Contents

Clase 7: Requisitos de Mantenibilidad y Portabilidad

Objetivos de la clase:

- Definir y comprender la importancia de los requisitos de mantenibilidad y portabilidad.
- Identificar factores que influyen en la mantenibilidad y portabilidad del software.
- Aprender a especificar requisitos de mantenibilidad y portabilidad de forma clara y medible.
- Conocer las estrategias para mejorar la mantenibilidad y portabilidad del software.

Contenido Teórico Detallado:

1. Requisitos de Mantenibilidad:

- **Definición:** La mantenibilidad se refiere a la facilidad con la que un software puede ser modificado, corregido, adaptado o mejorado para satisfacer nuevas necesidades o corregir defectos. Una alta mantenibilidad reduce los costos y el tiempo asociados con la evolución del software.
- **Importancia:**
  - **Reducción de Costos:** Facilita la corrección de errores y la implementación de nuevas funcionalidades, disminuyendo los costos a largo plazo.
  - **Adaptabilidad:** Permite que el software se adapte a los cambios en el entorno, como nuevas tecnologías, requisitos del negocio o regulaciones.
  - **Extensibilidad:** Facilita la adición de nuevas funcionalidades sin introducir defectos.
  - **Mejora Continua:** Permite la evolución del software a lo largo del tiempo.
- **Factores que Influyen en la Mantenibilidad:**
  - **Modularidad:** Dividir el sistema en módulos independientes y cohesivos facilita el entendimiento y la modificación.
  - **Documentación:** Una documentación completa y actualizada es crucial para entender el software.
  - **Complejidad:** Un código complejo y difícil de entender dificulta la mantenibilidad.
  - **Estilo de Codificación:** Seguir un estilo de codificación consistente y legible mejora la comprensión del código.
  - **Pruebas:** Las pruebas unitarias, de integración y de sistema facilitan la detección de errores y la verificación de cambios.
  - **Uso de Estándares:** Adherirse a estándares de codificación y diseño facilita la comprensión y modificación del código.
  - **Deuda Técnica:** La acumulación de decisiones de diseño subóptimas que sacrifican la calidad a corto plazo afecta negativamente la mantenibilidad.
- **Métricas de Mantenibilidad:**
  - **Complejidad Ciclomática:** Mide la complejidad de un programa a través del número de caminos linealmente independientes a través del código fuente. Un valor alto indica una mayor complejidad y menor mantenibilidad.
  - **Acoplamiento:** Mide la interdependencia entre los módulos. Un alto acoplamiento dificulta la modificación de un módulo sin afectar a otros.
  - **Cohesión:** Mide el grado en que los elementos dentro de un módulo están relacionados. Una alta cohesión facilita la comprensión y modificación del módulo.
  - **Número de Líneas de Código (LOC):** A mayor número de líneas, mayor complejidad y menor mantenibilidad.
- **Ejemplos de Requisitos de Mantenibilidad:**

- "El software debe estar diseñado de forma modular, con una alta cohesión dentro de cada módulo y un bajo acoplamiento entre ellos."
- "El código debe estar documentado de acuerdo con los estándares de la empresa, incluyendo comentarios explicativos y documentación de la API."
- "La complejidad ciclomática de cada función no debe exceder de 10."
- "El software debe incluir pruebas unitarias que cubran al menos el 80% del código."
- "El tiempo medio para corregir un error (MTTR) no debe exceder de 4 horas."

2. Requisitos de Portabilidad:

- **Definición:** La portabilidad se refiere a la facilidad con la que un software puede ser transferido de un entorno (hardware, sistema operativo, plataforma) a otro. Un software portable minimiza el esfuerzo necesario para adaptarlo a diferentes plataformas.
- **Importancia:**
  - **Amplia Compatibilidad:** Permite que el software funcione en una variedad de plataformas, aumentando su alcance y mercado potencial.
  - **Independencia de la Plataforma:** Reduce la dependencia de una plataforma específica, mitigando los riesgos asociados con los cambios en la tecnología.
  - **Reducción de Costos:** Minimiza los costos de desarrollo y mantenimiento al evitar la necesidad de escribir versiones separadas para cada plataforma.
  - **Flexibilidad:** Permite que el software se adapte a las necesidades cambiantes del usuario.
- **Factores que Influyen en la Portabilidad:**
  - **Uso de Lenguajes y Frameworks Portables:** Utilizar lenguajes como Java, Python o frameworks multiplataforma (React Native, Flutter) facilita la portabilidad.
  - **Abstracción de la Plataforma:** Aislar el código específico de la plataforma del resto del código.
  - **Independencia del Hardware:** Minimizar la dependencia de características específicas del hardware.
  - **Uso de Estándares:** Adherirse a estándares abiertos y evitar el uso de extensiones propietarias.
  - **Pruebas en Diferentes Plataformas:** Realizar pruebas exhaustivas en diferentes plataformas para asegurar la compatibilidad.
  - **Gestión de Dependencias:** Usar gestores de dependencias (e.g., Maven, pip) para facilitar la instalación y configuración en diferentes entornos.
  - **Contenedores:** Utilizar contenedores (e.g., Docker) para empaquetar la aplicación y sus dependencias en una unidad portable.
- **Ejemplos de Requisitos de Portabilidad:**
  - "El software debe ser portable a los sistemas operativos Windows, macOS y Linux."
  - "El software debe ser compatible con las versiones 11, 17 y 21 de la JVM."
  - "El software no debe depender de características específicas del hardware, como la tarjeta gráfica o el procesador."
  - "El software debe utilizar una base de datos compatible con diferentes sistemas operativos."
  - "El tiempo necesario para portar el software a una nueva plataforma no debe exceder de 2 semanas."

