Unidad 1

Desarrollo de Software

Contenidos

- Parte 1: ¿Cómo se crea el software que usamos a diario? 3
- Parte 2: Lenguajes de Programación 6
- Parte 3:Ejecución de programas. Del código fuente al ejecutable 10
- Parte 4:Desarrollo de Aplicaciones 14
- Parte 5: Metodologías Ágiles en el Desarrollo de Software 18

Parte 1: ¿Cómo se crea el software que usamos a diario?

1. Objetivos de Aprendizaje

- Comprender qué es un programa informático y qué es el software.
- Identificar los diferentes tipos de software y sus aplicaciones.
- Aplicar el conocimiento adquirido para seleccionar software adecuado para un equipo informático.

2. Introducción al Software y los Programas Informáticos

En la actualidad, vivimos en una era completamente digital donde la información es un recurso clave. Las computadoras están en el centro de este ecosistema, y lo que las hace realmente útiles es el **software**: ese conjunto de instrucciones que permite que los sistemas realicen tareas específicas.

Para comprender qué es el software, es útil explorar definiciones respaldadas por organismos internacionales de estándares:

- **IEEE** (Instituto de Ingenieros Eléctricos y Electrónicos): Principal organización profesional que impulsa el desarrollo tecnológico.
- **ISO** (Organización Internacional de Normalización): Crea estándares internacionales para facilitar la interoperabilidad y calidad.
- **IEC** (Comisión Electrotécnica Internacional): Publica estándares relacionados con tecnologías eléctricas y electrónicas.

3. Definiciones: Programa Informático y Software

¿Qué es un programa informático?

Un programa informático, según el estándar **IEEE/ISO/IEC 24765:2017**, es:

1. Un conjunto de instrucciones y datos para realizar tareas específicas.

- **Ejemplo:** El proceso de arranque de una computadora usa un programa que carga el sistema operativo. Otro caso es un reproductor de música que contiene instrucciones para reproducir canciones y datos para representarlas.

2. Una solución a problemas mediante un lenguaje de programación.

- **Ejemplo:** Ordenar una lista de nombres en orden alfabético usando Python es un problema resuelto mediante un algoritmo de ordenación.

Analogía: Piensa en un programa como una receta de cocina:

- Los **ingredientes** representan los datos (lechuga, tomate, pepino).
- Las **instrucciones** son los pasos para hacer el plato (lavar, cortar, mezclar).Al ejecutar los pasos, obtienes el resultado final, en este caso, una ensalada.

¿Qué es el software?

El software incluye:

- **Programas:** Instrucciones para que la computadora realice acciones.
- **Procedimientos:** Métodos necesarios para ejecutar los programas.
- **Documentación:** Guías y manuales de uso.

El software opera junto con el hardware para crear un sistema informático funcional. No olvidemos a las personas que usan y desarrollan estos sistemas, conocidas como **peopleware**.

4. Tipos de Software

Software de Sistema

- Gestiona los recursos básicos de la computadora.**Ejemplos:**
  - Sistemas operativos: Windows, macOS, Linux.
  - Utilidades: Programas de limpieza de disco, antivirus.
  - Drivers: Controladores para impresoras o tarjetas gráficas.

Software de Aplicación

- Ayuda a realizar tareas específicas.**Ejemplos:**
  - Microsoft Word para escribir documentos.
  - Photoshop para edición de imágenes.
  - GIMP y LibreOffice como alternativas gratuitas de código abierto.

Software de Soporte

- Herramientas utilizadas por desarrolladores para crear otros programas.**Ejemplos:**
  - **Compiladores:** Transforman código fuente en código máquina (GCC).
  - **Intérpretes:** Ejecutan código línea por línea (Python).
  - **Editores de texto:** Visual Studio Code, Sublime Text.

Ejemplo por profesión:

- Científicos de datos: Jupyter Notebook.
- Desarrolladores web: Visual Studio Code.

Ejercicio Práctico: Configurando tu Equipo

Imagina que necesitas un nuevo equipo para un curso de informática. Responde estas preguntas:

1. Elige un sistema operativo.

- Justifica tu elección según tus necesidades (Windows, macOS, Linux).

2. Instala software de aplicación.

- Ejemplo: Word para documentos, LibreOffice si prefieres alternativas gratuitas.

