Desarrollo de Software

A. Juan y R. Dhanielt.
160004903 - 160004935

***Resumen*—** La ingeniería de software ha revolucionado la interacción con la tecnología moderna, siendo fundamental en áreas como la medicina, finanzas, etc. Este artículo presenta una revisión de algunos modelos, metodologías y marcos de trabajo empleados en el desarrollo de software, comparando su estructura, flexibilidad y enfoque, destacando sus ventajas y desafíos en diferentes contextos. Se determina que cada uno de los modelos, metodologías y marcos de trabajo, cumplen un propósito específico que no reemplaza los demás. El modelo en cascada ofrece una estructura detallada, adecuada para proyectos con requisitos estables, mientras que el desarrollo incremental proporciona mayor flexibilidad y mejor gestión de riesgos. Entre las metodologías ágiles, Scrum se destaca por su estructura clara y entrega continua de valor, en tanto que el Desarrollo Adaptativo de Software (DAS) permite mayor autoorganización y aprendizaje continuo. Los marcos de trabajo Spring y Django, aunque diferentes en su enfoque y lenguaje, son efectivos en sus respectivos dominios. Spring es ideal para aplicaciones empresariales complejas, mientras que Django facilita el desarrollo rápido con herramientas integradas. En conjunto, estos enfoques permiten adaptar el proceso de desarrollo a las necesidades específicas del proyecto, optimizando tanto la calidad del software como la eficiencia del proceso.

Palabras clave: Desarrollo de Software, Modelos, Metodologías, Marcos de Trabajo.

***Abstract*—** Software engineering has revolutionized the interaction with modern technology, being fundamental in areas like medicine, finance, etc. This article presents a review of some models, methodologies and frameworks used in software development, comparing their structure, flexibility and approach, highlighting their advantages and challenges in different contexts. It is determined that each of the models, methodologies and frameworks serves a specific purpose and does not replace the others. The waterfall model offers a detailed structure, suitable for projects with stable requirements, while incremental development provides greater flexibility and better risk management. Among agile methodologies, Scrum stands out for its clear structure and continuous delivery of value, while Adaptive Software Development (ASD) allows for more self-organization and continuous learning. The Spring and Django frameworks, while different in their approach and language, are effective in their respective domains. Spring is ideal for complex enterprise applications, while Django facilitates rapid development with integrated tools. Together, these approaches allow the development process to be tailored to the specific needs of the project, optimizing both software quality and process efficiency.

Keywords: Software Development, Models, Methodologies, Frameworks.

I. Introducción

La ingeniería de software ha transformado profundamente la manera en que las sociedades modernas interactúan con la tecnología. Desde aplicaciones móviles hasta sistemas complejos en medicina y finanzas, el software es el motor que impulsa el mundo digital. Sin embargo, para entender mejor, es necesario darle un vistazo al concepto de software.

El software es el conjunto de programas de cómputo y documentación asociada; los productos de software se desarrollan para un cliente en particular o para un mercado en general [1].

En el proceso de desarrollo de software existen diferentes piezas clave como los modelos, las metodologías y los marcos de trabajo; los cuales se emplean como herramientas para facilitar el proceso de desarrollo y poder desarrollar software de calidad.

Un modelo en ingeniería de software es una representación simplificada y abstracta de un sistema, proceso, o concepto. Se utilizan para entender, analizar, y comunicar ideas sobre cómo un sistema debe ser construido o cómo funciona [1].

Una metodología es un conjunto de principios, prácticas, procedimientos y herramientas que guían y organizan el proceso de desarrollo de software. Es un enfoque general que define cómo se debe planificar, estructurar, y gestionar un proyecto de software [1].

Un marco de trabajo es una estructura conceptual y tecnológica de soporte definida que sirve como base para la organización y desarrollo de software. Proporciona una plantilla o esquema predefinido que los desarrolladores pueden usar, extender o personalizar para crear aplicaciones más complejas [2]. El uso de marcos de trabajo en la ingeniería de software ayuda a estandarizar el proceso de desarrollo, reduce el tiempo de desarrollo, mejora la calidad del código y facilita el mantenimiento del software.

