Capítulo 3

Metodologías y procesos de desarrollo de software

Mónica Elizabeth Páez Padilla

Resumen

Este capítulo se enfoca en la relevancia esencial de las metodologías y procesos de desarrollo de software para lograr programas confiables y de alta calidad. Su objetivo central es explorar en detalle una gama de metodologías y paradigmas que guían este proceso, delineando sus características fundamentales y su clasificación en categorías como estructuradas, orientadas a objetos y ágiles. Además, el capítulo explora los ciclos de vida del desarrollo de software, los paradigmas que orientan la planificación y ejecución de proyectos, y clasifica los paradigmas de proceso comunes. Finalmente, se analiza la importancia del proceso de desarrollo y la función de los estándares en asegurar coherencia y calidad en los proyectos de software. En resumen, el capítulo proporciona una comprensión sólida de las metodologías, paradigmas y procesos que moldean la creación de software en la industria de la ingeniería de software.
Palabras claves: desarrollo de software, proyectos, planificación.

Páez Padilla, M.E. (2023). Metodologías y procesos de desarrollo de software. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 123-167). Religación Press. http://doi.org/10.46652/religacionpress.89.c84

3.1 Metodologías de Desarrollo de Software

3.1.1 Definiciones metodología de desarrollo de software.

Las metodologías de desarrollo de software abarcan un conjunto de enfoques y procedimientos de organización de empleados para concebir soluciones de software. El propósito de estas diversas metodologías radica en estructurar equipos de trabajo de manera eficiente para lograr una ejecución óptima de las funcionalidades de un programa.

Cuando se busca crear productos o respuestas para un cliente o mercado específico, es esencial considerar aspectos como los gastos, la programación, la complejidad, el equipo disponible, los idiomas empleados, entre otros. Estos elementos se integran en un enfoque de desarrollo que posibilita estructurar la mano de obra de manera altamente organizada (Santander, 2023).

3.1.2 Características metodología de desarrollo de software.

- Un proceso integral de ciclo de vida, abarcando tanto aspectos empresariales como técnicos
- Un conjunto exhaustivo de conceptos y modelos que mantengan una coherencia interna

- Un compendio de reglas y directrices
- Una descripción total de los elementos a desarrollar
- Una notación utilizada para trabajar, preferentemente respaldada por diversas herramientas CASE y diseñada para una experiencia óptima del usuario
- Un repertorio de técnicas con validación
- Un conjunto de métricas, junto con asesoramiento sobre calidad, estándares y enfoques de prueba
- Identificación de los roles dentro de la organización
- Orientación para la gestión de proyectos y aseguramiento de la calidad
- Asesoramiento sobre la administración de bibliotecas y la reutilización

3.1.3 Clasificación de las metodologías de desarrollo: estructurados, orientadas a objetos, ágiles.

Las estrategias empleadas en la creación de software se dividen en diversas corrientes principales, que incluyen los métodos estructurados, los orientados a objetos y las metodologías ágiles.

Según Maida y Pacienzia (2015) dicen que a continuación, se ofrece una descripción de cada una de estas categorías:

Metodologías Estructuradas:

Las metodologías estructuradas se fundamentan en un enfoque secuencial y jerárquico para crear software. Su enfoque se centra en dividir un proyecto en etapas bien definidas y lineales, destacando la importancia del análisis y diseño antes de la implementación. Ejemplos de estas metodologías son el modelo en cascada y el modelo en V.

Metodologías Orientadas a Objetos:

Las metodologías orientadas a objetos se centran en el diseño y desarrollo de software basado en objetos, los cuales son unidades autónomas que encapsulan datos y funcionalidades. Estas metodologías promueven la reutilización, modularidad y flexibilidad. Ejemplos de estas metodologías incluyen el Lenguaje de Modelado Unificado (UML) y los Métodos de Desarrollo de Software Orientado a Objetos (Object-Oriented Software Development Methods, OOSD).

Metodologías Ágiles:

Las metodologías ágiles son enfoques flexibles y colaborativos que priorizan la adaptación y la entrega continua de software funcional. Se basan en la interacción frecuente con los clientes, la comunicación entre los miembros del equipo y la capacidad de responder de manera ágil a los cambios. Ejemplos de estas me-

todologías son Scrum, Kanban, Extreme Programming (XP) y Lean. Cada una de estas categorías posee sus propias ventajas y desventajas. La elección de una metodología adecuada dependerá de factores como la naturaleza del proyecto, las preferencias del equipo, los plazos y la complejidad del software a desarrollar. Elegir la metodología correcta es crucial para maximizar la eficiencia y la calidad en el proceso de desarrollo.

Autoevalución 9

1. **¿Cuál es el propósito principal de las metodologías de desarrollo de software mencionadas en el texto?**
    1. Crear soluciones de software eficientes.
    2. Organizar equipos de trabajo de manera óptima.
    3. Maximizar los gastos en desarrollo de software.
    4. Utilizar múltiples idiomas en el desarrollo de software.
2. **¿Qué elementos se deben considerar al buscar crear productos o soluciones de software según el texto?**
    a. El clima.
    b. La historia de la empresa.
    c. Los gastos, la programación, la complejidad y otros factores.
    d. El costo de los servidores.
3. **¿Qué papel desempeñan las metodologías de desarrollo de software en la organización de equipos de trabajo?**
    a. Reducen la eficiencia de los equipos.
    b. Aumentan la complejidad del proceso de desarrollo.
    c. Estructuran la mano de obra de manera altamente organizada.

d. No tienen impacto en la organización de equipos.

4. **¿Cuál es uno de los aspectos clave mencionados en el texto que se deben considerar al elegir una metodología de desarrollo de software?**
    1. El idioma del cliente.
    2. La ubicación geográfica del equipo de desarrollo.
    3. La complejidad del proyecto.
    4. El nombre de la empresa.
5. **¿Cuál es la finalidad principal de las metodologías de desarrollo de software en relación con las funcionalidades de un programa?**
    1. Aumentar la complejidad de las funcionalidades.
    2. Lograr una ejecución óptima de las funcionalidades.
    3. Maximizar los costos de desarrollo.
    4. Ignorar las funcionalidades del programa.
6. **¿Cuál es el enfoque principal de las metodologías estructuradas?**
    1. Enfoque secuencial y jerárquico.
    2. Enfoque flexible y colaborativo.
    3. Enfoque centrado en objetos.
    4. Enfoque centrado en la adaptación continua.
7. **¿Qué importancia tienen el análisis y diseño en las metodologías estructuradas?**
    a. Mínima importancia.
    b. Importancia moderada.
    c. Importancia alta.
    d. No se menciona en el texto.
8. **¿Qué modelos son ejemplos de metodologías estructuradas?**
    1. Modelo en cascada y modelo en V.