3. Incluye software de soporte.

- Ejemplo: Visual Studio Code para desarrollo web, Eclipse para proyectos en Java.

Test Multirespuesta

1. ¿Qué es un programa informático?

a) Un conjunto de datos y reglas para operar hardware.
b) Instrucciones y datos que permiten realizar tareas específicas.
c) Documentos necesarios para usar aplicaciones.
d) Todas las anteriores.

2. ¿Cuál de estos NO es un tipo de software?

a) Software de soporte.
b) Software de sistema.
c) Software de hardware.
d) Software de aplicación.

3. ¿Qué herramienta usarías para desarrollar en Python?

a) Visual Studio Code.
b) Photoshop.
c) LibreOffice.
d) IntelliJ IDEA.

Ejercicio Guiado

1. Selecciona un equipo informático (real o ficticio).
2. Elige un sistema operativo (Windows, macOS, Linux).

- Justifica tu elección considerando tus objetivos (diseño, programación, etc.).

3. Enumera tres aplicaciones esenciales que instalarías.

- Ejemplo: Word, Excel, Visual Studio Code.

4. Agrega al menos una herramienta de soporte para el desarrollo de software.

- Justifica por qué la consideras importante.

Parte 2: Lenguajes de Programación

1. Objetivos de Aprendizaje

- Comprender qué es un lenguaje de programación y su importancia.
- Identificar las diferentes clasificaciones de los lenguajes de programación.
- Diferenciar entre lenguajes de programación y lenguajes informáticos.
- Conocer ejemplos específicos de lenguajes en cada clasificación.
- Aplicar los conocimientos a la elección de un lenguaje adecuado para distintos contextos.

2. Introducción

Imagina que los lenguajes de programación son los idiomas que utilizamos para comunicarnos con las computadoras. Aunque nosotros podemos usar palabras o gráficos, las máquinas solo entienden **ceros y unos**, también conocido como lenguaje binario. Para salvar esta brecha, los programadores crearon lenguajes más cercanos a la lógica humana, que permiten traducir nuestras ideas al formato que las computadoras entienden.

La programación, entonces, es el arte de traducir soluciones de problemas en instrucciones que una máquina puede ejecutar.

3. ¿Qué es un Lenguaje de Programación?

Según el estándar **IEEE/ISO/IEC 24765:2017**, un lenguaje de programación es un conjunto de reglas y palabras que los programadores usan para crear instrucciones comprensibles por una computadora.

Diferencia entre “Lenguaje de Programación” y “Lenguaje Informático”

- **Lenguaje de Programación**: Diseñado para escribir programas que ejecuten tareas específicas. Ejemplos: Python, Java.
- **Lenguaje Informático**: Más amplio, incluye lenguajes para estructurar o transportar datos. Ejemplos: XML, YAML.

4. Clasificación de los Lenguajes de Programación

Por Nivel de Abstracción

1. Bajo Nivel

- Muy cercano al hardware. Ejemplo: Ensamblador.
- **Ventaja**: Control total del hardware.
- **Desventaja**: Difícil de aprender.
- **Usos**: Sistemas embebidos, controladores.

2. Nivel Intermedio

- Mezcla características de alto y bajo nivel. Ejemplo: C.
- **Ventaja**: Equilibrio entre control y facilidad.
- **Usos**: Sistemas operativos, software de alto rendimiento.

3. Alto Nivel

- Similar al lenguaje humano. Ejemplo: Python.
- **Ventaja**: Fáciles de aprender.
- **Desventaja**: Menor control sobre el hardware.
- **Usos**: Aplicaciones web, software científico.

Por Propósito

1. Propósito General:

- Ejemplo:
  - Python: Análisis de datos, desarrollo web.
  - JavaScript: Interactividad web.
  - Java: Aplicaciones empresariales.

2. Propósito Específico:

- Ejemplo:
  - R: Estadística y análisis de datos.
  - SQL: Gestión de bases de datos.

Por Evolución Histórica

1. 1GL (Primera Generación): Código máquina.

- **Ejemplo**: Programación de microcontroladores.

2. 2GL (Segunda Generación): Ensamblador.

- **Ejemplo**: Sistemas embebidos.

3. 3GL (Tercera Generación): Lenguajes modernos.

- **Ejemplo**: C++, Java.

4. 4GL (Cuarta Generación): Lenguajes para grandes volúmenes de datos.

- **Ejemplo**: SQL.