Este trabajo se enfoca en realizar una revisión bibliográfica y comparativa de algunos de los modelos, metodologías y marcos de trabajo que existen actualmente diseñados para garantizar el cumplimiento de la mayor cantidad de estándares de calidad posibles y hacer del proceso de desarrollo de software un proceso estandarizado, estructurado y secuencial.

II. Modelos

A. Modelo de cascada

Es el primer modelo publicado sobre el proceso de software que se deriva a partir de procesos más generales de ingeniería de sistemas, este modelo es un claro ejemplo de un proceso dirigido por un plan [1].

Este modelo consta de 6 pasos consecutivos puesto que la siguiente fase no debe comenzar si no hasta terminar la fase previa, esto no quiere decir que si en alguna etapa se encuentra un error de las etapas anteriores no se pueda devolver a esa etapa y continuar de nuevo, ya que el proceso del software no es un simple modelo lineal, si no que se va retroalimentando en cada una de las fases [1].

Cada una de estas etapas debe generar documentación, esto hace que el proceso sea visible, de modo que los administradores puedan monitorizar el progreso, uno de los principales inconvenientes de este modelo es que en una etapa muy temprana del desarrollo se deben poner compromisos, lo que dificulta responder a los requerimientos cambiantes del cliente [1].

Los pasos que se deben seguir en este modelo son:

1. *Definición:* La primera etapa del modelo en cascada es la definición de requisitos o especificación de las características que debe cumplir el software que se va a

desarrollar. Esta fase es posiblemente la más crucial, ya que el producto final depende en gran medida de las decisiones que se tomen en este punto. El enfoque principal es analizar las necesidades y preferencias del usuario. Además, es esencial documentar claramente las decisiones tomadas en esta etapa para que puedan ser consultadas en fases posteriores del proceso. Por esta razón, la documentación generada en esta fase debe ser precisa y estar disponible a lo largo de todo el desarrollo [5].

2. *Diseño:* El proceso de desarrollo de un programa implica analizar las posibles soluciones técnicas y elegir la más adecuada. Luego, se decide la estructura general del programa, dividiéndolo en partes y definiendo sus relaciones. Este análisis se repite para cada parte hasta definir completamente el programa y estar listo para codificar. Aunque existen varias soluciones posibles para cada parte, la elección de una de ellas suele basarse en la intuición, ya que no hay metodologías efectivas que guíen esta decisión [5].
3. *Codificación:* En proyectos de gran escala, la codificación suele ser la etapa más simple si el diseño está bien elaborado y detallado, lo que la hace casi automática. Un aspecto clave en esta fase es seleccionar el lenguaje de programación, aunque a veces ya se define durante el diseño. Hoy en día, la tendencia es utilizar lenguajes de alto nivel, aprovechando compiladores más eficientes, lo que aumenta la productividad al permitir a los programadores pensar más en términos de lógica que en detalles de la máquina. No obstante, estos lenguajes pueden ser más complejos de aprender, y los programadores, que tienden a ser conservadores, pueden mostrarse reacios a adoptarlos. Además, si una organización ya tiene una gran base de programas en un lenguaje específico, cambiar a otro puede ser una decisión complicada [5].
4. *Integración:* Una vez que los módulos están codificados, deben ser ensamblados, lo que puede traer consigo dificultades relacionadas con las interfaces entre módulos, la comunicación de datos compartidos y la coordinación de los flujos de ejecución. En proyectos más grandes, la gestión de versiones se vuelve un reto considerable. Afortunadamente, en esta etapa contamos con diversas herramientas CASE que pueden facilitar el proceso [5].
5. *Prueba:* En esta fase, es crucial verificar que el programa cumple con todas las especificaciones, aunque es casi imposible probarlo completamente, dejando posibles errores. Repetidos cambios pueden empeorar la situación si no se gestionan bien. Se están adoptando técnicas de verificación y validación como alternativas a las pruebas tradicionales, evaluando el software en un contexto más amplio, incluyendo hardware, usuarios e interfaces [5].
6. *Documentación:* La documentación es esencial para mantener un programa, incluso para quien lo codifica, ya que es fácil perderse después de un tiempo. No solo se debe documentar el código, sino todas las etapas del ciclo de vida, explicando las decisiones tomadas y sus razones. También es necesario crear documentación "de caja negra", que cubre el uso y las características externas del programa, incluyendo un manual de usuario y un manual técnico para su instalación y mantenimiento [5].