2. Scrum y Kanban.
3. UML y OOSD.
4. Agile y Lean.

9. ¿En qué se centran las metodologías orientadas a objetos?

a. En el diseño secuencial.
b. En la adaptación continua.
c. En el desarrollo basado en objetos.
d. En la comunicación con el cliente.

10. ¿Qué promueven las metodologías orientadas a objetos?

1. La reutilización, modularidad y flexibilidad.
2. La entrega continua de software funcional.
3. La adaptación a cambios sin restricciones.
4. La comunicación entre los miembros del equipo.

11. ¿Cuál es un ejemplo de lenguaje de modelado orientado a objetos mencionado en el texto?

a. Scrum.
b. UML.
c. Kanban.
d. Lean.

12. ¿Qué priorizan las metodologías ágiles?

1. La entrega única de software.
2. La comunicación con el cliente.
3. La adaptación continua y la entrega frecuente de software funcional.
4. La planificación a largo plazo.

13. ¿En qué se basa la interacción frecuente en las metodologías ágiles?

1. En la comunicación escrita.
2. En la comunicación con otros equipos.

3. En la comunicación con el cliente.
4. No se basa en la interacción frecuente.

14. ¿Cuál de las siguientes no es una metodología ágil mencionada en el texto?

a. Scrum.
b. Kanban.
c. UML.
d. Extreme Programming (XP).

15. ¿Qué factores influyen en la elección de una metodología adecuada?

a. La preferencia del equipo.
b. El clima.
c. La ubicación geográfica.
d. Todas las anteriores.

3.2 Definiciones metodología de desarrollo de software

3.2.1 Ciclos de vida de Desarrollo de Software.

Concepto

El proceso de desarrollo de software a lo largo de su ciclo de vida (también reconocido como SDLC o Ciclo de Vida del Desarrollo de Sistemas) involucra las etapas necesarias para validar la creación del software, asegurando que este se ajuste a los requisitos establecidos y permitiendo la verificación de los procedimientos de desarrollo. Esto garantiza que los métodos utilizados sean apropiados (Bournissen, 1999).

La génesis de este enfoque surge del hecho de que corregir errores que se detectan en etapas avanzadas, durante la implementación, resulta sumamente costoso. Mediante el uso de enfoques metodológicos adecuados, es posible identificar problemas a tiempo, permitiendo que los programadores se concentren en la calidad del software y cumplan con los plazos y los costos preestablecidos (Bournissen, 1999).

Fases de desarrollo de software

La metodología de desarrollo de software se constituye como un enfoque estructurado para la ejecución, gestión y supervisión de un proyecto, con el fin de aumentar significativamente las probabilidades de éxito. Esta sistematización establece cómo dividir

un proyecto en componentes más pequeños para estandarizar su gestión.

En este contexto, una metodología de desarrollo de software engloba los procesos que deben ser seguidos de manera ordenada para idear, construir y mantener un producto de software, desde el instante en que se identifica la demanda del producto hasta que se logra el objetivo para el cual fue concebido.

De esta manera, las etapas del proceso de desarrollo de software son las siguientes:

Planificación

Antes de comenzar un proyecto de desarrollo de un sistema de información, es esencial realizar una serie de actividades que desempeñarán un papel fundamental en su éxito. Estas actividades son comúnmente referidas como la "fase inicial difusa" del proyecto, ya que no están restringidas por plazos específicos.

Algunas de las actividades en esta etapa engloban acciones como definir el alcance del proyecto, llevar a cabo un análisis de viabilidad, evaluar los riesgos asociados, estimar los costos involucrados en el proyecto, planificar su cronograma y distribuir los recursos para las diferentes fases del proyecto.

Análisis

Indudablemente, resulta necesario investigar con precisión

cuál es la función que el software debe desempeñar. En consecuencia, la etapa de análisis dentro del ciclo de vida del software representa el proceso mediante el cual se busca con precisión determinar lo que se necesita y obtener una comprensión total de los requisitos del sistema (las características específicas que el sistema debe tener).

Diseño

En esta fase, se investigan diversas posibilidades de cómo implementar el software que se va a crear y se selecciona la estructura básica del mismo. El proceso de diseño es complejo y su progreso debe basarse en un enfoque que involucre iteraciones.

Es viable que la solución inicial no resulte óptima, en cuyo caso es necesario pulirla. Sin embargo, existen catálogos de modelos de diseño que resultan extremadamente beneficiosos al captar los errores cometidos por otros, evitando así caer en los mismos obstáculos.

Implementación

En esta etapa, resulta fundamental elegir las herramientas apropiadas, configurar un ambiente de desarrollo que simplifique las tareas y decidir sobre un lenguaje de programación adecuado para el tipo de software que se va a desarrollar. Esta elección estará influenciada tanto por las decisiones de diseño adoptadas como por el entorno en el que el software estará funcionando.

Al codificar, es esencial evitar que el código se vuelva incomprensible, y esto se logra al seguir directrices como las siguientes:

- Evitar la creación de bloques de control que carezcan de estructura.
- Llevar a cabo una precisa identificación de las variables y definir su extensión.
- Elegir algoritmos y estructuras de datos apropiados para abordar el problema específico.
- Mantener la simplicidad en la lógica de la aplicación.
- Incluir una documentación y comentarios adecuados en el código de los programas.
- Enriquecer la legibilidad del código a través de la aplicación de reglas de formato previamente acordadas en el equipo de desarrollo.

También resulta crucial contemplar la adquisición de los recursos requeridos para el correcto desempeño del software, y a medida que se programa, crear casos de prueba que posibiliten la evaluación de su operatividad.

Pruebas

Dado que todos somos susceptibles a cometer errores, la etapa de pruebas en el ciclo de vida del software tiene como objetivo identificar los errores que puedan haber surgido en las fases pre-

vias para subsanarlos. Claro está, el objetivo óptimo es realizar esta corrección antes de que los usuarios finales lleguen a notarlos. Se considera que una prueba ha sido exitosa si logra identificar algún tipo de error.

Instalación o despliegue

El paso siguiente implica poner en marcha el software, por lo cual es esencial llevar a cabo una planificación del entorno considerando las interrelaciones existentes entre los distintos elementos que lo componen.

Es factible que algunos componentes funcionen sin inconvenientes de manera individual, pero al unirse, puedan generar dificultades. Por esta razón, es fundamental recurrir a combinaciones previamente probadas que no ocasionen conflictos de compatibilidad.