5. 5GL (Quinta Generación): Resolución de problemas complejos.

- **Ejemplo**: Prolog.

Por Forma de Ejecución

1. Compilados: Traducen todo el código antes de ejecutarse. Ejemplo: C.
2. Interpretados: Ejecutan el código línea por línea. Ejemplo: Python.
3. Mixtos: Compilan a un bytecode que luego es interpretado. Ejemplo: Java.

Por Estilo o Paradigma de Programación

1. Imperativa:

- Procedimental: Divide problemas en subprogramas. Ejemplo: C.
- Orientada a objetos: Usa objetos y clases. Ejemplo: Java.

2. Declarativa:

- Funcional: Usa funciones puras. Ejemplo: Haskell.
- Lógica: Usa reglas y hechos. Ejemplo: Prolog.

Por Lugar de Ejecución

1. Frontend: Ejecutado en el navegador. Ejemplo: JavaScript.
2. Backend: Ejecutado en el servidor. Ejemplo: PHP.
3. Full-Stack: Combina frontend y backend.

5. Apéndice

Definiciones Clave (IEEE/ISO/IEC 24765:2017)

- **Programa Informático**: Instrucciones que permiten realizar funciones computacionales.
- **Software**: Incluye programas, procedimientos y documentación.

6. Conclusión

Los lenguajes de programación son la base del desarrollo de software y permiten resolver problemas de manera eficiente. Al comprender sus clasificaciones y características, puedes elegir el lenguaje más adecuado para tus necesidades.

Test Multirespuesta

1. ¿Qué tipo de lenguaje es Ensamblador?

a) Bajo nivel
b) Intermedio
c) Alto nivel
d) Funcional

Respuesta: a) Bajo nivel

2. ¿Cuál es un lenguaje interpretado?

a) Java
b) C++
c) Python
d) SQL

Respuesta: c) Python

3. ¿Qué paradigma utiliza objetos y clases?

a) Procedimental
b) Funcional
c) Declarativo
d) Orientado a objetos

Respuesta: d) Orientado a objetos

Ejercicio Guiado

1. Escenario: Eres parte de un equipo de desarrollo y debes elegir un lenguaje de programación para:

- Crear una aplicación móvil.
- Desarrollar un sistema de gestión empresarial.
- Diseñar una página web interactiva.

2. Instrucciones:

- Investiga qué lenguajes son adecuados para cada tarea (por ejemplo, Java para móviles, Python para sistemas empresariales, JavaScript para páginas web).
- Justifica tu elección considerando ventajas y limitaciones.

Parte 3:Ejecución de programas. Del código fuente al ejecutable

1. Objetivos de Aprendizaje

- Comprender el proceso de traducción de código fuente a código ejecutable.
- Diferenciar entre compiladores e intérpretes.
- Identificar las fases del ciclo de compilación y ejecución.
- Conocer las tecnologías de virtualización y su relevancia.
- Entender el concepto de multiplataforma y su importancia en el desarrollo de software.

2. Introducción

Cuando los desarrolladores escriben programas, utilizan lenguajes de alto nivel como Python, Java o C++, que son fáciles de entender para los humanos pero no para las máquinas. Los ordenadores, por el contrario, solo entienden lenguaje máquina (ceros y unos). Para salvar esta brecha, se utilizan **traductores** que convierten el código fuente escrito por los programadores en un formato que las máquinas pueden ejecutar.

El proceso de traducir código fuente a un programa ejecutable es fundamental para que las computadoras realicen tareas, y esta unidad explica cada paso en detalle.

3. Traductores, Compiladores e Intérpretes

Traductores

Los traductores son programas que convierten lenguajes de programación de alto nivel en código máquina, haciéndolo comprensible para el hardware.

Compiladores

- **Función**: Traducen el programa completo antes de su ejecución.
- **Ejemplos**: C, C++.
- **Ventajas**:
  - Detectan errores antes de la ejecución.
  - El ejecutable resultante es más rápido al ejecutarse.
- **Desventajas**:
  - El proceso de compilación puede ser lento.

Intérpretes

- **Función**: Traducción y ejecución línea por línea del programa.
- **Ejemplos**: Python, JavaScript.
- **Ventajas**:
  - Ejecución más rápida durante el desarrollo.
  - Ideal para prototipado y pruebas rápidas.
- **Desventajas**:
  - Más lentos en ejecución que los compiladores.
  - Los errores se detectan durante la ejecución.

