

## Actividad de aprendizaje #1.2

24-1-2025

# Metodologías de Desarrollo de Software



**CUCEI**

**Integrantes:**

**Ruiz Beraud Iker Paul**

**Palacios Palacios Leonardo Daniel**

METODOLOGIA	CARACTERISTICAS	VENTAJAS	DESVENTAJAS
Desarrollo Ágil	<ul style="list-style-type: none"> <li>• Priorización de entregas rápidas e iterativas.</li> <li>• Alta adaptabilidad a cambios durante el desarrollo.</li> <li>• Fuerte comunicación y colaboración entre equipos y con el cliente.</li> <li>• Organización basada en ciclos cortos denominados sprints o iteraciones.</li> <li>• Énfasis en la entrega de software funcional en lugar de documentación extensa.</li> </ul>	<ul style="list-style-type: none"> <li>• Mejora la satisfacción del cliente al recibir entregas frecuentes.</li> <li>• Permite responder rápidamente a cambios en requisitos.</li> <li>• Facilita la detección temprana de errores o problemas.</li> <li>• Fomenta una comunicación constante entre desarrolladores y partes interesadas.</li> <li>• Incrementa la motivación y el compromiso del equipo al mantenerlo involucrado y enfocado.</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser difícil de implementar en proyectos grandes o con equipos poco experimentados.</li> <li>• Exige un nivel alto de participación del cliente durante todo el proceso.</li> <li>• Es menos efectiva en proyectos donde los requisitos están completamente definidos desde el inicio.</li> <li>• La falta de documentación detallada puede ser un problema para el mantenimiento futuro.</li> </ul>
Scrum	<ul style="list-style-type: none"> <li>• Basado en iteraciones llamadas sprints, con una duración fija (generalmente de 2 a 4 semanas).</li> <li>• Roles definidos: Product Owner, Scrum Master y Equipo de Desarrollo.</li> <li>• Uso de artefactos como el Product Backlog, Sprint Backlog y el Incremento.</li> <li>• Rituales principales: Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective.</li> </ul>	<ul style="list-style-type: none"> <li>• Fomenta la colaboración y la transparencia entre los miembros del equipo.</li> <li>• Permite una respuesta rápida a cambios en los requisitos.</li> <li>• Facilita la priorización de tareas importantes mediante el Product Backlog.</li> <li>• Mejora la calidad del producto al integrar revisiones constantes.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere compromiso constante y alta participación del equipo y del Product Owner.</li> <li>• Puede ser complicado para equipos grandes o distribuidos.</li> <li>• No es adecuado para proyectos con requisitos completamente definidos desde el inicio.</li> <li>• Si no se aplican bien los roles o ceremonias, el proceso puede volverse ineficiente.</li> </ul>

	<ul style="list-style-type: none"> <li>• Enfoque en la entrega incremental de valor al cliente.</li> </ul>	<ul style="list-style-type: none"> <li>• Promueve la autoorganización y autonomía del equipo.</li> </ul>	
FDD: desarrollo basado en características	<ul style="list-style-type: none"> <li>• Se centra en desarrollar funcionalidades específicas denominadas características.</li> <li>• Incluye cinco etapas principales: desarrollo de un modelo general, creación de una lista de características, planificación por características, diseño por características, y construcción por características.</li> <li>• Las características son pequeñas funcionalidades que aportan valor al usuario final y se completan en ciclos cortos.</li> <li>• Uso de métricas y estándares para evaluar el progreso y la calidad.</li> <li>• Enfoque en equipos pequeños y colaborativos con roles definidos.</li> </ul>	<ul style="list-style-type: none"> <li>• Permite un seguimiento detallado del progreso gracias a las métricas.</li> <li>• Ayuda a identificar y priorizar funcionalidades que son realmente valiosas para el cliente.</li> <li>• Proporciona entregas frecuentes de valor al usuario.</li> <li>• Buena adaptabilidad a proyectos grandes y equipos distribuidos.</li> <li>• Reduce riesgos mediante ciclos cortos y manejables.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere planificación inicial detallada y tiempo para descomponer funcionalidades.</li> <li>• Depende de un buen modelado inicial para tener éxito.</li> <li>• Es menos adecuado para proyectos pequeños debido a su enfoque detallado.</li> <li>• Exige un alto nivel de experiencia y habilidad de los desarrolladores en roles específicos.</li> </ul>
Desarrollo esbelto	<ul style="list-style-type: none"> <li>• Busca maximizar el valor entregado al cliente mientras minimiza el</li> </ul>	<ul style="list-style-type: none"> <li>• Reducción significativa del desperdicio de recursos, tiempo y esfuerzo.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere una buena comprensión del flujo de trabajo y habilidades para identificar desperdicios.</li> </ul>

