UT7.1. Realización de pruebas

18 horas

Contenidos

1. El proceso de desarrollo.
    a. Fases
    b. Objetivo, importancia y limitaciones del proceso de prueba.
    c. Depuración de código.
2. Tipos de pruebas
    a. Pruebas de integración: ascendentes y descendentes.
    b. Pruebas de regresión.
    c. Pruebas funcionales.
    d. Pruebas de sistema: configuración, recuperación, entre otras.
    e. Pruebas de capacidad y rendimiento.
    f. Pruebas de uso de recursos.
    g. Pruebas de seguridad.
    h. Pruebas manuales y automáticas. Herramientas software para la realización de pruebas.
    i. Pruebas de usuario.
    j. Pruebas de aceptación.
3. Versiones alfa y beta.

Resultados de aprendizaje

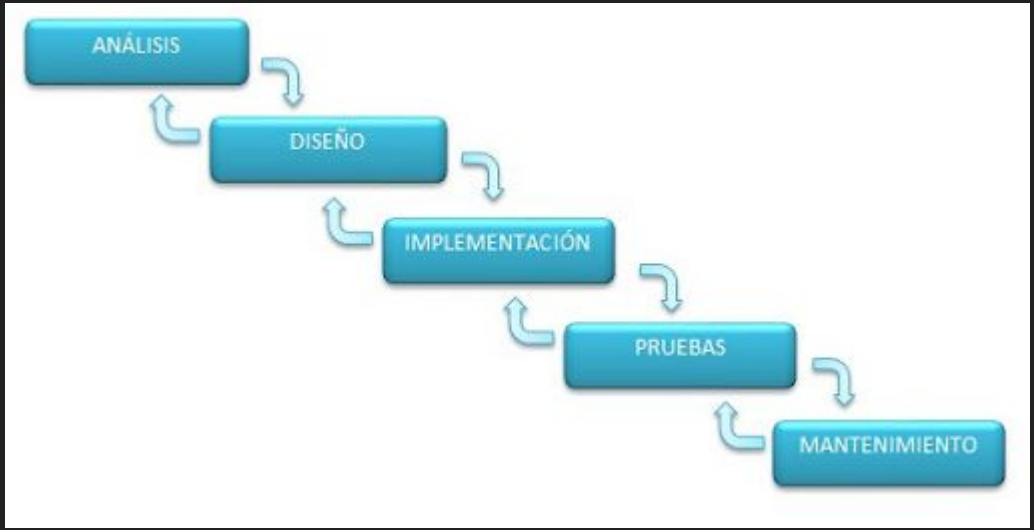
RA8: Evalúa el funcionamiento de aplicaciones diseñando y ejecutando pruebas	**15%**
a) Se ha establecido una estrategia de pruebas.	15%
b) Se han realizado pruebas de integración de los distintos elementos.	15%
c) Se han realizado pruebas de regresión.	15%
d) Se han realizado pruebas de volumen y estrés.	15%
e) Se han realizado pruebas de seguridad.	15%
f) Se han realizado pruebas de uso de recursos por parte de la aplicación.	15%
g) Se ha documentado la estrategia de pruebas y los resultados obtenidos.	10%