4. Código Fuente, Código Objeto y Código Ejecutable

Código Fuente

- **Definición**: Instrucciones escritas por un programador en un lenguaje de alto nivel.
- **Ejemplo**: Un archivo con extensión `.java` para Java o `.py` para Python.

Código Objeto

- **Definición**: El resultado de traducir el código fuente a un formato más cercano al lenguaje máquina, pero aún no ejecutable.
- **Ejemplo**: Archivos `.o` o `.obj`.

Código Ejecutable

- **Definición**: El código final que puede ser ejecutado directamente por la máquina.
- **Ejemplo**: Archivos `.exe` en Windows o binarios en Linux.

Proceso de Compilación

1. Edición: Creación del código fuente utilizando un editor de texto o IDE.

2. Traducción: El compilador transforma el código fuente en código objeto.

3. Enlazado: Combina los archivos de código objeto con bibliotecas necesarias para generar el ejecutable.

5. Tecnologías de Virtualización: Java y .NET

Máquinas Virtuales

- **Definición**: Programas que permiten la ejecución de código en un entorno independiente del hardware y sistema operativo subyacentes.

Tipos de Máquinas Virtuales

1. Máquinas Virtuales de Sistema:

- Simulan un hardware completo.
- **Ejemplo**: VMware, VirtualBox.

2. Máquinas Virtuales de Proceso:

- Ejecutan aplicaciones en un entorno independiente del hardware.
- **Ejemplo**: Java Virtual Machine (JVM), Common Language Runtime (CLR).

Ejemplos de Virtualización

1. Java:

- Traduce el código fuente a **bytecode**.
- La **JVM** ejecuta el bytecode en cualquier sistema operativo.

2. .NET:

- Compila el código fuente a **Common Intermediate Language (CIL)**.
- El **CLR** se encarga de ejecutarlo.

6. El Concepto de Multiplataforma

Definición

El software **multiplataforma** es aquel que puede ejecutarse en distintas combinaciones de hardware y sistemas operativos.

Ejemplos de Aplicaciones Multiplataforma

- **LibreOffice**: Alternativa gratuita a Microsoft Office.
- **Mozilla Firefox**: Navegador web.
- **GIMP**: Editor gráfico.

Software Libre vs. Software Propietario

1. Software Libre:

- Permite a los usuarios ejecutar, modificar y redistribuir el código.
- **Ejemplo**: Linux.

2. Software Propietario:

- Solo se distribuye el ejecutable.
- **Ejemplo**: Microsoft Windows.

7. Conclusión

Esta unidad detalla el proceso que transforma el código fuente en programas ejecutables, destacando la importancia de los traductores, las tecnologías de virtualización y el concepto de multiplataforma. Estos conocimientos son clave para entender cómo se ejecuta el software y su relación con los sistemas modernos.

Test Multirespuesta

1. ¿Qué hace un compilador?

a) Traduce y ejecuta código línea por línea.
b) Traduce todo el código antes de ejecutarlo.
c) Traduce el código fuente a un formato binario intermedio.
d) Detecta errores solo durante la ejecución.

2. ¿Qué extensión corresponde a un código objeto?

a) `.exe`
b) `.py`
c) `.o`
d) `.bin`

3. ¿Qué tecnología utiliza bytecode?

a) CLR
b) JVM
c) VMware
d) VirtualBox

Ejercicio Guiado

1. **Escenario: Eres un desarrollador y debes elegir una tecnología para implementar un programa que funcione en Windows, macOS y Linux.**
2. **Instrucciones:**

- Elige entre usar Java o .NET.
- Investiga cómo estas tecnologías manejan la virtualización (JVM vs. CLR).
- Justifica tu elección considerando ventajas como la portabilidad y compatibilidad multiplataforma.

Parte 4:Desarrollo de Aplicaciones

1. Objetivos de Aprendizaje

- Comprender las fases del ciclo de vida del software.
- Identificar y diferenciar los distintos modelos de desarrollo de software.
- Aplicar un modelo de desarrollo a un caso práctico.
- Evaluar las ventajas y desventajas de cada modelo de desarrollo.