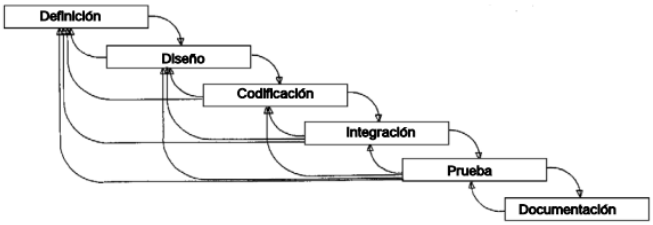

Fig 1. Flujo del modelo de cascada [4].

B. Desarrollo incremental

El desarrollo incremental consiste en crear una versión inicial del sistema, obtener retroalimentación del usuario, y luego mejorarla en varias iteraciones hasta lograr un sistema adecuado. Las actividades de especificación, desarrollo y validación se realizan de forma interrelacionada, con retroalimentación constante. Este enfoque, que es parte de los métodos ágiles, es más flexible que el modelo en cascada, permitiendo cambios más fáciles y económicos conforme avanza el desarrollo [1].

Cada versión o incremento del sistema incluye algunas funciones clave, lo que permite al cliente evaluar el progreso temprano y ajustar los requerimientos si es necesario. Este enfoque es común en el desarrollo de aplicaciones y puede combinarse con enfoques planificados o ágiles, adaptándose a las prioridades del cliente [1].

Sin embargo, el desarrollo incremental presenta desafíos: la falta de visibilidad del progreso puede dificultar la gestión, y la estructura del sistema puede degradarse con el tiempo si no se refactoriza adecuadamente. Estos problemas son más críticos en sistemas grandes y complejos, donde la estabilidad de la arquitectura y la claridad en la responsabilidad de los equipos son esenciales, lo que requiere una planificación previa en lugar de un enfoque puramente incremental [1].

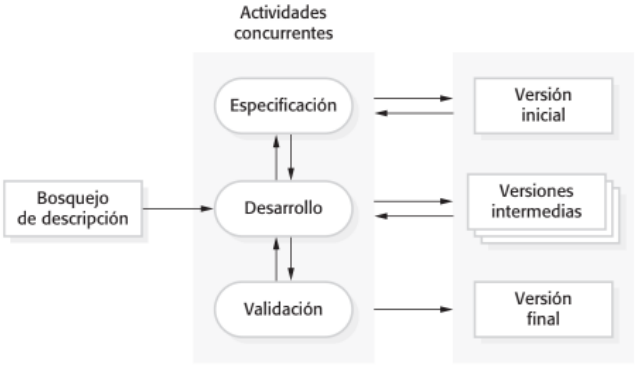

Fig 2. Flujo del Desarrollo incremental [4].

II. METODOLOGÍAS

A. Scrum

Es un método de desarrollo ágil de software creado por Jeff Sutherland y su equipo de desarrollo a principios de los 90, se utiliza para guiar el desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evaluación y entrega [3].

