Contents

‘‘‘

Clase 3: Profundizando en los Requisitos de Calidad del Software: Funcionales y No Funcionales

1. Objetivos Específicos de la Clase:

- Diferenciar los requisitos funcionales y no funcionales con mayor detalle y ejemplos avanzados.
- Comprender la importancia de la priorización de requisitos de calidad.
- Aprender técnicas básicas para la elicitación de requisitos de calidad.
- Identificar la relación entre requisitos de calidad y pruebas de software.

2. Contenido Teórico Detallado:

2.1. Revisión y Expansión de Requisitos Funcionales:

Como vimos en la clase anterior, los requisitos funcionales describen *qué* debe hacer el software. Es crucial que estos requisitos sean:

- **Completos:** Cubren todas las funciones necesarias.
- **Consistentes:** No se contradicen entre sí.
- **No ambiguos:** Claros y sin margen a interpretaciones erróneas.
- **Trazables:** Vinculados a las necesidades del usuario y a las pruebas.

Ejemplo Ampliado: En un sistema de comercio electrónico, un requisito funcional podría ser: ”El sistema debe permitir a los usuarios registrados guardar múltiples direcciones de envío”. Este requisito es específico, indica la funcionalidad deseada, y puede ser probado (un usuario puede guardar varias direcciones).

2.2. Profundizando en los Requisitos No Funcionales:

Los requisitos no funcionales restringen *cómo* el sistema debe realizar sus funciones. Son cruciales para la experiencia del usuario y el éxito del software. Vamos a profundizar en algunos tipos clave:

- **Rendimiento:** Más allá de la velocidad, considerar *escalabilidad* (cómo el sistema maneja el aumento de usuarios y datos), *capacidad* (cantidad de datos que puede almacenar), y *eficiencia* (uso óptimo de recursos).
  - **Ejemplo:** ”El sistema debe soportar 1000 usuarios concurrentes con un tiempo de respuesta promedio inferior a 3 segundos durante las horas pico.”
- **Seguridad:** Incluye *autenticación* (verificación de la identidad del usuario), *autorización* (control de acceso a recursos), *confidencialidad* (protección de datos sensibles), *integridad* (prevención de alteraciones no autorizadas), y *disponibilidad* (asegurar el acceso al sistema cuando se necesita).
  - **Ejemplo:** ”Todos los datos de tarjetas de crédito deben almacenarse utilizando encriptación AES-256 y cumplir con la norma PCI DSS.”
- **Usabilidad:** Facilidad de uso, aprendizaje, eficiencia y satisfacción del usuario. Considerar *accesibilidad* (para usuarios con discapacidades) y *experiencia del usuario (UX)*.
  - **Ejemplo:** ”El sistema debe cumplir con las pautas WCAG 2.1 nivel AA de accesibilidad.”
- **Confiabilidad:** Probabilidad de que el sistema funcione correctamente durante un período de tiempo específico. Incluye *disponibilidad* (tiempo durante el cual el sistema está operativo), *robustez* (capacidad de recuperarse de errores), y *tolerancia a fallos* (capacidad de seguir funcionando incluso si algunas partes fallan).
  - **Ejemplo:** ”El sistema debe tener una disponibilidad del 99.99% (cuatro nueves).”

- **Mantenibilidad:** Facilidad para modificar, corregir, adaptar y mejorar el software. Considerar *modularidad* (diseño en componentes independientes), *legibilidad del código*, y *documentación*.
  - **Ejemplo:** "El código debe seguir las convenciones de estilo de la empresa y estar documentado con comentarios claros y concisos."
- **Portabilidad:** Facilidad para transferir el software a diferentes plataformas o entornos.
  - **Ejemplo:** "La aplicación debe ser compatible con los sistemas operativos iOS y Android."

2.3. Priorización de Requisitos de Calidad:

No todos los requisitos de calidad son igualmente importantes. La priorización ayuda a enfocar los esfuerzos y recursos. Técnicas comunes incluyen:

- **MoSCoW:** *Must have* (imprescindible), *Should have* (debería tener), *Could have* (podría tener), *Won't have* (no tendrá en esta versión).
- **Priorización basada en valor:** Asignar un valor a cada requisito según su importancia para el negocio y su costo de implementación.
- **Matriz de priorización:** Comparar los requisitos entre sí utilizando criterios como impacto en el cliente, riesgo, y costo.

Ejemplo: En una aplicación de banca móvil, la *seguridad* (protección de datos financieros) sería un *Must have*, mientras que una función de *personalización de la interfaz* podría ser un *Could have*.