2. Introducción

El desarrollo de aplicaciones sigue un proceso estructurado conocido como el **ciclo de vida del software**, que abarca desde la identificación de necesidades hasta el mantenimiento del producto final. Este proceso se adapta según la metodología utilizada, lo que influye en la forma de abordar cada fase y la relación con los clientes.

3. Fases en el Desarrollo de una Aplicación

1. Análisis

En esta fase se identifican y documentan los **requisitos funcionales** y **no funcionales**. Es como hacer una lista de compras detallada antes de empezar a cocinar.

Herramientas comunes:

- **Diagramas de flujo de datos**: Representan cómo se mueve y procesa la información.
    - **Ejemplo:** En un sistema de gestión de pedidos, los diagramas mostrarán cómo un pedido pasa de la recepción al envío.
- **Diagramas de casos de uso y de secuencia**: Muestran las interacciones entre los usuarios y el sistema.
    - **Ejemplo:** Cómo un usuario realiza una compra en una tienda online.
- **Diagramas entidad-relación**: Representan la estructura de la base de datos.
    - **Ejemplo:** Usuarios, pedidos y productos, con sus relaciones.
- **Wireframes y prototipos**: Representaciones visuales de la interfaz de usuario.

2. Diseño

Se define cómo será la aplicación. Es como planear una receta y los pasos a seguir.

Incluye:

- **Arquitectura del sistema**: Describe la estructura del sistema y las interacciones entre componentes.
    - **Ejemplo:** En una aplicación web, la arquitectura puede incluir cliente, servidor, base de datos y API.
- **Diseño de la base de datos**: Define cómo se organizarán y conectarán los datos.
    - **Ejemplo:** Tablas para usuarios, productos y transacciones.
- **Diseño de la interfaz de usuario**: Cómo interactuarán los usuarios con el sistema.

3. Codificación

Es la fase en la que los desarrolladores traducen el diseño en código fuente usando lenguajes de programación adecuados.

- **Ejemplo:** Implementar una funcionalidad de registro de usuarios.

4. Pruebas

Se verifican y validan las funcionalidades del sistema.

Tipos de pruebas:

- **Unitarias:** Prueban componentes individuales.
- **De integración:** Evalúan cómo interactúan diferentes partes del sistema.
- **De aceptación:** Aseguran que el sistema cumple con los requisitos del usuario.

5. Implantación

Se pone el sistema en producción para que los usuarios finales puedan utilizarlo.

- **Ejemplo:** Publicar una aplicación web en un servidor.

6. Mantenimiento

Es una fase continua para garantizar que el sistema siga funcionando correctamente y cumpla con las necesidades cambiantes.

Tipos de mantenimiento:

- **Correctivo:** Corregir errores.
- **Adaptativo:** Ajustar a nuevas tecnologías.
- **Perfectivo:** Añadir funcionalidades.

4. Modelos de Desarrollo de Software

A. Ciclo de Vida en Cascada

Un enfoque secuencial donde cada fase debe completarse antes de pasar a la siguiente.

- **Ventajas:**
  - Proceso estructurado y controlado.
  - Puntos de control claros.
- **Desventajas:**
  - Poco flexible ante cambios.
  - No permite retroalimentación temprana.

Ejemplo: Seguir una receta paso a paso sin hacer modificaciones.

B. Modelos de Desarrollo Evolutivo

Más flexibles y centrados en iteraciones continuas.

- **Prototipado**:
  - Se crea un prototipo para validar requisitos antes del desarrollo completo.
  - **Ventajas:** Minimiza errores en etapas tempranas.
- **Iterativo e Incremental**:
  - Desarrolla y mejora el sistema en ciclos continuos.
  - **Ventajas:** Permite entregas parciales con funcionalidades.
- **Modelo en Espiral**:
  - Iterativo, con énfasis en la gestión de riesgos.
  - **Ventajas:** Adecuado para proyectos complejos con requisitos cambiantes.

Ejemplo: Desarrollar una red social empezando con funciones básicas y añadiendo nuevas en cada iteración.

5. Caso Práctico: Ciclo de Vida en Espiral

Proyecto: Crear una red social basada en imágenes

1. Análisis:

- Identificar requisitos iniciales como subir fotos y verlas.

2. Diseño:

- Crear diagramas entidad-relación para usuarios, fotos y comentarios.
- Diseñar wireframes de la interfaz.

3. Codificación:

- Implementar funcionalidad básica de subir fotos.

