidad de adaptación a nuevos requisitos. Sin embargo, persisten desafíos, como la necesidad de heurísticas y guías más pragmáticas para la detección temprana de microservicios en sistemas complejos. También es crucial seguir investigando cómo la

evolución de los requisitos impacta en el diseño arquitectónico y cómo mitigar los riesgos asociados a decisiones de diseño tomadas en etapas tempranas.

En resumen, nuestro trabajo propone un marco metodológico que integra DDD y BDD para la identificación, diseño e implementación de microservicios, promoviendo la adaptabilidad y evolución del sistema. Este enfoque no solo aborda los desafíos actuales, sino que también sienta las bases para futuras investigaciones y mejoras en el campo del desarrollo de software basado en microservicios

6. Referencias

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.

Beck, K. (2022). *Test driven development: By example*. Addison-Wesley Professional.

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.

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

De Lauretis, L. (2019). From monolithic architecture to microservices architecture. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 93–96.

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

Fowler, M. (2012). *Patterns of enterprise application architecture*. Addison-Wesley.

Fowler, M. (2015a). *Monolith First*. https://martinfowler.com/bliki/MonolithFirst.html

Fowler, M. (2015b). *YAGNI*. https://martinfowler.com/bliki/Yagni.html

Hellesoy, A., Tooke, S., & Wynne, M. (2017). *The cucumber book: behaviour-driven development for testers and developers*. The Pragmatic Bookshelf.

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

Irshad, M., Britto, R., & Petersen, K. (2021). Adapting Behavior Driven Development (BDD) for large-scale software systems. *Journal of Systems and Software*, *177*, 110944.

Kapferer, S., & Zimmermann, O. (2020). Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling. *MODELSWARD*, 299–306.

Kruchten, P. (2010). Software architecture and agile software development: a clash of two cultures? *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, 497–498.

Lewis, J., & Fowler, M. (2014). *Microservices - a definition of this new architectural term*. https://martinfowler.com/articles/microservices.html

Lima, H. R., Souza, K. C., de Paula, L. V., e Martins, L. M. C., Giozza, W. F., & de Sousa, R. T. (2021). Acceptance tests over microservices architecture using behaviour-driven development. *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6.

Ma, S.-P., Chuang, Y., Lan, C.-W., Chen, H.-M., Huang, C.-Y., & Li, C.-Y. (2018). Scenario-based microservice retrieval using Word2Vec. *2018 IEEE 15th International Conference on E-Business Engineering (ICEBE)*, 239–244.

Nord, R. L., & Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *IEEE Software*, *23*(2), 47–53.

North, D., & others. (2006). Introducing BDD. *Better Software*, *12*, 7.

Rahman, M., & Gao, J. (2015). A reusable automated acceptance testing architecture for microservices in behavior-driven development. *2015 IEEE Symposium on Service-Oriented System Engineering*, 321–325.

SAID, M. A. I. T., EZZATI, A., MIHI, S., & BELOUADDANE, L. (2024). Microservices adoption: An industrial inquiry into factors influencing decisions and implementation strategies. *International Journal of Computing and Digital Systems*, *15*(1), 1417–1432.

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

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.

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*. Open Access Te Herenga Waka-Victoria University of Wellington.

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*.

Zimmermann, O. (2017). Microservices tenets: Agile approach to service development and deployment. *Computer Science-Research and Development*, *32*, 301–310.