INSTITUTO KHIPU

Semestre III

Sesión 04

PROGRAMA DE ESTUDIOS

DESARROLLO DE SISTEMAS DE INFORMACIÓN

ANÁLISIS Y DISEÑO DE SISTEMAS

Tema:

CICLO DE VIDA DE DESARROLLO DE SOFTWARE - DISEÑO

CICLO DE VIDA DE DESARROLLO DE SOFTWARE - DISEÑO

El ciclo de vida del desarrollo de software (SDLC) es el proceso paso a paso de crear un nuevo software. Los ingenieros y desarrolladores de software utilizan modelos de ciclo de vida para mapear las fases del desarrollo de software, asegurando que el proceso sea eficiente y rentable. Este blog discutirá cada paso del SDLC en profundidad y describirá diferentes modelos de uso común.

¿Qué es el Ciclo de Vida del Desarrollo de Software?

El ciclo de vida del desarrollo de software abarca el desarrollo de nuevo software desde la etapa de planificación inicial hasta la implementación y el mantenimiento a largo plazo. Es una herramienta de mapeo que ayuda a los desarrolladores de software a medida que crean un nuevo software.

Diferentes metodologías SDLC se adaptarán mejor a diferentes tipos de proyectos. Cada modelo enfatiza los diferentes beneficios de SDLC.

¿Por qué es importante SDLC?

El software suele ser desarrollado por equipos de ingenieros y desarrolladores. Es fundamental que cada persona que trabaje en el proyecto siga el mismo proceso. Sin esta consistencia, sería casi imposible desarrollar un software exitoso y entregarlo al cliente a tiempo.

SDLC le da a cada proyecto un framework individualizado para trabajar dentro. Mantiene a todo el equipo de desarrollo en la misma página para que el proceso permanezca ordenado y eficiente y resulte en productos de software de alta calidad y bajo costo.

7 fases del Ciclo de Vida del Desarrollo de Software

Hay siete fases del ciclo de vida de desarrollo de software que se siguen en cada uno de los diferentes modelos. Cada fase cumple una función distinta y juntas proporcionan un framework de programación integral para el desarrollo de software eficiente.

Hay requisitos previos que deben cumplirse antes de que cada fase pueda comenzar o finalizar. Estos se conocen como puntos de entrada y puntos de salida. Todos los miembros del equipo deben seguir las fases de SDLC en orden secuencial para garantizar que el software se complete de manera precisa, rentable y antes de la fecha límite del cliente.

1. **Planificación y Análisis de Requerimientos**

La fase de planificación y análisis de requisitos es el primer paso en el desarrollo de software. Aquí es cuando los miembros superiores del equipo recopilan información sobre el software que se desarrollará. Hablan con el cliente para conocer las intenciones del proyecto y luego identificar posibles riesgos, problemas y oportunidades.

Los miembros del equipo a menudo reciben aportes de múltiples partes interesadas y expertos de la industria. En esta fase, el equipo determina qué costos y recursos se requerirán para completar el proyecto.

2. **Definir requisitos**

Una vez que se completa el análisis de requisitos, los requisitos del programa de software se documentan en un documento de especificación de requisitos de software. Deben ser aceptados por las partes interesadas antes de que el equipo de desarrollo pueda comenzar el proceso de diseño.

3. **Diseño y Prototipado**

Durante esta fase, toda la información recopilada en los dos pasos anteriores se reúne cuando el equipo comienza a diseñar la arquitectura del software. Se pueden presentar múltiples diseños para que el equipo pueda decidir cuál es el mejor para seguir adelante.

4. **Desarrollo de software**

Esta fase es la más larga de todas las fases SDLC. Es donde los desarrolladores comienzan a programar y dar vida al proyecto. Puedes hacerlo usando una variedad de herramientas como compiladores, intérpretes y depuradores. El lenguaje de programación que utilices dependerá de los requisitos del software.

5. **Pruebas de software**

Una vez que se completa el desarrollo del software, debe probarse para asegurarse de que cumple con los requisitos que se identificaron en las fases anteriores. Las pruebas generalmente las realizan equipos de control de calidad y pruebas de software. Comprueban que no haya defectos en el código y que el software funcione como se espera.

Si se encuentran defectos durante las pruebas, el código se envía de vuelta al equipo de desarrollo para que lo arreglen. Esta fase continúa hasta que se corrigen todos los errores.

6. **Implementación del software**

Durante la fase de implementación, el software se entrega al cliente y se pone en uso.

7. **Operaciones y Mantenimiento**

Una vez que se implementa el software, el trabajo no ha terminado. Es probable que surjan problemas que no se detectaron durante la fase de prueba. Durante la fase de mantenimiento continuo, los problemas que surgen se solucionan mediante actualizaciones y parches de software. También se pueden agregar nuevas características a medida que avanza la tecnología.