4. Implantación:

- Desplegar la versión inicial y recibir retroalimentación de los usuarios.

5. Mantenimiento y Mejora:

- Incorporar funciones adicionales como comentarios y “me gusta”.

6. Conclusión

El desarrollo de aplicaciones es un proceso estructurado que incluye varias fases, desde el análisis hasta el mantenimiento. Comprender los modelos de desarrollo y sus ventajas y desventajas es esencial para elegir el más adecuado según las necesidades del proyecto. Aplicar conceptos teóricos en casos prácticos, como el modelo en espiral, refuerza el aprendizaje y prepara para escenarios reales.

Test Multirespuesta

1. ¿Cuál de las siguientes no es una fase del ciclo de vida del software?

a) Análisis
b) Codificación
c) Finanzas
d) Mantenimiento

2. ¿Qué modelo de desarrollo es más adecuado para proyectos con requisitos cambiantes?

a) Cascada
b) Modelo en espiral
c) Modelo iterativo
d) Modelo evolutivo

3. ¿Qué tipo de mantenimiento implica añadir nuevas funcionalidades?

a) Correctivo
b) Perfectivo
c) Adaptativo
d) Progresivo

Ejercicio Guiado

Escenario: Eres parte de un equipo que debe desarrollar una aplicación de reservas para un restaurante.

1. Requisitos:

- Los usuarios deben poder reservar mesas y recibir confirmaciones.

2. Instrucciones:

- Diseña un diagrama de flujo para el proceso de reservas.
- Crea un wireframe para la interfaz del usuario.
- Elige un modelo de desarrollo (cascada, iterativo o espiral) y justifica tu elección.
- Implementa una funcionalidad básica de simulación para la reserva de mesas.

Parte 5: Metodologías Ágiles en el Desarrollo de Software

1. Objetivos de Aprendizaje

- Comprender la importancia de las metodologías ágiles en el desarrollo de software.
- Conocer el marco de trabajo Scrum y sus componentes principales.
- Identificar y diferenciar otras metodologías ágiles como Kanban, XP y Lean Development.
- Evaluar las ventajas y desventajas de las metodologías ágiles en diferentes contextos de desarrollo.

2. Introducción

En un mundo en constante cambio, los proyectos de software necesitan ser flexibles y adaptativos. Las **metodologías ágiles** surgieron como respuesta a esta necesidad, ofreciendo ciclos cortos de desarrollo, entregas frecuentes y una comunicación continua con el cliente.

El enfoque ágil se basa en la premisa de que los requisitos pueden cambiar, y los equipos deben estar preparados para adaptarse. Esto permite entregar productos funcionales rápidamente, con retroalimentación constante que guía el desarrollo hacia el resultado esperado.

3. Scrum

Scrum: Un Marco de Trabajo Ágil

Scrum es una de las metodologías ágiles más utilizadas. Organiza los proyectos en **sprints**, intervalos de tiempo definidos (2-4 semanas), en los que el equipo debe entregar un incremento funcional del producto.

Organización del Equipo

Los equipos en Scrum suelen ser pequeños, con roles bien definidos:

1. **Scrum Master: Facilita el proceso de Scrum, elimina impedimentos y asegura que el equipo siga las prácticas ágiles.**
2. **Product Owner: Representa al cliente, prioriza los requisitos en el Product Backlog y asegura que el equipo esté alineado con los objetivos.**
3. **Equipo de Desarrollo: Implementa las funcionalidades del producto.**
    - **Desarrolladores:** Escriben el código.
    - **QA:** Aseguran la calidad mediante pruebas.

Reuniones en Scrum

1. Sprint Planning:

- Se seleccionan tareas del **Product Backlog** y se crean los objetivos del sprint.
- **Duración:** 4 horas.

2. Daily Scrum:

- Reunión breve para discutir el progreso diario y los obstáculos.
- **Duración:** 15 minutos.

3. Sprint Review:

- Se presenta el producto al cliente y se recopila feedback.
- **Duración:** 2 horas.

4. Sprint Retrospective:

- Análisis del sprint, identificando áreas de mejora.
- **Duración:** 1.5 horas.

Ejemplo Práctico de Scrum

Proyecto: Añadir filtros a fotos en una red social.

1. Sprint Planning:

- Crear la interfaz de usuario para aplicar filtros.
- Implementar filtros básicos.
- Realizar pruebas de integración.

