Contents

Clase 2: Dimensiones de la Calidad del Software y el Triángulo de Hierro

1. Objetivos específicos de la clase:

- Identificar y comprender las diferentes dimensiones (atributos) que componen la calidad del software.
- Comprender el concepto del "Triángulo de Hierro" (alcance, tiempo, costo) y su impacto en la calidad del software.
- Analizar cómo las decisiones de diseño y desarrollo afectan a cada una de las dimensiones de calidad.
- Aplicar el concepto del Triángulo de Hierro a escenarios de proyectos de software.

2. Contenido teórico detallado:

- **Repaso de la Definición de Calidad:** Recordemos que la calidad del software es el grado en que un producto de software satisface las necesidades del usuario y cumple con los requisitos especificados. Es una evaluación multidimensional que considera varios atributos.
- **Dimensiones (Atributos) de la Calidad del Software:**
  - **Funcionalidad:** Se refiere a si el software hace lo que se supone que debe hacer. ¿Cumple con las especificaciones? ¿Proporciona las características y funciones requeridas? Esto incluye exactitud, completitud y adecuación. Por ejemplo, en un software de contabilidad, la funcionalidad incluye la capacidad de realizar cálculos precisos, generar informes financieros completos y cumplir con las regulaciones fiscales aplicables.
  - **Confiabilidad:** Se refiere a la capacidad del software para funcionar sin fallas durante un período de tiempo especificado y bajo condiciones específicas. Incluye la madurez (frecuencia de fallas), la tolerancia a fallas (capacidad de recuperación ante errores) y la recuperabilidad (facilidad de restaurar el sistema después de una falla). Por ejemplo, un sistema de control de tráfico aéreo debe ser altamente confiable para evitar accidentes.
  - **Usabilidad:** Se refiere a la facilidad con la que los usuarios pueden aprender a usar el software y realizar tareas de manera eficiente y satisfactoria. Incluye la facilidad de aprendizaje, la eficiencia en el uso, la memorabilidad, la prevención de errores y la satisfacción del usuario. Por ejemplo, una aplicación móvil de banca debe ser intuitiva y fácil de usar para que los usuarios puedan realizar transacciones financieras sin dificultad.
  - **Eficiencia:** Se refiere a la cantidad de recursos (tiempo, memoria, energía) que el software utiliza para realizar sus funciones. Incluye el comportamiento temporal (rapidez de respuesta), la utilización de recursos (uso eficiente de memoria y procesador) y la capacidad (cantidad de usuarios o datos que el sistema puede manejar). Por ejemplo, un motor de búsqueda debe ser eficiente para proporcionar resultados rápidos y relevantes a los usuarios.
  - **Mantenibilidad:** Se refiere a la facilidad con la que el software puede ser modificado para corregir errores, agregar nuevas funciones o adaptarse a cambios en el entorno. Incluye la analizabilidad (facilidad para identificar la causa de los errores), la modificabilidad (facilidad para implementar cambios), la estabilidad (impacto mínimo de los cambios en otras partes del sistema) y la facilidad de prueba. Por ejemplo, un sistema operativo debe ser fácil de mantener para que los desarrolladores puedan corregir errores de seguridad y agregar nuevas características.
  - **Portabilidad:** Se refiere a la facilidad con la que el software puede ser transferido de un entorno a otro (hardware, sistema operativo, base de datos). Incluye la adaptabilidad (facilidad para modificar el software para un nuevo entorno), la instalabilidad (facilidad para instalar el software en un nuevo entorno) y la coexistencia (capacidad para funcionar con otros softwares en el mismo entorno). Por ejemplo, una aplicación web debe ser portable para que los usuarios puedan acceder a ella desde diferentes navegadores y dispositivos.
  - **Seguridad:** Se refiere a la capacidad del software para proteger la información y los recursos del acceso no autorizado, la manipulación o la destrucción. Incluye la confidencialidad, la integridad y la disponibilidad.
- **El Triángulo de Hierro (Alcance, Tiempo, Costo):**