Scrum acentúa el uso de un conjunto de patrones de proceso del software que han demostrado ser eficientes para proyectos

con plazos de desarrollo muy apretados, requerimientos cambiantes y negocios críticos, este método de desarrollo se divide en las siguientes partes [3]:

1. *Retraso:* lista de requerimientos o características del proyecto que le dan un valor al negocio, a esta lista se le puede añadir otro aspectos en cualquier momento, el gerente del proyecto se encarga de actualizar y organizar las prioridades según se requiera.
2. *Sprints:* es el centro de la metodología scrum, es donde las ideas propuestas en el retraso cobran valor, son eventos con una longitud definida y fija de un mes o menos, un nuevo sprint comienza inmediatamente después de la terminación del anterior, para el sprint se coje un objetivo de la lista de retraso y se trabaja para cumplirlo, durante este proceso no se hacen cambios que pongan en riesgo el objetivo, la calidad no disminuye y el sprint solo puede ser cancelado si el objetivo se vuelve obsoleto. Solo el propietario del producto tiene la autoridad de cancelar el sprint [4].
3. *Reunión Scrum*: son reuniones cortas de una duración aproximada de 15 minutos las cuales se realizan a diario, en estas reuniones participan todos los miembros del equipo respondiendo 3 preguntas claves:

   -¿Qué hiciste desde la última reunión?.

   -¿Qué obstáculos estas encontrando?.

   -¿Qué planeas hacer mientras llega la siguiente reunión?.

   Un líder de equipo llamado maestro scrum, dirige la reunión y toma nota de las respuestas proporcionadas para poder evaluar y descubrir los problemas potenciales tan pronto como sea posible [3].
4. *Demostraciones preliminares:* se entrega el avance del software al cliente de modo que la funcionalidad que se haya implementado pueda demostrarse y este pueda evaluarla [3].

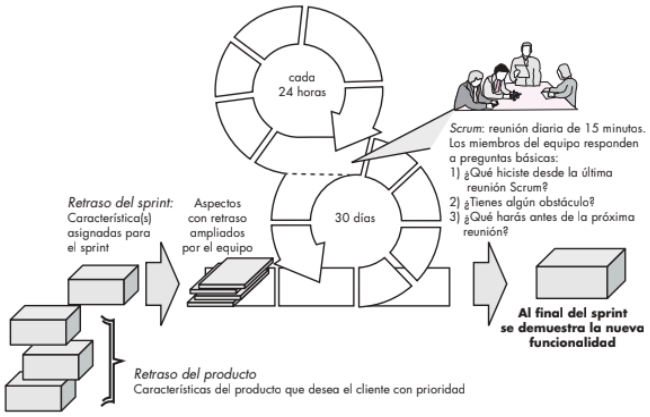

Fig 3. Flujo del proceso Scrum [3].

B. Desarrollo adaptativo de software (DAS)

El DAS, propuesto por Jim Highsmith, se enfoca en la colaboración y autoorganización del equipo para el desarrollo de software y sistemas complejos. Su enfoque está basado en tres fases principales [3]:

1. *Especulación:* se da inicio al proyecto y se realiza la planificación adaptativa. En esta etapa, se utiliza la información inicial del proyecto, que incluye el enunciado de misión, las restricciones del proyecto (como fechas de entrega y descripciones de usuario) y los requisitos básicos, para definir los ciclos de entrega o incrementos de software necesarios. Aunque el plan inicial es fundamental, se reconoce que es inevitable que cambie a medida que se avanza. Así, la información obtenida al final de cada ciclo permite revisar y ajustar el plan, adaptándolo a la realidad en la que trabaja el equipo DAS [3].
2. *Colaboración:* se centra en la interacción entre los miembros del equipo. La colaboración es clave para maximizar el talento y la producción creativa, aunque no siempre es fácil. Implica una comunicación efectiva y trabajo en equipo, pero también destaca la importancia de la creatividad individual en el pensamiento colaborativo. La confianza mutua es esencial, permitiendo a los miembros del equipo criticarse sin enojo, ayudarse sin resentimiento, y trabajar con dedicación. También es importante que cada miembro tenga las habilidades necesarias para contribuir al trabajo y que se comuniquen los problemas de manera que se puedan tomar acciones efectivas [3].
3. *Aprendizaje:* el enfoque se dirige hacia la adquisición de conocimiento a medida que el proyecto avanza. Highsmith destaca que los desarrolladores a menudo sobreestiman su comprensión del proceso, la tecnología y el proyecto. Por ello, el aprendizaje continuo es crucial para mejorar el entendimiento real. Los equipos DAS aprenden de varias maneras, como mediante grupos de enfoque, revisiones técnicas y análisis post mórtem del proyecto, para afinar su comprensión y mejorar continuamente [3].

