1. Requisito software

Declaración que describe una característica, función, o propiedad que debe tener el software. Esenciales para definir y comprender las expectativas del cliente, y sirve como guía para los equipos de desarrollo.

1.1. Requisito funcional

Funciones del sistema a diseñar. Descripción del sistema y cómo funcionará para satisfacer las necesidades del usuario. Define lo que hace el sistema.

1.2. Requisito no funcional

Limitaciones y restricciones del sistema a diseñar. No tienen impacto en la funcionalidad. Define como lo hace el sistema.

2. Prueba software

Proceso controlado y planificado que examina el programa o aplicación para identificas problemas, errores o defectos. Pueden ser fallos en el funcionamiento o situaciones inesperadas que podrían hacer que falle.

2.1. Planificación de las pruebas

- Verificación: confirma si el software se ha construido correctamente y cumple con las especificaciones definidas.
- Validación: determina si el software cumple con las necesidades reales del usuario.

2.2. Planificación de pruebas

Durante el desarrollo, va a hacer algunas cosas donde es muy probable que se produzca un error humano, que pueden ser:

- Incorrecta especificación de los objetivos.
- Errores durante el proceso de diseño.
- Errores en la fase de desarrollo.

2.3. Errores de programación

Problemas o defectos en el código de un programa. Pueden causar que el programa falle parcial o totalmente.

2.3.1. Errores de sintaxis

Error cuando el código no sigue las reglas gramaticales y estructurales del lenguaje de programación.

2.3.2. Errores en tiempo de ejecución

Ocurre durante la ejecución y no está relacionado con la sintaxis. Son errores por situaciones inesperadas.

2.3.3. Errores lógicos

Problemas en la lógica que pueden hacer que haya resultados incorrectos o inesperados.

2.4. Casos de prueba

Conjuntos de condiciones y acciones diseñados para evaluar el comportamiento de un programa en situaciones específicas. Incluyen una entrada y un resultado esperado.

2.5. Enfoques en las pruebas

- Pruebas estructurales o de caja blanca: enfocadas en la estructura interna del código, tienen acceso al código fuente y buscan identificar errores de lógica y verificar cobertura de código.
- Pruebas funcionales o de caja negra: evalúan la funcionalidad sin tener acceso al código fuente, centrándose en la entrada y en la salida.

2.6. Tipos de pruebas

2.6.1. Pruebas unitarias

Evalúan unidades individuales de código de manera aislada. Su objetivo es verificar que cada unidad funciona correctamente.

2.6.2. Pruebas de integración

Se centran en verificar la interacción adecuada entre diferentes componentes o unidades de código. Su objetivo es verificar que las partes del sistema funcionen correctamente juntas

2.6.3. Pruebas de aceptación de usuario

Pruebas finales que involucraran a los usuarios. Su objetivo es validar que el software cumple con los requisitos desde la perspectiva del usuario.

2.6.4. Pruebas de seguridad

Evaluaciones del software diseñadas para identificar vulnerabilidades y riesgos de seguridad. Su objetico es garantizar que el software está protegido ante amenazas y ataques.

2.6.5. Pruebas de regresión

Pruebas realizadas tras realizar cambios en el software para verificar que no haya nuevos errores o problemas en funcionalidad ya probadas. Su objetivo es garantizar que las actualizaciones no afectan a las partes existentes del software.

2.6.6. Pruebas de usabilidad

Evaluaciones del software centradas en la experiencia del usuario. Su objetivo es garantizar que el software se a intuitivo, eficiente y agradable de usar.

2.6.7. Pruebas de compatibilidad

Se enfocan en asegurar que el software funcione adecuadamente en diferentes entornos y configuraciones.

2.6.8. Pruebas de localización e internacionalización

Evalúan si el software se adapta correctamente a las especificaciones culturales y lingüísticas de una región o un mercado específico.

2.6.9. Pruebas de cobertura

- Cubrimiento: son los casos que están cubiertos en la prueba.
- Valores límite: son los valores límite donde justo una función va a cambiar de resultado.
- Clases de equivalencia: los valores que representan a un conjunto de valores, donde cada conjunto da un resultado diferente.

3. Depuración de código

Proceso de identificar, analizar y corregir errores o fallos en un programa de software. Se revisa el código en busca de problemas de sintaxis, lógica o comportamiento incorrecto. Su objetivo es garantizar que el software funcione correctamente.

4. Mock

Objeto o componente simulado utilizado en pruebas para reemplazar un componente real. Imitan el comportamiento de un componente real, pero se controlan y se configuran para las pruebas. Útiles para verificar la interacción entre componentes y garantizar que una unidad de código se comunique con otras correctamente.

5. Desarrollo Dirigido por Pruebas (TDD)

Metodología de desarrollo donde se escriben las pruebas antes que el código. implica tres pasos:

1. Escribir una prueba que falle.
2. Escribir el código mínimo necesario para que pase la prueba.
3. Refactorizar el código.

Fomenta la creación de código más limpio, confiable y orientado a los requisitos, al tiempo que se acelera la detección temprana de errores.