	<p>desperdicio en recursos y tiempo.</p> <ul style="list-style-type: none"> <li>• Enfocado en entregar resultados funcionales de manera rápida.</li> <li>• Adopta principios como optimización del flujo de trabajo, mejora continua y aprendizaje validado.</li> <li>• Utiliza ciclos iterativos y colaborativos.</li> <li>• Se centra en satisfacer las necesidades reales del cliente basándose en retroalimentación constante.</li> </ul>	<ul style="list-style-type: none"> <li>• Permite adaptarse rápidamente a los cambios en los requisitos.</li> <li>• Mejora la productividad y eficiencia del equipo al enfocarse en tareas esenciales.</li> <li>• Refuerza la colaboración entre equipos y con las partes interesadas.</li> <li>• Incrementa la calidad del producto gracias a un enfoque en la mejora continua.</li> </ul>	<ul style="list-style-type: none"> <li>• Exige un alto grado de participación activa del cliente durante todo el proceso.</li> <li>• Puede ser difícil implementar los principios esbeltos en equipos grandes o en entornos con poca flexibilidad.</li> <li>• La dependencia en retroalimentación frecuente puede retrasar decisiones clave si los involucrados no están disponibles.</li> </ul>
XP: programación extrema	<ul style="list-style-type: none"> <li>• Fomenta la comunicación constante entre desarrolladores y el cliente.</li> <li>• Ciclos de desarrollo cortos con entregas frecuentes de software funcional.</li> <li>• Prácticas clave: desarrollo guiado por pruebas (TDD), programación en parejas, integración continua, diseño simple y refactorización constante.</li> <li>• Enfoque en la calidad del código y en satisfacer los</li> </ul>	<ul style="list-style-type: none"> <li>• Mejora la calidad del software mediante pruebas continuas y refactorización.</li> <li>• Incrementa la colaboración entre desarrolladores al promover la programación en parejas.</li> <li>• Permite una rápida adaptación a cambios en los requisitos.</li> <li>• Facilita la detección temprana de errores con integración continua y pruebas frecuentes.</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser difícil para equipos con poca experiencia en sus prácticas.</li> <li>• Exige un alto compromiso del equipo y del cliente durante todo el proceso.</li> <li>• No es adecuado para proyectos muy grandes o equipos distribuidos.</li> <li>• La insistencia en pruebas y refactorización puede ser vista como una carga adicional si no se percibe como valiosa.</li> </ul>

	<p>requerimientos del cliente.</p> <ul style="list-style-type: none"> <li>Participación activa del cliente durante todo el desarrollo.</li> </ul>	<ul style="list-style-type: none"> <li>Genera software funcional desde las primeras etapas del proyecto.</li> </ul>	
Modelo de cascada	<ul style="list-style-type: none"> <li>Basado en un enfoque lineal y secuencial, donde cada fase debe completarse antes de pasar a la siguiente.</li> <li>Fases principales: análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento.</li> <li>Requiere una planificación detallada al inicio del proyecto.</li> <li>Se centra en la documentación extensa en cada etapa.</li> </ul>	<ul style="list-style-type: none"> <li>Proceso estructurado y fácil de entender.</li> <li>Buena documentación facilita el seguimiento y mantenimiento del proyecto.</li> <li>Adecuado para proyectos con requisitos claramente definidos desde el inicio.</li> <li>Permite medir el progreso con claridad, ya que cada fase tiene metas específicas.</li> </ul>	<ul style="list-style-type: none"> <li>Dificultad para adaptarse a cambios en los requisitos después de haber comenzado el proyecto.</li> <li>Problemas identificados en etapas avanzadas son costosos y difíciles de resolver.</li> <li>No permite retroalimentación continua, ya que el cliente sólo interactúa de forma significativa al inicio y al final.</li> <li>Poca flexibilidad, lo que puede generar demoras si hay incertidumbre en los requisitos iniciales.</li> </ul>
Modelo prototipo	<ul style="list-style-type: none"> <li>Basado en la creación de un modelo funcional inicial o prototipo para entender y refinar los requisitos del sistema.</li> <li>Permite iterar entre la construcción del prototipo y la retroalimentación del cliente.</li> </ul>	<ul style="list-style-type: none"> <li>Ayuda a aclarar requisitos ambiguos o inciertos.</li> <li>Incrementa la satisfacción del cliente al permitir retroalimentación temprana.</li> <li>Reduce el riesgo de construir un sistema que</li> </ul>	<ul style="list-style-type: none"> <li>Puede llevar a un aumento en costos y tiempo si los prototipos se rehacen constantemente.</li> <li>Riesgo de que los usuarios confundan el prototipo con el producto final.</li> </ul>