La filosofía del DAS ofrece un enfoque valioso para el desarrollo de software al poner énfasis en la autoorganización, la colaboración y el aprendizaje, lo que aumenta significativamente la probabilidad de éxito en los proyectos de software [3].

Fig 4. Flujo del DAS [3].

III. MARCOS DE TRABAJO

A. Spring Framework.

Spring es un marco de trabajo de código abierto para el desarrollo de aplicaciones en Java. Es uno de los frameworks

más populares para crear aplicaciones empresariales robustas y escalables.

Características principales:

1. Spring maneja la creación y gestión de objetos, permitiendo un acoplamiento flexible entre componentes. Es el principio fundamental de Spring. En lugar de que el desarrollador cree y administre los objetos manualmente, el contenedor Spring se encarga de ello. Esto reduce el acoplamiento entre componentes, mejora la modularidad y facilita la prueba y mantenimiento del código [6].
2. Spring Security es un potente y altamente personalizable framework de autenticación y control de acceso. Proporciona protección contra ataques comunes, soporta diversos mecanismos de autenticación (como formularios, LDAP, OAuth), y permite un control de acceso granular basado en roles o expresiones [6].
3. Spring Boot es una extensión de Spring que simplifica significativamente la configuración y el despliegue de aplicaciones Spring. Proporciona configuración automática basada en las dependencias del classpath, incluye un servidor embebido, y ofrece 'starters' para simplificar la configuración del proyecto, permitiendo a los desarrolladores comenzar rápidamente con aplicaciones productivas [6].
4. Spring ofrece una capa de abstracción para la gestión de transacciones que funciona con diferentes APIs (JDBC, JTA, Hibernate, JPA). Permite definir transacciones de manera declarativa usando anotaciones o programáticamente, y proporciona control fino sobre el comportamiento transaccional, incluyendo propagación y aislamiento [6].
5. Permite que las dependencias de un objeto sean inyectadas automáticamente, reduciendo el acoplamiento y facilitando las pruebas. Spring inyecta automáticamente las dependencias requeridas por un objeto, ya sea a través del constructor, métodos setter o directamente en los campos. Esto elimina la necesidad de que los objetos busquen sus dependencias y permite una mayor flexibilidad en la configuración de la aplicación [7].
6. Spring posee un módulo que implementa el patrón Modelo-Vista-Controlador para aplicaciones web. Utiliza un servlet central (DispatcherServlet) para manejar todas las solicitudes HTTP y proporciona una estructura clara para organizar el código de la aplicación web. Soporta múltiples tecnologías de vista y facilita la creación de servicios RESTful [7].
7. Permite separar las preocupaciones transversales (como logging, seguridad, transacciones) del código principal de la aplicación. AOP en Spring permite definir "aspectos" que encapsulan estas funcionalidades y aplicarlas de manera declarativa a múltiples partes del código, reduciendo la duplicación y mejorando la modularidad [7].

Spring se utiliza comúnmente para desarrollar aplicaciones web empresariales, microservicios, y aplicaciones basadas en la nube. Su arquitectura modular permite a los desarrolladores usar solo las partes que necesitan.