Uso y mantenimiento

Este constituye uno de los momentos de mayor relevancia en el ciclo de vida de desarrollo de software. Dado que el software no experimenta desgaste ni deterioro en su uso, su mantenimiento abarca tres aspectos distintos:

- Rectificación de los fallos identificados durante su período de utilización (mantenimiento correctivo).
- Ajuste para satisfacer nuevas necesidades emergentes

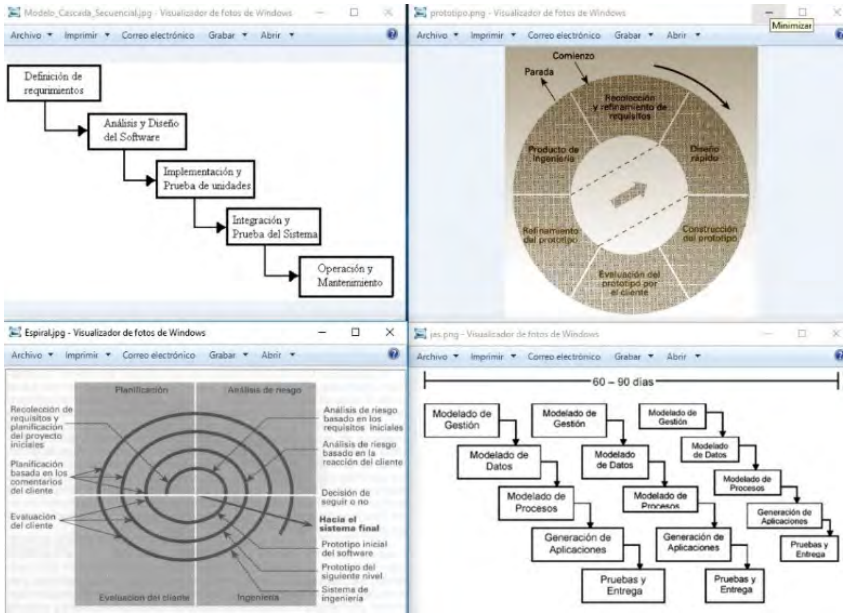
(mantenimiento adaptativo).

- Introducción de nuevas capacidades y funciones (mantenimiento perfectivo).

A pesar de que pueda parecer paradójico, cuanto mayor es la calidad del software, mayor inversión temporal debe destinarse a su mantenimiento. Esto se debe en gran medida al incremento en su uso (inclusive de maneras que no se habían anticipado), lo cual conlleva más propuestas de mejora y optimización (Cedeño, 2015).

3.2.2 Definición de paradigmas de proceso

Figura 7. Paradigmas en el desarrollo de software



Nota. El grafico 7 representa los paradigmas en el desarrollo de software

Dentro del campo de la Ingeniería de Software, un paradigma se define como un conjunto de técnicas, herramientas y procedimientos que se combinan con la finalidad de establecer un modelo.

En el ámbito de la Ingeniería de Software, cada metodología de desarrollo de software ofrece su propio enfoque único para el proceso de construcción del software. La Ingeniería de Software establece paradigmas de desarrollo estructurado como un punto de partida en un proyecto de software (Maida y Pacienzia, 2015).

Cuando ninguna de estas metodologías se adecua a la problemática en cuestión, los desarrolladores pueden verse en la necesidad de combinar paradigmas existentes o incluso crear uno nuevo.

Para abordar desafíos prácticos, el profesional de la ingeniería de software debe incorporar una estrategia que guíe el proceso, los métodos y el conjunto de herramientas utilizadas. Esta estrategia es comúnmente conocida como un modelo de proceso o paradigma en el campo de la ingeniería de software.

La elección de un modelo de proceso para la ingeniería de software depende de la naturaleza del proyecto y de la aplicación, así como de los métodos, herramientas, controles y entregables necesarios. Los modelos o paradigmas más comúnmente empleados en el desarrollo de software incluyen el enfoque en cascada, el enfoque de prototipos y el enfoque en espiral (Maida y Pacienzia, 2015).

Autoevaluación 10

1. **¿Qué significa el acrónimo SDLC en el contexto del desarrollo de software?**
   a. Sistema de Desarrollo de Lenguaje de Código.
   b. Software Development Life Cycle (Ciclo de Vida del Desarrollo de Software).
   c. Sistema de Diseño y Lenguaje de Código.
   d. Seguridad de Desarrollo de Lenguaje de Código.
2. **¿Por qué es importante seguir un enfoque metodológico adecuado en el desarrollo de software?**
   1. Para reducir el tamaño del proyecto.
   2. Para aumentar la complejidad del software.
   3. Para estandarizar la gestión y aumentar las probabilidades de éxito.
   4. Para acelerar el desarrollo.
3. **¿Qué actividades se llevan a cabo en la fase de Planificación del ciclo de vida del software?**
   1. Codificación y pruebas.
   2. Definición del alcance, análisis de viabilidad y evaluación de riesgos.
   3. Diseño de la interfaz de usuario.
   4. Instalación y despliegue.
4. **¿Cuál es el propósito de la fase de Análisis en el desarrollo de software?**
   a. Seleccionar herramientas de desarrollo.
   b. Determinar los requisitos del sistema.
   c. Diseñar la estructura del software.
   d. Codificar el programa.

5. **¿Qué se busca lograr en la fase de Diseño del ciclo de vida del software?**
    1. Determinar los requisitos del sistema.
    2. Seleccionar herramientas de desarrollo.
    3. Elegir la estructura básica del software.
    4. Identificar errores en el código.
6. **¿Qué se debe evitar al codificar el software durante la fase de Implementación?**
    1. Documentación y comentarios adecuados.
    2. Identificación precisa de variables y definición de su extensión.
    3. Creación de bloques de control con estructura.
    4. Simplificación de la lógica de la aplicación.
7. **¿Cuál es el objetivo de la fase de Pruebas en el ciclo de vida del software?**
    a. Identificar errores y corregirlos antes de que los usuarios los noten.
    b. Documentar el código fuente.
    c. Seleccionar el lenguaje de programación.
    d. Implementar el software.
8. **¿Qué aspecto se considera exitoso en una prueba de software?**
    1. Identificar errores.
    2. No encontrar ningún error.
    3. Encontrar errores que los usuarios también encuentren.
    4. Documentar el código fuente.

9. ¿Qué implica la fase de Instalación o despliegue en el ciclo de vida del software?

1. Poner en marcha el software y considerar interrelaciones entre componentes.
2. Identificar errores en el código fuente.
3. Seleccionar herramientas de desarrollo.
4. Codificar el software.

10. ¿Por qué el mantenimiento es una fase crucial en el ciclo de vida del desarrollo de software?

1. Porque el software se desgasta con el tiempo.
2. Porque el mantenimiento es más costoso que el desarrollo inicial.
3. Porque el software no experimenta desgaste y puede requerir correcciones, ajustes y mejoras.
4. Porque el mantenimiento no es necesario en el desarrollo de software de alta calidad.