	<ul style="list-style-type: none"> <li>• El prototipo puede ser descartable (no evoluciona al producto final) o evolutivo (se mejora hasta convertirse en el producto final).</li> <li>• Enfocado en una comprensión temprana de las expectativas del usuario.</li> </ul>	<ul style="list-style-type: none"> <li>• no cumpla con las expectativas.</li> <li>• Facilita la detección de problemas antes de comprometer grandes recursos.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere habilidades adicionales para construir prototipos rápidos y funcionales.</li> <li>• En ocasiones, se presta menos atención a la calidad técnica en favor de la velocidad.</li> </ul>
RAD: desarrollo rápido de aplicaciones	<ul style="list-style-type: none"> <li>• Enfocado en el desarrollo rápido mediante la reutilización de componentes de software y herramientas automatizadas.</li> <li>• Uso de prototipos para iterar y validar requisitos con el cliente.</li> <li>• Ciclos de desarrollo cortos, con entregas funcionales frecuentes.</li> <li>• Equipos pequeños y altamente cualificados trabajan de manera colaborativa.</li> <li>• Se adapta mejor a proyectos con requisitos bien definidos y plazos ajustados.</li> </ul>	<ul style="list-style-type: none"> <li>• Reducción significativa en el tiempo de desarrollo.</li> <li>• Mayor satisfacción del cliente gracias a la retroalimentación continua y entregas frecuentes.</li> <li>• Menor riesgo de desarrollar características innecesarias o inadecuadas.</li> <li>• Promueve la reutilización de componentes, lo que reduce costos y esfuerzo.</li> <li>• Flexible ante cambios en los requisitos.</li> </ul>	<ul style="list-style-type: none"> <li>• Difícil de aplicar en proyectos grandes o muy complejos.</li> <li>• Requiere una alta participación del cliente durante el desarrollo.</li> <li>• Depende de la disponibilidad de herramientas automáticas y componentes reutilizables.</li> <li>• Menos enfoque en la documentación formal, lo que puede dificultar el mantenimiento posterior.</li> </ul>
Modelo de desarrollo de	<ul style="list-style-type: none"> <li>• Se enfoca en el desarrollo continuo y adaptable de sistemas en evolución.</li> </ul>	<ul style="list-style-type: none"> <li>• Muy flexible y adaptable a cambios durante el proceso de desarrollo.</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser más costoso y llevar más tiempo debido</li> </ul>

sistemas dinámicos	<ul style="list-style-type: none"> <li>• Basado en un enfoque flexible que permite ajustar los requisitos a medida que se avanza en el proyecto.</li> <li>• Utiliza la retroalimentación constante para ir ajustando el sistema a los cambios en los requisitos y entorno.</li> <li>• Está orientado a sistemas que tienen que evolucionar debido a nuevas condiciones o cambios en el entorno.</li> </ul>	<ul style="list-style-type: none"> <li>• Permite un alto grado de innovación, dado que el sistema se ajusta con frecuencia a nuevas necesidades.</li> <li>• Facilita la integración de nuevas características y la resolución de problemas a medida que surgen.</li> <li>• Ofrece una mejor capacidad para gestionar la incertidumbre y los cambios en los requisitos.</li> </ul>	<p>a la evolución continua del sistema.</p> <ul style="list-style-type: none"> <li>• Requiere un control constante y un seguimiento activo para evitar desvíos significativos en el alcance.</li> <li>• La continua adaptación puede generar complejidad en la arquitectura del sistema.</li> <li>• Menos predecible en cuanto a fechas de entrega y costos, lo que puede dificultar la planificación a largo plazo.</li> </ul>
Modelo espiral	<ul style="list-style-type: none"> <li>• Combina elementos de diseño y prototipado con los principios de desarrollo iterativo y la evaluación de riesgos.</li> <li>• El proceso se organiza en ciclos, llamados espirales, que incluyen fases de planificación, desarrollo, prototipado y evaluación de riesgos.</li> <li>• En cada espiral, el software se mejora y refina basándose en los resultados y la</li> </ul>	<ul style="list-style-type: none"> <li>• Identificación temprana de riesgos y problemas, permitiendo su resolución antes de que se agraven.</li> <li>• Flexibilidad para adaptarse a los cambios y nuevas necesidades durante el desarrollo.</li> <li>• Cada ciclo de espiral puede generar prototipos funcionales que mejoran con cada iteración.</li> <li>• Adecuado para proyectos grandes, complejos y con</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser costoso y lento debido a la iteración continua y la constante evaluación de riesgos.</li> <li>• Requiere experiencia en la gestión de riesgos y en la planificación.</li> <li>• Puede ser difícil de aplicar en proyectos pequeños o con requisitos claramente definidos desde el inicio.</li> <li>• Requiere una vigilancia constante para asegurarse de que se</li> </ul>