Modelos del Ciclo de Vida del Desarrollo de Software

Se utilizan diferentes metodologías de desarrollo de software para diferentes tipos de proyectos. Si bien cada uno sigue los mismos pasos básicos, los pasos se ponen en práctica de manera diferente en diferentes modelos. Aquí hay un vistazo a siete de los modelos más utilizados, incluidos los beneficios y desventajas de cada uno.

1. **El modelo Ágil**

El modelo ágil se llama acertadamente, ya que funciona muy bien para proyectos que tienen requisitos cambiantes. Implementa las fases SDLC en una estructura circular que permite flexibilidad.

Beneficios

- Flexible para proyectos que experimentan requisitos cambiantes
- Uno de los modelos más rápidos para trabajar con productos que tienen plazos rápidos
- Recopila comentarios de los clientes durante cada fase

Desventajas

- La documentación no se lleva a cabo hasta más adelante en el proceso, lo que dificulta la transferencia del proyecto a diferentes equipos.
- Es difícil medir con precisión los recursos que se necesitarán para productos grandes al principio del proceso
- Difícil de utilizar para los programadores menos experimentados

2. **Modelo Espiral**

Al igual que el modelo ágil, el modelo en espiral utiliza una estructura circular para moverse por los pasos del SDLC. Utiliza iteraciones, o repeticiones de procedimientos, y cada iteración aborda los requisitos específicos del proyecto. Es un modelo basado en el riesgo que funciona bien para proyectos de alto riesgo con requisitos complejos.

Beneficios

- Identifica los riesgos al principio del proceso.
- Transparente, que requiere retroalimentación durante cada fase
- Debido a que los problemas se descubren en las primeras etapas, las estimaciones de costo y tiempo para el proyecto son bastante precisas

Desventajas

- Muy intensivo en tiempo
- Un modelo complejo que requiere que los desarrolladores sean expertos en evaluación de riesgos
- Es posible que la espiral continúe indefinidamente, sin llegar a completar el desarrollo del producto final

3. Modelo de cascada

El modelo de cascada es uno de los modelos SDLC más probados. Avanza a través de las fases de forma lineal, lo que lo hace ideal para proyectos pequeños que tienen requisitos claros e invariables.

Ventajas

- Fácil de usar incluso para programadores sin experiencia
- Los puntos de entrada y salida de cada fase son fácilmente comprensibles
- Esto conduce a un producto final de alta calidad

Desventajas

- No proporciona comentarios del cliente a lo largo del proyecto
- No funcionará bien para proyectos de alto riesgo o en curso
- Las pruebas no comienzan hasta el final del proceso, dejando los problemas desapercibidos hasta el final

4. **Modelo iterativo**

El modelo iterativo es un modelo cíclico que repite una serie de pasos repetidamente, con cada iteración acercándose al desarrollo del producto final. Comienza con una versión simple del software y se vuelve más complejo con cada iteración. Este modelo funciona mejor cuando los requisitos del proyecto se entienden claramente.

Ventajas

- Las fallas de diseño se detectan temprano en el proceso
- Muy flexible
- Rentable cuando es necesario realizar cambios

Desventajas

- Difícil de administrar debido a múltiples iteraciones
- No es efectivo para proyectos pequeños.
- Requiere miembros del equipo de análisis de riesgos altamente calificados

5. **Modelo DevOps**

El modelo DevOps se ha vuelto muy popular en el desarrollo de software. Se centra en la colaboración entre todos los involucrados en el proceso de desarrollo. Es un proceso flexible diseñado para entornos de ritmo rápido.

Ventajas

- Pruebas automatizadas
- Tiempo de respuesta rápido
- Resuelve problemas de diseño rápidamente

Desventajas

- Todos en la organización deben estar comprometidos con la colaboración

- Las herramientas deben estar estandarizadas en todos los equipos
- Administrar la implementación de software puede ser un desafío

6. **Modelo en forma de V**

Usando el modelo SDLC en forma de V, el diseño y las pruebas del software se completan en conjunto. Su estructura es similar a la del modelo de cascada, que sigue pasos secuenciales, pero tiene dos brazos: uno para la fase de diseño y otro para la fase de prueba.

Ventajas

- Minimiza los riesgos del proyecto
- Resultados en un producto de alta calidad debido a la garantía de calidad continua
- Económico

Desventajas

- Poca flexibilidad
- No produce primeros prototipos de software
- Muy simplista

7. **Modelo del Big Bang**

El modelo big bang no sigue ningún proceso claro y es ideal para proyectos muy pequeños.

Ventajas

- Requiere muy poca planificación
- Fácil de usar para los nuevos programadores
- No intensivo en recursos

Desventajas

- Riesgo muy alto
- Ineficaz para proyectos complejos
- No funcionará bien para proyectos en curso

Mejores prácticas en SDLC

Seguir las mejores prácticas de SDLC es fundamental para garantizar que se mantenga la seguridad y que el proceso de desarrollo funcione sin problemas. Las tres mejores prácticas comúnmente utilizadas son el control de fuente, la integración continua y los sistemas de gestión SDLC.