11. ¿Cómo se define un paradigma en el contexto de la Ingeniería de Software?

1. Como un modelo de proceso.
2. Como un conjunto de técnicas de programación.
3. Como un enfoque único para el desarrollo de software.
4. Como un conjunto de herramientas de desarrollo.

12. ¿Por qué los desarrolladores de software pueden necesitar combinar paradigmas existentes o crear uno nuevo?

a. Para aumentar la complejidad del proyecto.
b. Cuando ninguna metodología existente se ajusta a la problemática en cuestión.
c. Para reducir los costos del desarrollo.
d. Para seguir las tendencias actuales en desarrollo de software.

13. ¿Qué se entiende por "modelo de proceso" o "paradigma" en la Ingeniería de Software?

1. Un conjunto de herramientas de desarrollo.
2. Una estrategia que guía el proceso, los métodos y las herramientas utilizadas.
3. Un lenguaje de programación.
4. Un conjunto de técnicas de programación.

14. ¿Cuál es un factor importante a considerar al elegir un modelo de proceso en la Ingeniería de Software?

1. La complejidad del código fuente.
2. La naturaleza del proyecto y de la aplicación, así como los métodos y herramientas necesarios.
3. La preferencia personal del desarrollador.
4. El tamaño del equipo de desarrollo.

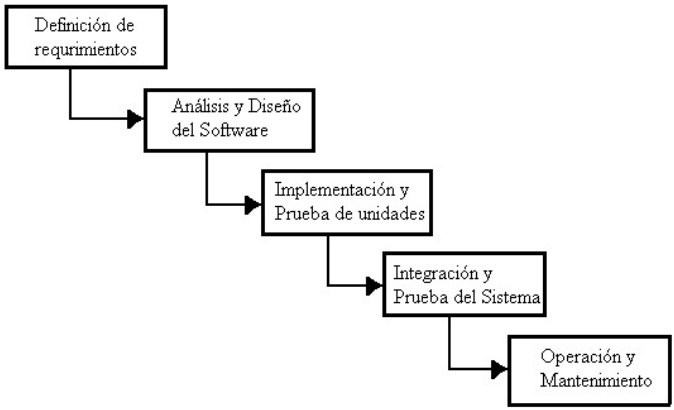
15. ¿Cuáles son algunos de los modelos o paradigmas más comúnmente empleados en el desarrollo de software mencionados en el texto?

a. Enfoque en criptografía, enfoque de redes neuronales, enfoque de inteligencia artificial.
b. Enfoque en cascada, enfoque de prototipos, enfoque en espiral.
c. Enfoque de diseño gráfico, enfoque de marketing, enfoque de ventas.
d. Enfoque de hardware, enfoque de software, enfoque de seguridad.

3.3 Clasificación de los paradigmas de proceso:

3.3.1 Modelo en cascada

Figura 8.Modelo Lineal Secuencial o de Cascada (Waterfall)



Nota. La figura 8 hace mención al modelo tradicional o en cascada, mostrando cada una de las etapas se desarrollan de manera consecutiva y secuencial.

Es un procedimiento de desarrollo secuencial en el que los pasos de progreso descienden de manera continua, siguiendo una secuencia similar al flujo de una cascada de agua. Esta secuencia abarca fases que van desde el análisis de requerimientos, el diseño, la implementación, las pruebas (validación), la integración y el mantenimiento.

Comúnmente, se atribuye la primera descripción formal del modelo en cascada a un artículo escrito por Winston Royce en 1970, aunque en dicho artículo Royce no empleó el término "cascada".

Este modelo representa el paradigma más antiguo y predominante durante la era del método estructurado. El número de fases propuestas puede variar según el proyecto en desarrollo, pero en este enfoque existen etapas comunes.

Dentro de este enfoque, las fases de desarrollo se consideran como procesos que ocurren en diferentes momentos en el tiempo, lo que implica que no pueden llevarse a cabo simultáneamente. Cada fase comienza después de que se haya completado la fase anterior, y para avanzar a la siguiente etapa, es necesario cumplir con todos los objetivos de la fase anterior.

Las etapas en este paradigma siguen una secuencia lineal. Cuando se detecta un error en alguna fase, generalmente es necesario retroceder hasta la fase inicial de análisis de requisitos del sistema. Aunque es posible retroceder a través de las etapas, esto conlleva un esfuerzo significativo y puede resultar en el fracaso del proyecto (Sulbarán, 2018).

Definición de los requisitos

Dentro de este procedimiento, se reconocen las demandas y condiciones del cliente en relación al software.

Según Sommerville (2011), nos señala las siguientes definiciones:

Análisis y Diseño: Durante la etapa de análisis, se evalúa la posibilidad del software tanto en términos técnicos como económicos, además se trazan los pasos y el financiamiento previsto. En la fase de diseño del software, el enfoque se centra en cuatro características clave de un programa: la organización de los datos, la arquitectura del software, las representaciones de la interfaz y los procedimientos detallados (algoritmos).

Codificación: El modelo diseñado previamente es transformado en un formato que la máquina pueda comprender. Esta tarea es realizada por el proceso de generación de código. Cuando el diseño se elabora de manera exhaustiva, la generación de código se ejecuta de manera automática.

Pruebas: El proceso de pruebas se focaliza tanto en los aspectos internos lógicos del software como en sus funciones externas. Estas pruebas tienen como objetivo identificar fallos y verificar que las entradas predefinidas generen resultados concretos que se alineen con los resultados esperados.

Mantenimiento: Es innegable que el software experimentará modificaciones después de su entrega al cliente. El proceso de mantenimiento implica volver a implementar todas las fases anteriores en un programa ya existente en lugar de uno nuevo.

3.3.2 Desarrollo incremental

Figura 9. Metodología de Desarrollo Incremental.

Nota. Con relación a la metodología de desarrollo incremental, es posible observar las etapas de su proceso de desarrollo en la Figura 9.

La metodología incremental inicia con un diseño inicial que abarca aspectos fundamentales, denominado esquema de descripción. A medida que avanza el proceso de desarrollo, se elaboran versiones progresivamente más avanzadas del sistema, culminando en una versión definitiva que cumple plenamente con las necesidades del usuario y satisface todos los requisitos de manejo y gestión de la información.

Tal como afirma León et al. (2021), que el enfoque incremental emplea secuencias de pasos lineales progresivos, a medida que

avanza en el calendario. Entre las actividades simultáneas se hace referencia a la especificación, en la que se establecen los requisitos y se recopila la información específica para ese incremento, junto con su análisis y diseño correspondiente. Posteriormente, se procede a la etapa de desarrollo en el lenguaje de programación elegido, y al final se verifica la congruencia de los resultados obtenidos con los requisitos iniciales del incremento. A medida que avanza, se van entregando versiones intermedias del sistema. Estas secuencias iterativas se repetirán hasta alcanzar un producto que satisfaga las exigencias del cliente.