B. Django.

Es un popular marco de trabajo para back-end basado en Python. Django sigue el patrón de diseño modelo-vista-controlador (MVC) y es conocido por su filosofía "baterías incluidas", lo que significa que proporciona muchas funcionalidades listas para usar.

Algunas de sus características son:

1. Django incluye un potente ORM que permite interactuar con bases de datos utilizando código Python en lugar de SQL directo. Esto abstrae las diferencias entre los sistemas de gestión de bases de datos y facilita la migración entre ellos. El ORM soporta múltiples bases de datos y proporciona una API intuitiva para realizar consultas complejas [8].
2. Genera automáticamente una interfaz de administración basada en los modelos de la aplicación. Esta interfaz permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los datos de la aplicación sin necesidad de escribir código adicional, lo que acelera significativamente el desarrollo y la gestión de contenidos [8].
3. Utiliza un sistema de mapeo URL-a-vista basado en expresiones regulares. Esto permite crear URLs limpias y semánticas, y facilita la organización y el mantenimiento de las rutas de la aplicación. Las URL se definen en archivos separados, promoviendo una estructura de proyecto clara y modular [8].
4. Django incluye un lenguaje de plantillas potente y flexible que permite separar la lógica de presentación del código Python. Las plantillas soportan herencia, lo que facilita la creación de layouts consistentes, y proporcionan filtros y tags para manipular datos en la capa de presentación [8].
5. Django incluye un sistema completo de autenticación y autorización. Proporciona modelos de usuario listos para usar, manejo de sesiones, y decoradores para controlar el acceso a vistas. También soporta autenticación de terceros y permite una personalización fácil para requisitos específicos de la aplicación [8].
6. Django implementa por defecto múltiples características de seguridad, como protección contra XSS (Cross-Site Scripting), CSRF, inyección SQL, y clickjacking. También maneja de forma segura la autenticación y el almacenamiento de contraseñas [8].

Estas características hacen de Django un marco de trabajo robusto y versátil para el desarrollo de aplicaciones web back-end, especialmente adecuado para proyectos que requieren un rápido desarrollo y una amplia gama de funcionalidades integradas.

IV. Análisis comparativo

A. Modelo en cascada vs modelo de desarrollo incremental.

- *Planificación:* El modelo en cascada es más planificado y estructurado desde el principio, lo que puede ser beneficioso para proyectos con requisitos bien definidos y estables. En contraste, el desarrollo incremental permite una mayor adaptabilidad, siendo

más adecuado para proyectos donde los requisitos son susceptibles de cambiar.

- *Flexibilidad vs. Control:* El modelo en cascada ofrece un control riguroso y una visión clara del progreso a través de la documentación detallada, pero carece de flexibilidad. El desarrollo incremental, por otro lado, sacrifica algo de ese control y visibilidad en favor de una mayor flexibilidad para responder a cambios.
- *Gestión de Riesgos:* El desarrollo incremental gestiona mejor los riesgos al permitir iteraciones rápidas y retroalimentación continua, lo que reduce el impacto de errores y cambios de requisitos. El modelo en cascada, al ser más rígido, puede ser más riesgoso si se detectan errores o cambios tardíos en el proceso.

B. *Metodología Scrum vs modelo de desarrollo adaptativo de software (DAS).*

- *Estructura vs. Flexibilidad:* Scrum tiene una estructura más rígida con roles y procedimientos bien definidos, mientras que DAS es más fluido y adaptable, permitiendo cambios más frecuentes en la planificación y ejecución.
- *Gestión de Cambios:* Scrum gestiona los cambios principalmente entre sprints, mientras que DAS los integra continuamente a lo largo del proyecto.
- *Enfoque del Equipo:* DAS se centra más en la autoorganización del equipo y en maximizar la colaboración, mientras que Scrum sigue un enfoque más directivo con roles claramente establecidos.
- *Aprendizaje:* DAS promueve el aprendizaje continuo como una parte integral del proceso, lo cual es menos enfatizado en Scrum, donde el enfoque está más en la ejecución efectiva de las tareas dentro de los sprints.