2.4. Técnicas Básicas para la Elicitación de Requisitos de Calidad:

La elicitación es el proceso de descubrir y documentar los requisitos. Algunas técnicas:

- **Entrevistas:** Conversaciones con usuarios, stakeholders y expertos.
- **Cuestionarios:** Recopilación de información estructurada de un gran número de personas.
- **Talleres:** Reuniones colaborativas para generar y validar requisitos.
- **Brainstorming:** Generación libre de ideas.
- **Análisis de documentos existentes:** Revisión de documentos como especificaciones de productos similares, estándares de la industria, y regulaciones.
- **Prototipos:** Creación de versiones tempranas del software para obtener retroalimentación.

2.5. Relación entre Requisitos de Calidad y Pruebas de Software:

Los requisitos de calidad son la base para las pruebas de software. Cada requisito debe ser verificable, lo que significa que debe ser posible diseñar pruebas para determinar si se ha cumplido.

- **Pruebas funcionales:** Verifican que el software cumple con los requisitos funcionales.
- **Pruebas no funcionales:** Verifican que el software cumple con los requisitos no funcionales (rendimiento, seguridad, usabilidad, etc.).
- **Casos de prueba:** Describen cómo probar una función específica o un requisito de calidad.

Ejemplo: Si un requisito de rendimiento establece que "el sistema debe responder en menos de 2 segundos", se pueden diseñar casos de prueba para medir el tiempo de respuesta bajo diferentes condiciones de carga.

3. Ejemplos o Casos de Estudio:

Caso de Estudio: Aplicación de Streaming de Video

Consideremos una aplicación de streaming de video como Netflix. Algunos requisitos de calidad clave podrían ser:

- **Funcional:** Los usuarios deben poder buscar películas y series por título, género, actor, etc.
- **No Funcional (Rendimiento):** El video debe comenzar a reproducirse en menos de 3 segundos, incluso con conexiones de internet de baja velocidad.
- **No Funcional (Seguridad):** La aplicación debe proteger los datos de la tarjeta de crédito de los usuarios.

- **No Funcional (Usabilidad):** La interfaz debe ser intuitiva y fácil de navegar en diferentes dispositivos (teléfonos, tabletas, televisores).
- **No Funcional (Confiabilidad):** El servicio debe estar disponible el 99.9% del tiempo.

4. Problemas Prácticos o Ejercicios con Soluciones:

Ejercicio 1: Para una aplicación de gestión de citas médicas, identifique 3 requisitos funcionales y 3 requisitos no funcionales (de diferentes tipos).

Solución:

- **Funcionales:**
  - El sistema debe permitir a los pacientes reservar citas online.
  - El sistema debe enviar recordatorios de citas a los pacientes por SMS y correo electrónico.
  - El sistema debe permitir al personal médico acceder al historial clínico de los pacientes.
- **No Funcionales:**
  - *Rendimiento:* El sistema debe permitir la reserva de una cita en menos de 5 segundos.
  - *Seguridad:* El acceso a la información del paciente debe estar restringido al personal autorizado mediante autenticación de dos factores.
  - *Usabilidad:* La interfaz de la aplicación debe ser intuitiva y fácil de usar para personas mayores.

Ejercicio 2: Priorice los siguientes requisitos de calidad para un sistema de control de tráfico aéreo utilizando la técnica MoSCoW:

- El sistema debe mostrar la posición de todas las aeronaves en un radio de 500 km.
- El sistema debe emitir alertas en caso de riesgo de colisión.
- El sistema debe permitir a los controladores comunicarse con las aeronaves.
- El sistema debe registrar todos los eventos del sistema para fines de auditoría.
- El sistema debería tener una interfaz visualmente atractiva.

Solución:

- *Must have:*
  - El sistema debe mostrar la posición de todas las aeronaves en un radio de 500 km.
  - El sistema debe emitir alertas en caso de riesgo de colisión.
  - El sistema debe permitir a los controladores comunicarse con las aeronaves.
- *Should have:* El sistema debe registrar todos los eventos del sistema para fines de auditoría.
- *Could have:* El sistema debería tener una interfaz visualmente atractiva.

5. Materiales Complementarios Recomendados:

- **Libros:**
  - "Software Requirements" de Karl Wiegers y Joy Beatty.
  - "Requirements Engineering" de Elizabeth Hull, Ken Jackson y Jeremy Dick.
- **Artículos:**
  - IEEE Standard 830-1998: Recommended Practice for Software Requirements Specifications.
- **Recursos en línea:**
  - Sitios web de ingeniería de requisitos (IREB, IIBA).
  - Artículos y tutoriales sobre la técnica MoSCoW.

‘‘‘