Es relevante destacar que el enfoque de desarrollo incremental sirve de base para la mayoría de las metodologías ágiles de desarrollo de software. Es de importancia que los estudiantes que se encuentran en proceso de aprender el desarrollo de sistemas empleen esta metodología antes de recurrir directamente a enfoques ágiles, como una experiencia previa. Uno de los beneficios notables de la adopción de esta metodología es que los costos asociados con cambios en los requisitos, que son más significativos en metodologías tradicionales y rígidas como el enfoque en cascada, resultan mucho menores y se pueden incorporar en cada una de las etapas iterativas del sistema.

Comparado con la metodología en cascada, una ventaja evidente es la mayor facilidad para obtener retroalimentación del cliente. Esto se debe a que, durante la implementación de los incrementos, el usuario tiene la oportunidad de interactuar con el sistema en las versiones parciales y ofrecer comentarios y sugerencias antes de avanzar con los siguientes incrementos. Esto se

traduce en un sistema más elaborado y con un alto nivel de aceptación, tal como lo menciona (Sommerville, 2011).

El progreso gradual en la ampliación de funcionalidades de un sistema, en el contexto de la metodología incremental, implica que el líder del proyecto debe efectuar revisiones después de cada iteración. Esta revisión tiene como fin evaluar si el avance está en línea con el plan establecido o si requiere ajustes para cumplir con la meta principal del proyecto. Además, esta evaluación permite verificar si el cronograma planificado se ajusta a los avances logrados en cada incremento y si el proyecto general sigue su curso normalmente.

No obstante, surgen ciertos desafíos al emplear la metodología incremental. Uno de ellos es la tarea compleja de identificar los requisitos comunes entre los distintos incrementos que se llevarán a cabo durante el desarrollo del sistema. Esto puede llevar a duplicar esfuerzos o definir de manera superficial algunos de los requerimientos esenciales de la aplicación. Cuando se trata de reemplazar un sistema antiguo por uno nuevo, no se aconseja utilizar la metodología incremental. Esto se debe a que los usuarios podrían intentar aplicar todas las funcionalidades tan pronto como los primeros incrementos se presenten, lo que podría generar resistencia e insatisfacción con la nueva propuesta.

En muchas empresas u organizaciones, al momento de establecer contratos, se requiere especificar de manera anticipada y con un alto nivel de detalle las características y funciones del sistema. Sin embargo, en la metodología incremental, los detalles de las especificaciones se desarrollan a medida que avanzan los

incrementos del software. Por lo tanto, adaptar una nueva forma de contratación se vuelve complicado, especialmente en instituciones gubernamentales (Sommerville, 2011).

Desventajas del desarrollo incremental

El enfoque de desarrollo incremental, como cualquier modelo, presenta ventajas y desventajas, las cuales dependen del proyecto, el contexto, el equipo de trabajo, los acuerdos contractuales y otros factores. Hasta ahora, he resaltado los aspectos positivos del modelo de manera general, pero ninguna metodología es completamente infalible.

Un desafío principal se origina en la existencia de procedimientos burocráticos arraigados en las grandes organizaciones, lo que puede generar descoordinación entre estos procesos y un enfoque iterativo o ágil más flexible.

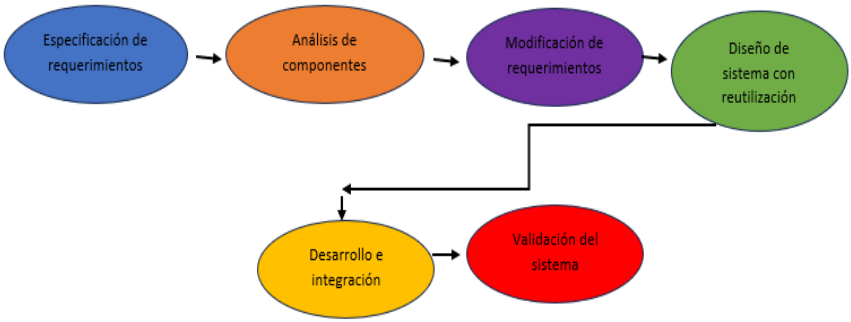
Los inconvenientes del desarrollo incremental se hacen más evidentes en sistemas amplios, complejos y de largo plazo, donde equipos diversos trabajan en distintas partes del sistema. Para sistemas de gran envergadura, es fundamental contar con una estructura arquitectónica sólida y definir con claridad las responsabilidades de los equipos involucrados en las diferentes áreas del sistema. Estas cuestiones deben ser planeadas de antemano en lugar de evolucionar de manera incremental.

A pesar de lo mencionado anteriormente, el concepto de presentar una versión inicial del producto con las funcionalida-

des esenciales y luego mejorarla progresivamente sigue siendo válido en el desarrollo incremental. Sin embargo, esta metodología no siempre es factible, ya que la implementación de nuevo software en un entorno de producción real puede afectar los procesos empresariales normales (Costanzo, 2023).

3.3.3 Ingeniería de software orientado a la reutilización y otros

Figura 10.Ingeniería de software orientada a la reutilización



Nota. Se presenta en la figura 10 indica un esquema que ilustra el modelo típico del proceso de desarrollo basado en reutilización.

En la mayoría de los proyectos de desarrollo de software, se produce cierto grado de reutilización de componentes de software. Este fenómeno ocurre de manera frecuente en un contexto informal, donde los profesionales involucrados en el proyecto identifican diseños o códigos similares a lo que se necesita y pro-

ceden a buscarlos, adaptarlos según sea necesario e integrarlos en sus sistemas.

Este tipo de reutilización no está vinculado a un proceso de desarrollo particular. Sin embargo, en el siglo XXI, los enfoques de desarrollo de software que hacen hincapié en la reutilización de componentes existentes son ampliamente adoptados. Estos enfoques se basan en una extensa colección de componentes de software reutilizables y en la incorporación de marcos de trabajo para ensamblar estos componentes. En algunas ocasiones, estos componentes pueden ser sistemas completos en sí mismos, como software comercial listo para usar (COTS), que mejoran funciones específicas, como procesadores de texto o hojas de cálculo.

Aunque las etapas iniciales de definición de requisitos y validación se asemejan a otros procesos de desarrollo de software en un enfoque orientado a la reutilización, las fases intermedias muestran notables distinciones.

En la investigación de Whitten et al. (2003), estas etapas comprenden:

Análisis de componentes: Una vez se cuenta con la descripción de los requisitos, se procede a buscar componentes que se ajusten a esa descripción para llevar a cabo la implementación. En la mayoría de los casos, no se encuentra una correspondencia exacta, y los componentes empleados solo abarcan una porción de la funcionalidad necesitada.