Fuente de control

El control de fuente se refiere al almacenamiento de todo el código en un solo lugar seguro. Es imperativo para mantener los programas seguros durante el desarrollo. El control de código fuente también permite a los programadores realizar un seguimiento de los cambios realizados en el código de un proyecto y mantiene un historial de todos los cambios realizados.

Integración continua

La integración continua (CI) automatiza la fusión de códigos de diferentes desarrolladores en una plataforma de software. Esto permite que los errores se identifiquen antes en el proceso. IC mejora la calidad y acelera el tiempo que lleva lanzar el nuevo software.

Sistemas de gestión SDLC

Los sistemas de gestión SCDL supervisan el ciclo de vida en cada fase para aumentar la eficiencia y la transparencia.

¿Qué es la Fase de Diseño?

La fase de diseño es el puente que conecta el análisis de requisitos con la implementación del software. En esta etapa, se traduce la visión conceptual del sistema, definida en la fase de análisis, en una representación detallada de cómo se construirá el software.

Objetivos principales:

- **Definir la arquitectura del software:** Cómo se organizarán los componentes del sistema.
- **Especificar los módulos y sus interfaces:** Qué hará cada parte del sistema y cómo se comunicarán entre sí.
- **Diseñar la base de datos:** Cómo se almacenará la información.
- **Crear la interfaz de usuario:** Cómo interactuará el usuario con el sistema.

Un Método Práctico: El Diseño Orientado a Objetos (OO Design)

El Diseño Orientado a Objetos (OO Design) es una extensión natural del Análisis Orientado a Objetos. En esta fase, se detallan las clases, atributos, métodos y relaciones identificados en el análisis, y se definen las responsabilidades de cada objeto.

Pasos clave en el OO Design:

1. **Diseño de clases:**
    - **Detalles de atributos:** Definición de los tipos de datos y restricciones.
    - **Diseño de métodos:** Especificación de la lógica interna de cada método.
    - **Relaciones entre clases:** Refinamiento de las relaciones identificadas en el análisis.
2. **Diseño de interfaces:**
    - Definición de las interfaces que los objetos exponen al mundo exterior.
3. **Diseño de interacción:**
    - Cómo los objetos colaboran para realizar las tareas del sistema.

4. **Diseño de la base de datos:**
    - Diseño del esquema de la base de datos, incluyendo tablas, relaciones y índices.

Herramientas y Técnicas Adicionales

- **Diagramas UML:**
    - Diagramas de clases: Detallan la estructura de las clases y sus relaciones.
    - Diagramas de secuencia: Muestran la interacción entre objetos a lo largo del tiempo.
    - Diagramas de estado: Modelan el comportamiento de un objeto a lo largo de su vida.
- **Patrones de diseño:**
    - Soluciones reutilizables a problemas de diseño comunes.
- **Prototipos:**
    - Versiones simplificadas del sistema para validar el diseño con los usuarios.

Ejemplo Práctico: Una Tienda en Línea (Continuación)

Diseño de clases:

- **Cliente:** Atributos: ID, nombre, dirección, correo electrónico; Métodos: realizarPedido, verHistorial.
- **Producto:** Atributos: ID, nombre, precio, stock; Métodos: calcularDescuento.
- **Orden:** Atributos: ID, fecha, estado, cliente (asociación), items (composición); Métodos: calcularTotal.

Diagrama de secuencia (Ejemplo):

[Imagen de un diagrama de secuencia UML mostrando el proceso de realizar una compra en una tienda en línea]

Beneficios del OO Design

- **Reutilización:** Los objetos y clases pueden reutilizarse en diferentes partes del sistema.
- **Mantenibilidad:** Los cambios se pueden realizar de forma más localizada.
- **Flexibilidad:** El diseño orientado a objetos facilita la adaptación a nuevos requisitos.

Conclusiones

La fase de diseño es esencial para garantizar que el software se construya de manera eficiente y que cumpla con los requisitos del usuario. El Diseño Orientado a Objetos proporciona un marco sólido para modelar sistemas complejos. Al combinar las técnicas de diseño con herramientas como los diagramas UML, los ingenieros de software pueden crear diseños de alta calidad que faciliten la implementación y el mantenimiento del software.

Actividades para los Estudiantes:

- **Ejercicio:** Tomando como base el diagrama de clases UML de la tienda en línea, diseñar un diagrama de secuencia para el proceso de pago.
- **Estudio de caso:** Analizar el diseño de una aplicación existente e identificar los patrones de diseño utilizados.
- **Presentación:** Explicar los beneficios del Diseño Orientado a Objetos y comparar con otros enfoques de diseño.

FUENTE:

- https://www.uml.org/
- https://www.codingdojo.la/2023/06/16/guia-del-ciclo-de-vida-del-desarrollo-de-software/
- https://aws.amazon.com/es/what-is/sdlc/
- BURCH, John; GRUDNISKY, Gary. "Diseño de Sistemas de Información", Grupo Noriega editores.
- SENN, James A. "Análisis y diseño de sistemas de información", 2da. ed., McGraw-Hill.


INSTITUTO
KHIPU