Ejemplo/Caso de Estudio:

Caso de Estudio: Desarrollo de una Biblioteca de Componentes UI Reutilizables:

Una empresa está desarrollando una biblioteca de componentes UI reutilizables para usar en múltiples proyectos web y móviles. Para garantizar la mantenibilidad y portabilidad de la biblioteca, se deben considerar los siguientes requisitos:

- **Mantenibilidad:**
  - **Modularidad:** Cada componente debe ser un módulo independiente con una interfaz clara y documentada.

  - **Estilo de Codificación:** La biblioteca debe seguir un estilo de codificación consistente y legible (e.g., usando ESLint y Prettier).
  - **Pruebas:** Cada componente debe tener pruebas unitarias que cubran todos los casos de uso y estados posibles.
  - **Documentación:** La biblioteca debe tener una documentación completa que explique cómo usar cada componente y sus propiedades.
- **Portabilidad:**
  - **Framework:** La biblioteca debe estar construida usando un framework multiplataforma como React o Vue.js.
  - **CSS:** La biblioteca debe usar un sistema de estilos que sea compatible con diferentes navegadores y dispositivos (e.g., CSS Modules o Styled Components).
  - **Dependencias:** La biblioteca debe tener un mínimo de dependencias externas y utilizar un gestor de dependencias para facilitar la instalación y actualización.

Problemas Prácticos/Ejercicios con Soluciones:

1. **Ejercicio:** Identifica los requisitos de mantenibilidad y portabilidad para una aplicación web de comercio electrónico.
   - **Solución:**
     - **Mantenibilidad:**
       * Modularidad: El sistema debe estar dividido en módulos como catálogo, carrito de compras, pagos y gestión de usuarios.
       * Documentación: Documentación completa de la API y el esquema de la base de datos.
       * Pruebas: Pruebas unitarias, de integración y pruebas automatizadas de la interfaz de usuario.
     - **Portabilidad:**
       * El software debe ser compatible con los navegadores Chrome, Firefox, Safari y Edge.
       * La aplicación debe ser accesible desde dispositivos móviles (responsive design).
       * Debe ser desplegable en diferentes entornos de nube (AWS, Azure, GCP).
2. **Ejercicio:** Escribe especificaciones SMART para los siguientes requisitos no funcionales:
   - Mantenibilidad
   - Portabilidad
   - **Solución:**
     - **Mantenibilidad:**
       * Específico: El tiempo medio para corregir un error (MTTR) no debe exceder de 4 horas.
       * Medible: Medir el tiempo transcurrido desde la detección del error hasta su corrección.
       * Alcanzable: Establecer un proceso de gestión de errores y asignar recursos para su corrección.
       * Relevante: La mantenibilidad afecta directamente la capacidad de responder a problemas y mejorar la aplicación.
       * Temporal: El MTTR se debe medir mensualmente y revisar trimestralmente.
     - **Portabilidad:**
       * Específico: La aplicación debe ser portable a los sistemas operativos Windows, macOS y Linux.
       * Medible: Medir el tiempo necesario para completar la portabilidad a cada sistema operativo.
       * Alcanzable: Utilizar tecnologías multiplataforma y realizar pruebas exhaustivas.
       * Relevante: La portabilidad aumenta el alcance de la aplicación a diferentes usuarios.
       * Temporal: La portabilidad debe completarse en un plazo de 2 semanas por sistema operativo.

Materiales Complementarios Recomendados:

- **Libros:**
  - "Software Engineering" de Ian Sommerville
  - "Clean Code: A Handbook of Agile Software Craftsmanship" de Robert C. Martin
  - "Release It!: Design and Deploy Production-Ready Software" de Michael T. Nygard
- **Artículos:**
  - IEEE Std 1219-1998, IEEE Standard for Software Maintenance.
  - Artículos sobre métricas de calidad del código (Complejidad Ciclomática, Acoplamiento, Cohesión).
- **Cursos Online:**
  - Cursos en Coursera, edX o Udacity sobre diseño de software, patrones de diseño y arquitectura de software.
  - Tutoriales sobre Docker y contenedores.
- **Herramientas:**
  - SonarQube: Para análisis estático de código y medición de la calidad.
  - Docker: Para la creación y gestión de contenedores.