Modificación de requerimientos: En este punto del proceso, se examinan los requisitos empleando los detalles de los componentes identificados. Después, se ajustan para adecuarse a los componentes que están disponibles. En situaciones donde no es factible llevar a cabo las modificaciones necesarias, existe la posibilidad de volver a la fase de análisis de componentes para explorar en busca de enfoques alternativos.

Diseño de sistema con reutilización: En esta fase, se desarrolla la estructura conceptual del sistema o se hace uso de un marco conceptual existente. Los diseñadores analizan los componentes que están siendo reutilizados y ajustan el marco de trabajo en consecuencia. En casos en los que los componentes reutilizables no estén disponibles, es probable que sea necesario crear nuevo software para llenar esas brechas.

Desarrollo e integración: Luego, se procede con la creación del software que no puede ser obtenido de fuentes externas y se realiza la combinación de componentes y sistemas COTS para construir el nuevo sistema. En este enfoque, la integración del sistema puede estar incorporada en el proceso de desarrollo en lugar de ser una etapa separada y autónoma.

Existen tres categorías de elementos de software que pueden ser utilizados en un enfoque de reutilización en el desarrollo:

1. Servicios Web que se construyen de acuerdo con estándares específicos y se encuentran disponibles para ser llamados de forma remota.

2. Conjuntos de elementos que se crean como una entidad para ser incorporados en un entorno de componentes como .NET o J2EE.

3. Sistemas de software autónomos que se configuran para su utilización en un entorno específico.

La ingeniería de software centrada en la reutilización ofrece una ventaja evidente al reducir la necesidad de crear software nuevo, lo que a su vez conlleva una reducción en los costos y los riesgos. En general, también tiende a acelerar la entrega del software. No obstante, es inevitable que se deban hacer concesiones en algunos requisitos, lo que podría resultar en un sistema que no cumpla completamente con las necesidades de los usuarios. Además, existe una pérdida de control sobre la evolución del sistema, dado que las nuevas versiones de los componentes reutilizables no están bajo el control de la organización que los emplea (Sommerville, 2009).

Autoevaluación 11

1. **¿Qué caracteriza al modelo en cascada en el desarrollo de software?**
    a. Un enfoque de desarrollo paralelo.
    b. Un flujo continuo de pasos secuenciales.
    c. La retroalimentación constante con el cliente.
    d. La ausencia de fases de desarrollo.
2. **¿Quién es comúnmente atribuido como el autor de la primera descripción formal del modelo en cascada?**
    a. Winston Royce.
    b. Alan Turing.
    c. Richard Stallman.
    d. Linus Torvalds.
3. **¿Cómo se caracterizan las fases en el modelo en cascada en términos de secuencia temporal?**
    1. Se pueden llevar a cabo simultáneamente.
    2. Cada fase comienza antes de completar la fase anterior.
    3. No existe una secuencia lineal.
    4. Las fases no están relacionadas entre sí.
4. **¿Qué sucede en el modelo en cascada cuando se detecta un error en una fase particular?**
    a. Se ignora el error y se procede a la siguiente fase.
    b. Se retrocede hasta la fase inicial de análisis de requisitos del sistema.
    c. Se suspende el proyecto indefinidamente.
    d. Se comunica el error al cliente sin hacer correcciones.

5. **¿Cuál es uno de los propósitos de la fase de Pruebas en el modelo en cascada?**
    1. Evaluar la viabilidad económica del software.
    2. Transformar el diseño en un formato comprensible por la máquina.
    3. Identificar fallos y verificar que los resultados coincidan con los esperados.
    4. Realizar modificaciones en el software existente.
6. **¿Qué caracteriza al desarrollo incremental en el desarrollo de software?**
    1. Un enfoque rígido y secuencial.
    2. La elaboración de una versión definitiva desde el principio.
    3. La entrega de versiones progresivamente más avanzadas del sistema.
    4. La falta de retroalimentación del cliente.
7. **¿Cuál es una ventaja significativa del desarrollo incremental en comparación con el modelo en cascada?**
    a. Mayor rigidez en la gestión del proyecto.
    b. Facilita la obtención de retroalimentación del cliente.
    c. Requiere menos planificación inicial.
    d. Mayor costo asociado a cambios en los requisitos.
8. **¿Cuál es uno de los desafíos del desarrollo incremental mencionados en el texto?**
    1. Identificar los requisitos comunes entre los incrementos.
    2. Generar resistencia e insatisfacción con la nueva propuesta.
    3. Aplicar todas las funcionalidades tan pronto como los primeros incrementos se presenten.
    4. Adaptar una nueva forma de contratación en instituciones gubernamentales.

9. ¿En qué tipo de proyectos el desarrollo incremental puede ser menos adecuado?

a. En proyectos con un equipo diverso trabajando en distintas partes del sistema.
b. En proyectos pequeños y simples.
c. En proyectos donde no es necesario contar con retroalimentación del cliente.
d. En proyectos que no involucran software.

10. ¿Qué aspecto es fundamental para el éxito del desarrollo incremental en sistemas de gran envergadura?

1. No es necesario definir estructuras arquitectónicas sólidas.
2. Contar con un enfoque completamente burocrático.
3. Planificar de antemano las responsabilidades de los equipos involucrados.
4. Evolucionar de manera incremental en lugar de planificar.

11. ¿Qué caracteriza la reutilización de componentes de software en un contexto informal?

a. Se basa en un proceso de desarrollo específico.
b. Se enfoca en la creación de componentes nuevos desde cero.
c. Los profesionales buscan y adaptan diseños o códigos existentes.
d. Solo se aplica a proyectos pequeños.

12. ¿Cuál de las siguientes afirmaciones es cierta sobre los enfoques de desarrollo de software orientados a la reutilización?

1. No se basan en componentes de software reutilizables.
2. Utilizan exclusivamente componentes COTS.
3. Incorporan marcos de trabajo para ensamblar componentes reutilizables.
4. Se centran en la creación de software completamente nuevo.

13. ¿Cuál es una de las etapas intermedias en un enfoque orientado a la reutilización según el texto?

a. Definición de requisitos.
b. Análisis de componentes.
c. Validación.
d. Planificación del proyecto.

14. ¿Cuál es uno de los beneficios de la ingeniería de software centrada en la reutilización?

1. Reducción de costos y riesgos.
2. Pérdida de control sobre la evolución del sistema.
3. Mayor necesidad de crear software nuevo.
4. Retraso en la entrega del software.

15. ¿Qué categoría de elementos de software se menciona como parte de un enfoque de reutilización en el desarrollo?

a. Hardware incorporado en el sistema.
b. Sistemas de software autónomos.
c. Documentación técnica detallada.
d. Modelos de negocio.