- Este concepto fundamental en la gestión de proyectos de software ilustra las restricciones interdependientes que afectan la calidad. Cualquier cambio en uno de los lados del triángulo impacta inevitablemente a los otros dos y, por ende, a la calidad.
- **Alcance:** Se refiere a las características y funciones que se incluirán en el software. Un alcance más amplio generalmente requiere más tiempo y costo.
- **Tiempo:** Se refiere al cronograma para completar el proyecto. Un cronograma más corto generalmente requiere un alcance reducido o un mayor costo (más recursos).
- **Costo:** Se refiere a los recursos financieros disponibles para el proyecto. Un costo limitado generalmente requiere un alcance reducido o un cronograma más largo.
- **Relación con la Calidad:** Reducir el tiempo o el costo sin ajustar el alcance puede llevar a comprometer la calidad del código, a pruebas insuficientes o a una documentación deficiente. Aumentar el alcance sin aumentar el tiempo o el costo puede llevar a las mismas consecuencias. Un buen equilibrio entre estos tres factores es esencial para entregar software de alta calidad.

3. Ejemplos o Casos de Estudio:

- **Caso 1: Desarrollo de una aplicación móvil para un banco.** El banco quiere una aplicación con funcionalidades de consulta de saldo, transferencia y pago de servicios. Si el tiempo para el desarrollo es muy corto (2 meses), y el presupuesto limitado, es posible que se prioricen las funcionalidades básicas, pero se sacrifique la seguridad (implementación rápida de autenticación sin pruebas exhaustivas) o la usabilidad (interfaz poco intuitiva). Esto afectaría directamente la confiabilidad y la satisfacción del cliente.
- **Caso 2: Desarrollo de un sistema de gestión hospitalaria.** Si el alcance es muy amplio (incluir todos los procesos del hospital), pero el equipo de desarrollo es pequeño (costo bajo), y el tiempo es limitado, es probable que se comprometa la mantenibilidad del sistema. Se puede crear código con errores, difícil de entender y modificar en el futuro, generando problemas a largo plazo y aumentando los costos de mantenimiento.
- **Caso 3: Desarrollo de un videojuego.** En este caso, la eficiencia es crucial. Si se sacrifica el tiempo dedicado a la optimización del código para cumplir con una fecha de lanzamiento, el juego podría tener un bajo rendimiento en ciertos dispositivos, afectando la experiencia del usuario.

4. Problemas Prácticos o Ejercicios con Soluciones:

- **Problema 1:** Un equipo de desarrollo está creando un sistema de comercio electrónico. El cliente insiste en agregar muchas características nuevas justo antes del lanzamiento, pero no está dispuesto a aumentar el presupuesto ni extender el cronograma. ¿Cómo afecta esto a la calidad del software? ¿Qué dimensiones de la calidad se verán más afectadas? ¿Qué soluciones propondrías?
  - **Solución:** Esto afectará negativamente a la calidad del software. Las dimensiones más afectadas serán la confiabilidad (mayor probabilidad de errores por cambios apresurados), la mantenibilidad (código desorganizado para cumplir con el plazo) y potencialmente la seguridad (vulnerabilidades por falta de pruebas). Se propondría negociar con el cliente para priorizar las características esenciales y posponer las menos importantes para una versión futura, o solicitar un aumento del presupuesto y tiempo para asegurar la calidad.
- **Problema 2:** Describe un escenario donde la priorización de la usabilidad puede comprometer la eficiencia de un software. ¿Cómo se podría equilibrar la usabilidad y la eficiencia en ese escenario?
  - **Solución:** Un ejemplo podría ser una aplicación de edición de video con una interfaz muy intuitiva y llena de asistentes para facilitar la edición. Sin embargo, estos asistentes pueden consumir muchos recursos, haciendo que el software funcione lentamente, especialmente con archivos grandes. Para equilibrar usabilidad y eficiencia, se podría ofrecer diferentes niveles de usabilidad (modo básico y modo avanzado), permitir al usuario desactivar los asistentes, y optimizar el código para mejorar el rendimiento.

5. Materiales Complementarios Recomendados:

- Artículos sobre el Triángulo de Hierro en la gestión de proyectos de software.

- Lecturas sobre las dimensiones de la calidad del software (ISO 25010).
- Ejemplos de desastres de software causados por comprometer la calidad.