C. Marco de trabajo Spring vs Django.

- *Perspectiva técnica:* Spring y Django son marcos de desarrollo web robustos pero con enfoques distintos. Spring, basado en Java ofrece una arquitectura modular y altamente configurable utilizando inversión de control y programación orientada a aspectos para facilitar la creación de aplicaciones empresariales escalables. Por otro lado, Django fundamentado en Python adhiere al principio “baterías incluidas”, proporcionando un ORM potente, un sistema de administración integrado y una estructura MTV (Model-Template-View) que acelera el desarrollo.
- *Flexibilidad:* Spring destaca en su flexibilidad y ecosistema extenso, incluyendo Spring Boot para configuraciones rápidas, mientras que Django sobresale en su rapidez de desarrollo y convenciones predefinidas.
- *Rendimiento:* Spring tiende a tener una ventaja en aplicaciones de gran escala debido a la JVM, mientras que Django puede ser más eficiente en prototipos rápidos y aplicaciones de tamaño medio, la elección dependerá de factores como el lenguaje de programación preferido, los requisitos del proyecto y la experiencia del equipo de desarrollo.

V. Conclusiones

- Los modelos de desarrollo de software, como el de cascada y el incremental ofrecen enfoques distintos para abordar proyectos. El modelo cascada proporciona una estructura clara y documentación detallada, ideal para proyectos con requisitos estables. En contraste, el desarrollo incremental ofrece mayor flexibilidad y mejor gestión de riesgos, adaptándose mejor a entornos con requisitos cambiantes.
- Las metodologías ágiles, representadas por Scrum y el Desarrollo Adaptativo de Software (DAS), han revolucionado la gestión de proyectos de software. Scrum ofrece una estructura definida como sprints y roles claros, facilitando la entrega continua de valor. DAS, por su parte, enfatiza la autoorganización y aprendizaje continuo, proporcionando mayor flexibilidad.
- Los marcos de trabajo como Spring y Django representan diferentes filosofías en el desarrollo web. Spring, basado en Java, destaca por su modularidad y extenso ecosistema siendo ideal para aplicaciones complejas. Django, en Python, adopta un enfoque “baterías incluidas”, facilitando un desarrollo rápido con herramientas integradas.
- Cada uno de los modelos y metodologías de desarrollo descritas con anterioridad, no se reemplazan entre sí, ya que cada una cuenta con enfoques distintos que facilitan el desarrollo del software dependiendo de la robustez y finalidad del mismo. Asimismo sucede con los marcos de trabajo; estos al fin y al cabo terminan siendo elegidos entre otras cosas, por preferencias del desarrollador.

Referencias

[1] I. Sommerville, Ingeniería de software, 9ª ed. Pearson Educación, 2011, pp. 29-74.

[2] Gómez-Zea, José Manuel et al. “Optimización de la documentación en proyectos de software ágiles: Buenas prácticas y artefactos en el marco de trabajo SCRUM.” Programación Matemática y Software, 2023, pp. 2-4.

[3] R. S. Pressman, “Ingeniería de Software: Un Enfoque Práctico”, 7th ed., McGraw-Hill, 2010, pp. 68-71.

[4] K. Schwaber and J. Sutherland, “La Guía Definitiva de Scrum: Las Reglas del Juego”, Nov. 2020, pp. 7-8.

[5] F. Sáez Vacas, "Complejidad y Tecnologías de la Información", 1st ed., Octubre 2009, pp. 297-303.

[6] Spring Framework. (s. f.). Spring Framework. https://spring.io/projects/spring-framework.

[7] Martinez, Eugenia Pérez. Desarrollo de aplicaciones mediante el Framework de spring. Ra-Ma Editorial, 2011, .pp. 12-46

[8] Django overview. (s. f.). Django Project. https://www.djangoproject.com/start/overview.