3.4 Proceso de desarrollo de software

3.4.1 Introducción

El proceso de elaboración de software constituye una secuencia planificada y organizada de acciones y pasos destinados a concebir, diseñar, construir, evaluar, aplicar y mantener software de alto nivel. Este procedimiento está estructurado para dirigir de manera eficaz y eficiente la generación de soluciones informáticas que satisfagan los requerimientos del proyecto y las expectativas de los usuarios (Sulbarán, 2018).

La introducción al procedimiento de desarrollo de software conlleva la comprensión de su importancia en la creación de software funcional y confiable. En un contexto cada vez más tecnológico y orientado a la automatización, el proceso de desarrollo de software proporciona un marco organizativo para abordar las complejidades y desafíos inherentes a la construcción de aplicaciones y sistemas informáticos. Desde la etapa inicial de planificación hasta la ejecución y el mantenimiento continuo, este método orienta a los equipos de desarrollo a través de fases cruciales, incluyendo la definición de requisitos, el diseño, la programación, las pruebas y la entrega definitiva.

Esta presentación introductoria al proceso de desarrollo de software también hace hincapié en la necesidad de adaptarse a diferentes enfoques y metodologías según las particularidades del proyecto y las preferencias del equipo. Cada modelo de desa-

rrollo, ya sea el en cascada, el incremental, el ágil u otros, posee sus propias ventajas y restricciones, lo que pone de relieve la flexibilidad que se requiere en la gestión de proyectos de software. Además, este inicio destaca la importancia de establecer normas y buenas prácticas para asegurar la calidad del software resultante y la satisfacción de los usuarios finales.

En resumen, la introducción al proceso de desarrollo de software es fundamental para comprender la estructura y la finalidad de las actividades involucradas en la creación exitosa de software. Este procedimiento no solo aborda la fase técnica del desarrollo, sino que también contempla aspectos como la colaboración con los usuarios, la gestión de riesgos y la planificación de recursos. Como resultado, proporciona un marco sólido para alcanzar productos de software eficaces, confiables y alineados con las necesidades del mercado y los clientes (Martinez, 2015).

3.4.2 Estándares de desarrollo

Los estándares de desarrollo dentro del proceso de desarrollo de software comprenden conjuntos de directrices, prácticas y reglas instaurados con el propósito de asegurar la excelencia, uniformidad y eficacia en todas las fases del ciclo de vida del software. Estos estándares desempeñan un papel fundamental en orientar a los equipos de desarrollo y garantizar que el software elaborado cumpla con las condiciones establecidas por el proyecto, además de ser confiable, fácilmente mantenible y seguro (Martinez, 2015).

Algunos estándares en el proceso de desarrollo de software son:

Documentación: Establecimiento de estructuras y contenidos para la elaboración de documentación que abarca requisitos, especificaciones, diseño y guías de usuario.

Código fuente: Creación de normativas de diseño, convenciones de denominación y prácticas recomendadas para el código fuente del software.

Pruebas: Establecimiento de protocolos y enfoques para las pruebas, que engloban casos de prueba, criterios de aprobación y sistemas para registrar los resultados.

Gestión de configuración: Creación de flujos de trabajo para supervisar versiones, administrar modificaciones y monitorear la configuración del software.

Seguridad: Aplicación de lineamientos para enfrentar aspectos de seguridad, tales como la salvaguardia de información, la verificación de identidad y la prevención de debilidades en la protección.

Metodologías y modelos de desarrollo: Aplicación de estructuras como Scrum, Kanban, en cascada, desarrollo ágil, y otras similares, las cuales definen prácticas particulares para cada fase del procedimiento.

Cumplimiento normativo: Garantía de que el software se ajuste a las normativas y estándares particulares correspondientes al sector o al ámbito en el que se implemente.

Interoperabilidad: Establecimiento de normas y procedimientos con el propósito de asegurar una comunicación eficaz entre diversos sistemas y plataformas.

Documentación de usuarios: Elaboración de manuales y orientaciones para los usuarios, adhiriéndose a estructuras y enfoques predefinidos**.**

Auditoría y revisión: Creación de procedimientos para inspeccionar y confirmar la calidad del software durante su evolución.

La aplicación de estándares de desarrollo en el proceso de crear software ayuda a reducir fallos, fomentar la colaboración entre los integrantes del equipo, simplificar la comunicación con las partes interesadas y asegurar que el producto terminado sea seguro y cumpla con las anticipaciones. Estos estándares pueden ser adaptados a la organización en particular o estar fundados en regulaciones industriales ampliamente reconocidas, como las ISO/IEC 12207 para los procedimientos en el ciclo de vida del software o las ISO/IEC 25010 para medir la calidad del software (Martinez, 2015).

3.5 Conclusión

En conclusión, dentro de este capítulo hemos hablado de las metodologías de desarrollo de software, se han indagado descripciones que abarcan la totalidad del proceso, desde la concepción hasta la implementación de programas utilizando diferentes len-

guajes de programación. Estas metodologías presentan características particulares que guían su eficacia en proyectos. Han sido agrupadas en categorías de estructuradas, orientadas a objetos y ágiles, cada una con enfoques propios. Además, se han analizado detalladamente los ciclos de vida del desarrollo de software y se ha explorado en profundidad el concepto de paradigmas de proceso, que proporcionan marcos conceptuales para dirigir los proyectos.

Dentro de la categorización de los paradigmas de proceso, se han examinado el modelo en cascada con sus etapas secuenciales, el enfoque incremental basado en la liberación gradual de versiones, y la ingeniería de software orientada a la reutilización de componentes previamente construidos. También se han considerado otras aproximaciones. Se ha subrayado la importancia de establecer estándares en el proceso de desarrollo de software para garantizar la calidad y la confiabilidad de los productos resultantes. En resumen, la exploración de metodologías, definiciones y paradigmas de proceso en el desarrollo de software brinda una visión amplia y variada de los enfoques que afectan la gestión de proyectos, los ciclos de vida y la creación de software eficaz y confiable.

Autoevaluación 12

1. **¿Cuál es el objetivo principal del proceso de elaboración de software según el texto?**
   a. Generar software de bajo nivel.
   b. Concebir ideas para proyectos de software.
   c. Generar soluciones informáticas satisfaciendo requerimientos y expectativas.
   d. Evaluar aplicaciones existentes.
2. **¿Qué importancia tiene la introducción al proceso de desarrollo de software?**
   1. Ninguna importancia.
   2. Proporciona un marco organizativo para abordar desafíos tecnológicos.
   3. Define los requisitos técnicos de un proyecto.
   4. Establece las tareas de mantenimiento del software.
3. **¿Cuáles son algunas de las fases cruciales mencionadas en el texto?**
   a. Comunicación con usuarios y revisión de documentación.
   b. Diseño, pruebas y entrega definitiva.
   c. Comercialización y ventas.
   d. Reuniones de equipo y administración de recursos.
4. **¿Por qué es importante la flexibilidad en la gestión de proyectos de software?**
   1. Para evitar cualquier cambio en el proyecto.
   2. Para adaptarse a diferentes enfoques y metodologías.
   3. Para imponer un único modelo de desarrollo.
   4. Para acelerar el proceso de desarrollo.