	<p>retroalimentación del ciclo anterior.</p> <ul style="list-style-type: none"> <li>• El enfoque en la gestión de riesgos es fundamental durante todo el proceso.</li> <li>• Adaptable y adecuado para proyectos complejos y grandes.</li> </ul>	<p>un alto nivel de incertidumbre.</p>	<p>siguen las fases y objetivos adecuados.</p>
JAD: Desarrollo de aplicaciones conjuntas	<ul style="list-style-type: none"> <li>• Método participativo en el que los usuarios y los desarrolladores trabajan juntos para definir los requisitos y diseñar el sistema.</li> <li>• Se enfoca en sesiones intensivas de trabajo, denominadas talleres, para llegar a consensos rápidos sobre los requisitos y soluciones del sistema.</li> <li>• Fomenta la comunicación directa y constante entre todas las partes interesadas: usuarios finales, desarrolladores, y analistas de negocio.</li> <li>• Requiere una alta implicación de los usuarios en el proceso de toma de decisiones.</li> </ul>	<ul style="list-style-type: none"> <li>• Acelera la recopilación de requisitos al involucrar directamente a los usuarios.</li> <li>• Facilita la creación de sistemas más ajustados a las necesidades reales de los usuarios.</li> <li>• Aumenta la satisfacción del cliente y las posibilidades de éxito del proyecto debido a la alta participación del usuario.</li> <li>• Mejora la calidad del producto gracias a la colaboración constante entre usuarios y técnicos.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere la disponibilidad y el compromiso activo de los usuarios, lo que puede ser un desafío.</li> <li>• El proceso puede ser difícil de gestionar si las expectativas de los participantes no están bien alineadas.</li> <li>• Puede ser costoso en términos de tiempo y recursos si no se estructura adecuadamente.</li> <li>• No es adecuado para proyectos muy grandes o cuando los usuarios finales son difíciles de involucrar.</li> </ul>



Proceso racional unificado	<ul style="list-style-type: none"> <li>• Basado en un enfoque iterativo e incremental que abarca todas las fases del ciclo de vida del software, desde la concepción hasta el mantenimiento.</li> <li>• Se estructura en cuatro fases principales: inicio, elaboración, construcción y transición.</li> <li>• Cada fase incluye un ciclo de planificación, diseño, implementación y pruebas.</li> <li>• El desarrollo se organiza en iteraciones, lo que permite la mejora continua del sistema a lo largo del tiempo.</li> <li>• Uso de herramientas UML (Unified Modeling Language) para modelar el sistema y facilitar la comunicación entre el equipo.</li> <li>• Énfasis en la gestión de riesgos, asegurando que estos se identifiquen y mitiguen en las fases tempranas del proyecto.</li> </ul>	<ul style="list-style-type: none"> <li>• Alto enfoque en la calidad del software, con pruebas realizadas en cada iteración.</li> <li>• Aumento de la visibilidad del proyecto y mejor control sobre la planificación y los riesgos.</li> <li>• Flexibilidad para adaptarse a cambios en los requisitos o el entorno durante el desarrollo.</li> <li>• Promueve la reutilización de componentes y la documentación estándar, facilitando el mantenimiento y la evolución del sistema.</li> <li>• Mejora de la colaboración entre los miembros del equipo al tener roles y responsabilidades bien definidos.</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser complejo de aplicar en proyectos pequeños o con equipos sin experiencia en el proceso.</li> <li>• Requiere un alto compromiso de recursos para asegurar la calidad en todas las iteraciones y fases.</li> <li>• La documentación y las fases pueden ser percibidas como pesadas o innecesarias en proyectos ágiles.</li> <li>• Dependencia de herramientas y metodologías (como UML) que pueden no ser útiles en todos los proyectos.</li> </ul>
Metodología DevOps	<ul style="list-style-type: none"> <li>• Combina desarrollo de software (Dev) y</li> </ul>	<ul style="list-style-type: none"> <li>• Mejora la velocidad y la calidad de las entregas</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser difícil de implementar en</li> </ul>