2. Daily Scrum:

- Cada miembro comenta su progreso.

3. Sprint Review:

- Se demuestra la funcionalidad a los stakeholders.

4. Sprint Retrospective:

- El equipo evalúa qué funcionó y qué mejorar.

4. Otros Marcos y Metodologías Ágiles

Kanban

- **Descripción**: Método visual para gestionar el trabajo en progreso mediante un tablero con columnas (por hacer, en progreso, hecho).
- **Ventajas**:
  - Visibilidad del flujo de trabajo.
  - Alta flexibilidad.

- **Desventajas**:
  - Menos estructurado para proyectos grandes.
- **Uso recomendado**: Equipos con necesidades de gestión continua de tareas.

Extreme Programming (XP)

- **Descripción**: Enfocado en mejorar la calidad del software mediante prácticas como programación en parejas y revisión continua.
- **Ventajas**:
  - Alta calidad del código.
  - Rápida respuesta a cambios.
- **Desventajas**:
  - Requiere alta colaboración y disciplina.
- **Uso recomendado**: Proyectos que demandan alta calidad y flexibilidad.

Lean Development

- **Descripción**: Basado en principios de manufactura lean, busca maximizar el valor al cliente reduciendo desperdicios.
- **Ventajas**:
  - Entrega rápida de valor.
  - Eficiencia en el uso de recursos.
- **Desventajas**:
  - Difícil implementación en equipos grandes.
- **Uso recomendado**: Proyectos enfocados en maximizar la eficiencia y el valor.

5. Ejercicio Práctico

Instrucciones: Analiza las siguientes situaciones y elige la metodología ágil más adecuada. Justifica tu respuesta.

1. Desarrollo de una Aplicación Móvil para Tareas Diarias

- **Requisitos**: Equipo pequeño, cambios frecuentes basados en feedback de usuarios.
- **Metodología recomendada**: **Scrum**.
  - › **Razón**: Sprints cortos permiten iteraciones rápidas y adaptación a cambios.

2. Mantenimiento de un Sistema de Gestión Escolar

- **Requisitos**: Actualizaciones frecuentes basadas en solicitudes de usuarios.
- **Metodología recomendada**: **Kanban**.
  - › **Razón**: Su enfoque en la gestión continua del trabajo es ideal para mantenimiento.

3. Desarrollo de un Portal Web para un Gran Cliente Corporativo

- **Requisitos**: Equipo grande, funcionalidades complejas, alta calidad del código.
- **Metodología recomendada**: **Extreme Programming (XP)**.
  - **Razón**: Garantiza alta calidad y capacidad de respuesta a cambios complejos.

6. Conclusión

Las metodologías ágiles han revolucionado el desarrollo de software al priorizar la adaptabilidad, la colaboración y la entrega rápida de valor. Desde **Scrum** hasta **Lean Development**, cada enfoque tiene ventajas únicas que lo hacen adecuado para contextos específicos. Entender sus diferencias y aplicarlas correctamente es esencial para maximizar el éxito de los proyectos.

Evolución de las Metodologías de Desarrollo

- **Años 60-70**: Modelos secuenciales y en cascada.
- **Años 80-90**: Introducción de modelos iterativos e incrementales.
- **Años 90-2000**: Desarrollo de metodologías ágiles.
- **Años 2000-presente**: Adopción masiva en software y otras industrias, incluyendo nuevas prácticas como DevOps.

Test Multirespuesta

1. ¿Cuál de las siguientes es una característica de Scrum?

a) Uso de tableros visuales.
b) Sprints con objetivos claros.
c) Programación en parejas.
d) Eliminación de desperdicios.

2. ¿Qué metodología utiliza un tablero para gestionar tareas?

a) Lean Development
b) Scrum
c) Kanban
d) Extreme Programming

3. ¿Cuál es una práctica común en Extreme Programming (XP)?

a) Uso de tableros Kanban.
b) Programación en parejas.
c) Retroalimentación constante.
d) Gestión visual del trabajo.

Ejercicio Guiado

Escenario: Eres parte de un equipo encargado de desarrollar un sistema de reservas online para restaurantes.

1. **Define los roles del equipo bajo Scrum.**
2. **Diseña un tablero Kanban con las tareas iniciales.**
3. **Proporciona un ejemplo de mejora continua siguiendo principios de Lean Development.**