5. **¿Cuál de los siguientes NO es un modelo de desarrollo de software mencionado en el texto?**
   a. En cascada.
   b. Incremental.
   c. Ágil.
   d. Continuo.
6. **¿Qué aspectos son contemplados en el proceso de desarrollo de software según el texto?**
   1. Colaboración con usuarios y gestión de riesgos.
   2. Administración financiera y ventas.
   3. Diseño gráfico y publicidad.
   4. Estándares de seguridad informática.
7. **¿Cuál es el propósito principal de establecer normas y buenas prácticas en el desarrollo de software?**
   1. Aumentar la complejidad del proyecto.
   2. Garantizar la satisfacción de los desarrolladores.
   3. Asegurar la calidad del software y la satisfacción de los usuarios finales.
   4. Acelerar el proceso de desarrollo.
8. **¿Qué tipo de productos de software busca alcanzar el proceso de desarrollo mencionado en el texto?**
   a. Productos económicos.
   b. Productos de entretenimiento.
   c. Productos eficaces, confiables y alineados con las necesidades del mercado y los clientes.
   d. Productos de alta complejidad.

9. ¿Cuál es uno de los elementos destacados en la introducción al proceso de desarrollo de software?

1. La exclusión de los usuarios en el proceso.
2. La importancia de evitar la planificación.
3. La gestión de recursos humanos.
4. La necesidad de adaptarse a diferentes enfoques.

10. ¿Qué importancia tiene la colaboración con los usuarios en el proceso de desarrollo de software?

a. Ninguna importancia.
b. Facilita la planificación del proyecto.
c. Ayuda a definir los requisitos y expectativas.
d. Retrasa el proceso de desarrollo.

11. ¿Qué función desempeñan los estándares de desarrollo en el proceso de desarrollo de software?

1. Establecer criterios de diseño.
2. Asegurar la seguridad del hardware.
3. Garantizar la calidad y eficacia en todas las fases del ciclo de vida del software.
4. Facilitar la comunicación con clientes.

12. ¿Qué aspectos abarcan los estándares de desarrollo relacionados con la documentación?

1. Especificaciones y guías de usuario.
2. Códigos de programación.
3. Pruebas de rendimiento.
4. Reglas de seguridad.

13. ¿Cuál es uno de los propósitos de los estándares de desarrollo en relación con la gestión de configuración?

a. Definir casos de prueba.
b. Supervisar versiones y gestionar modificaciones.
c. Crear flujos de trabajo para pruebas.
d. Prevenir debilidades en la protección.

14. ¿Qué tipo de estructuras y enfoques definen las metodologías y modelos de desarrollo mencionados en el texto?

1. Estructuras de diseño.
2. Pruebas de seguridad.
3. Prácticas particulares para cada fase del procedimiento.
4. Convenciones de denominación.

15. ¿Cómo contribuyen los estándares de desarrollo a la creación exitosa de software?

a. Reduciendo fallos y fomentando la colaboración.
b. Aumentando la complejidad del proyecto.
c. Limitando la comunicación con las partes interesadas.
d. Ignorando los estándares industriales reconocidos.

Referencias

Bournissen, J.M. (1999). La evolución del software. *Enfoques, XI*(1 y 2), 123-140.

Costanzo, M. (s.f). *Desarrollo incremental* [Blog]. https://mauricio.mwebs.com.uy/blog/qué-es-el-desarrollo-incremental/23

León Yacelga, A.R., Acosta Espinoza, J.L., & Díaz Vásquez, R.A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Universidad Y Sociedad, 13*(5), 175-182. https://rus.ucf.edu.cu/index.php/rus/article/view/2223

Martinez, R.N. (2015). *El Proceso de Desarrollo de Software*. IT Campus Academy.

Maida, E.G., y Pacienzia, J. (2015). *Metodologías de desarrollo de software* [Tesis Licenciatura, Universidad Católica Argentina]

Santander Universidades. (2020, 21 de diciembre). *Metodologías de desarrollo de software: ¿qué son?* https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

Sommerville, I. (2011). *Ingeniería de Software*. Pearson Edcucación.

Sulbarán, H. (2014, 24 de septiembre). *Paradigmas en el desarrollo de software* [Blog]. https://acortar.link/VUTmOL

Sulbarán, I.. (2023, 27 de abril). Interfaz de usuario (ui): ejemplos y tipos. *Tiffin University*. https://global.tiffin.edu/noticias/interfaz-de-usuario-ui-ejemplos-y-tipos

§

Software development methodologies and processes
Metodologias e processos de desenvolvimento de software

Mónica Elizabeth Páez Padilla
Instituto de Educación Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0006-1030-1394
monica.paez@intsuperior.edu.ec

Abstract
This chapter focuses on the essential relevance of software development methodologies and processes to achieve reliable and high quality programs. Its central objective is to explore in detail a range of methodologies and paradigms that guide this process, outlining their fundamental characteristics and their classification into categories such as structured, object-oriented and agile. In addition, the chapter explores software development life cycles, the paradigms that guide project planning and execution, and classifies common process paradigms. Finally, it discusses the importance of the development process and the role of standards in ensuring consistency and quality in software projects. In summary, the chapter provides a solid understanding of the methodologies, paradigms, and processes that shape software creation in the software engineering industry.
Keywords: software development, projects, planning.

Resumo
Este capítulo se concentra na relevância essencial das metodologias e dos processos de desenvolvimento de software para obter software confiável e de alta qualidade. Seu objetivo central é explorar em detalhes uma série de metodologias e paradigmas que orientam esse processo, delineando suas características fundamentais e sua classificação em categorias como estruturada, orientada a objetos e ágil. Além disso, o capítulo explora os ciclos de vida do desenvolvimento de software, os paradigmas que orientam o planejamento e a execução do projeto e classifica os paradigmas de processos comuns. Por fim, ele discute a importância do processo de desenvolvimento e o papel dos padrões para garantir a consistência e a qualidade dos projetos de software. Em resumo, o capítulo oferece uma sólida compreensão das metodologias, dos paradigmas e dos processos que moldam a criação de software no setor de engenharia de software.
Palavras-chave: desenvolvimento de software, projetos, planejamento.