	<p>operaciones de TI (Ops) para mejorar la colaboración entre equipos de desarrollo y de infraestructura.</p> <ul style="list-style-type: none"> <li>• Enfocada en la automatización de procesos, desde la integración continua (CI) hasta la entrega continua (CD).</li> <li>• Promueve la cultura de colaboración, donde los desarrolladores y los equipos de operaciones comparten responsabilidades en todas las etapas del ciclo de vida del software.</li> <li>• Uso de herramientas que facilitan la automatización, monitoreo y pruebas.</li> <li>• Fomenta el uso de microservicios, infraestructura como código (IaC), contenedores (como Docker) y otras tecnologías que permiten una alta disponibilidad, escalabilidad y eficiencia.</li> </ul>	<p>mediante la automatización y la integración continua.</p> <ul style="list-style-type: none"> <li>• Permite una respuesta más rápida a los cambios de requisitos o problemas en producción.</li> <li>• Fomenta una colaboración más estrecha entre los equipos de desarrollo y operaciones, mejorando la eficiencia global.</li> <li>• Facilita la implementación de cambios pequeños y frecuentes sin comprometer la estabilidad.</li> <li>• Aumenta la fiabilidad de las aplicaciones a través de monitoreo constante y pruebas automatizadas.</li> </ul>	<p>organizaciones con estructuras tradicionales o jerárquicas.</p> <ul style="list-style-type: none"> <li>• Requiere una cultura organizacional de cooperación, lo que puede ser un desafío en equipos fragmentados.</li> <li>• Necesita de una inversión significativa en infraestructura, herramientas y formación del equipo.</li> <li>• Los problemas de integración entre equipos pueden surgir si no hay una alineación clara en procesos y objetivos.</li> </ul>
--	--	---	---

<p>Desarrollo de software adaptativo</p>	<ul style="list-style-type: none"> <li>• Enfoque flexible y adaptativo que responde rápidamente a los cambios en los requisitos, los cuales pueden evolucionar a lo largo del desarrollo.</li> <li>• El proceso de desarrollo se organiza en ciclos iterativos, permitiendo ajustes continuos según nuevas informaciones o condiciones del proyecto.</li> <li>• Fomenta la colaboración constante entre el equipo de desarrollo y el cliente o usuario final para asegurar que el producto se ajuste a sus necesidades.</li> <li>• Prioriza la entrega continua de pequeñas mejoras o funcionalidades funcionales, en lugar de una entrega masiva al final del proyecto.</li> <li>• El diseño del sistema es realizado de manera incremental, sin una planificación detallada y exhaustiva al principio.</li> </ul>	<ul style="list-style-type: none"> <li>• Alta flexibilidad para adaptarse a cambios y nuevas necesidades que surgen durante el desarrollo.</li> <li>• Mejora la satisfacción del cliente mediante entregas frecuentes y ajustes rápidos a sus requerimientos.</li> <li>• Fomenta una retroalimentación continua y activa del cliente, lo que asegura que el producto final sea más relevante y útil.</li> <li>• Reducción de riesgos, ya que los problemas se detectan temprano y pueden ser corregidos rápidamente.</li> <li>• El enfoque adaptativo permite que el equipo evolucione y mejore conforme se desarrollan nuevas tecnologías o mejores prácticas.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere una constante disponibilidad de los clientes o usuarios para proporcionar retroalimentación.</li> <li>• Puede generar una falta de claridad en la planificación a largo plazo, lo que dificulta prever el costo total o el tiempo necesario.</li> <li>• La evolución constante de requisitos y el diseño incremental pueden llevar a la fragmentación o falta de cohesión en el producto.</li> <li>• Requiere un nivel alto de comunicación y cooperación entre equipos, lo cual puede ser complicado en grandes proyectos o equipos distribuidos.</li> </ul>
--	---	--	---

<p>Desarrollo impulsado por el comportamiento</p>	<ul style="list-style-type: none"> <li>• El desarrollo se centra en especificar el comportamiento esperado del sistema en lugar de centrarse en los detalles técnicos o de implementación.</li> <li>• Utiliza herramientas y técnicas como BDD (Behavior Driven Development) para definir funcionalidades de manera clara y comprensible, normalmente usando lenguaje natural o expresiones legibles.</li> <li>• Implica la colaboración continua entre los desarrolladores, testers y clientes o usuarios para definir casos de prueba que reflejen el comportamiento del sistema.</li> <li>• Se enfoca en la validación continua del software mediante pruebas automatizadas que aseguran que el sistema cumpla con los comportamientos deseados.</li> </ul>	<ul style="list-style-type: none"> <li>• Facilita la comunicación entre los equipos técnicos y los no técnicos debido al uso de lenguaje claro y comprensible en las especificaciones.</li> <li>• Mejora la comprensión del negocio y de las necesidades del usuario, permitiendo una mayor alineación entre los desarrolladores y las partes interesadas.</li> <li>• Fomenta un enfoque proactivo hacia las pruebas, mejorando la calidad y reduciendo los errores en producción.</li> <li>• Ayuda a garantizar que el software desarrollado cumpla con los requisitos del cliente desde el principio, evitando malentendidos.</li> <li>• Soporta cambios frecuentes en los requisitos a través de la creación de nuevos escenarios de comportamiento.</li> </ul>	<ul style="list-style-type: none"> <li>• Puede ser complejo de aplicar al principio si el equipo no está acostumbrado al enfoque BDD o al uso de herramientas específicas.</li> <li>• Requiere una inversión inicial significativa para crear y mantener las pruebas basadas en el comportamiento.</li> <li>• Dependiendo del equipo, puede haber resistencia en adaptarse a una nueva forma de pensar sobre el desarrollo y las pruebas.</li> <li>• La correcta especificación de comportamientos puede ser desafiante, especialmente cuando los requisitos son ambiguos o poco claros.</li> </ul>
---	--	--	---

## Conclusión

Las metodologías de desarrollo de software son herramientas clave para estructurar y organizar el proceso de creación de productos tecnológicos. Estas metodologías permiten a los equipos gestionar proyectos de forma eficiente, asegurando que los resultados finales cumplan con los objetivos planteados. Entre las más conocidas se encuentran Agile, Scrum, Waterfall, Kanban y DevOps, cada una con enfoques particulares adaptados a diferentes necesidades.

En general, las metodologías buscan mejorar la colaboración y comunicación entre los equipos, clientes y otros interesados en el proyecto. Además, ayudan a establecer procesos claros que reducen riesgos asociados con cambios en los requisitos, retrasos o problemas técnicos. Por ejemplo, Agile promueve la flexibilidad y las entregas iterativas, mientras que Waterfall sigue un enfoque más lineal y predecible. Por su parte, DevOps fomenta la integración continua y la automatización para acelerar los ciclos de desarrollo.

Un beneficio importante es la mejora en la calidad del producto, ya que estas metodologías suelen incluir prácticas de prueba y retroalimentación constantes. También contribuyen a optimizar los recursos disponibles, minimizando desperdicios y tiempos muertos al priorizar tareas clave.

En conclusión, la elección de la metodología ideal dependerá de factores como la naturaleza del proyecto, su complejidad, el tamaño del equipo y la cultura organizacional. Sin embargo, todas comparten el objetivo de maximizar la eficiencia y garantizar la entrega de un software funcional, escalable y alineado con las expectativas del cliente y los objetivos del negocio.

## Bibliografía (Obviamente el poderoso CHATGPT)

Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley.

Beck, K., & Fowler, M. (2001). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addison-Wesley.

Sommerville, I. (2011). *Software Engineering* (9th ed.). Addison-Wesley.

Pressman, R. S. (2009). *Software Engineering: A Practitioner's Approach* (7th ed.). McGraw-Hill.

Boer, G. (1998). *Modelo de Cascada y Alternativas de Desarrollo en el Ciclo de Vida del Software*. IBM.

Van Vliet, H. (2008). *Software Engineering: Principles and Practice* (3rd ed.). Wiley.

Balaguer, R., & Holgado, C. (2015). *Modelos de Desarrollo de Software Adaptativos: Estrategias, procesos y nuevas tendencias*. Editorial UAM.