El proceso de desarrollo

Lineal

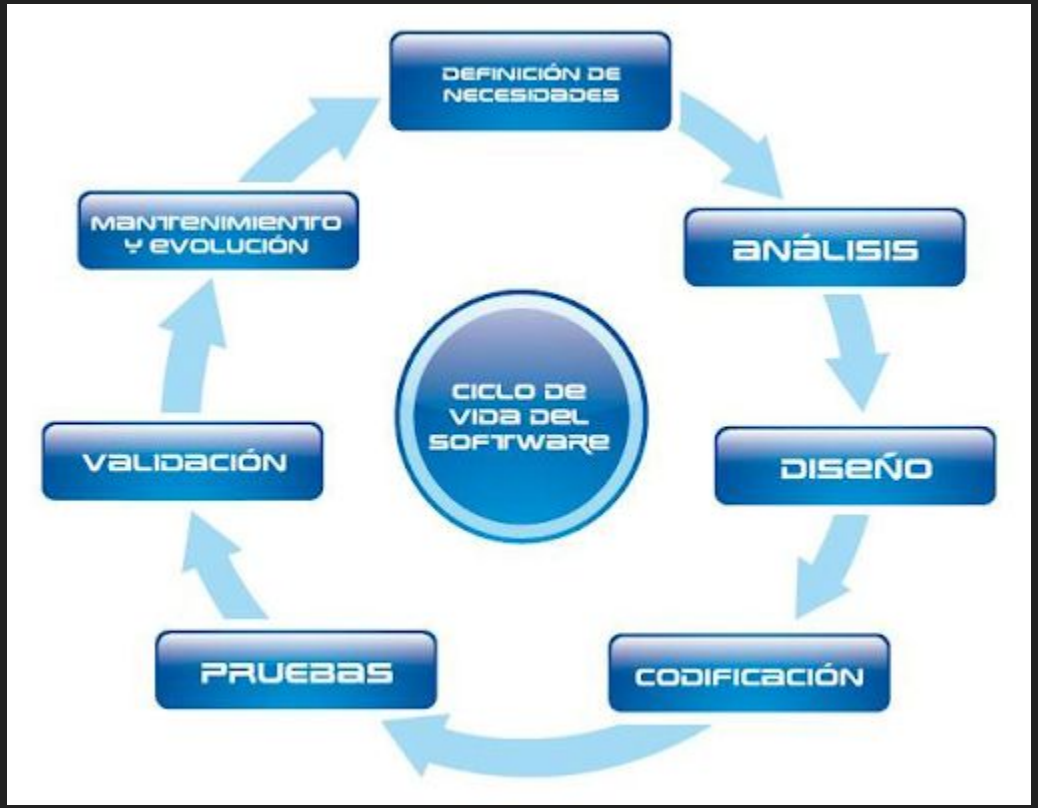
El proceso de desarrollo

Cascada



El proceso de desarrollo

Circular



¿Porqué son tan importantes las pruebas de SW?

Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e incrementando su perjuicio cuanto más tardía sea su detección.

(Bell y Thayer 1976)(Davis 1993)

¿Porqué son tan importantes las pruebas de SW?

¡Las pruebas se diseñan para destruir lo que acabamos de construir!

Objetivos:

- La prueba es una ejecución del programa buscando **descubrir un error**.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene **éxito** si descubre un error no detectado hasta entonces.
- La prueba no puede detectar la ausencia de defectos, sólo puede demostrar que existen defectos en el software.

¿Porqué son tan importantes las pruebas de SW?

Ejemplos de errores de SW:

Lanzadera Ariane-5 vuelo 501 → Error de excepción de software al convertir un número almacenado en coma flotante de 64 bits en entero de 16 bits.

Sonda Mariner 1 de la NASA → La omisión de un simple guión en las instrucciones del programa de guiado del cohete provocaba la desviación cuando se perdían las señales de radio.

Therac 25 → Un fallo en el software que aplicaba a las personas dosis de radiación hasta 100 veces más fuertes que las que un humano puede soportar.

Diseño de casos de prueba

Pruebas de caja blanca:

Conociendo el funcionamiento del producto se pueden desarrollar pruebas que aseguran que todas las piezas encajan → Pruebas sobre el código ejercitando condiciones y/o bucles.

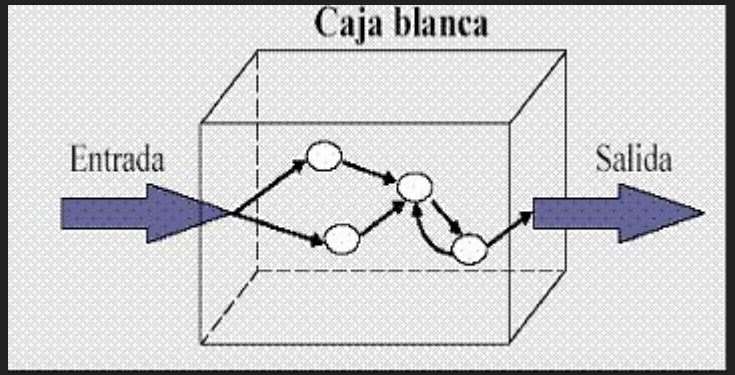
Pruebas de caja negra:

Conocida la función específica para la que se ha diseñado el producto, las pruebas se llevan a cabo para demostrar que cada función es completamente operativa → Pruebas sobre la interfaz del SW (la entrada, la salida y la integridad de la información externa).

Pruebas de caja blanca

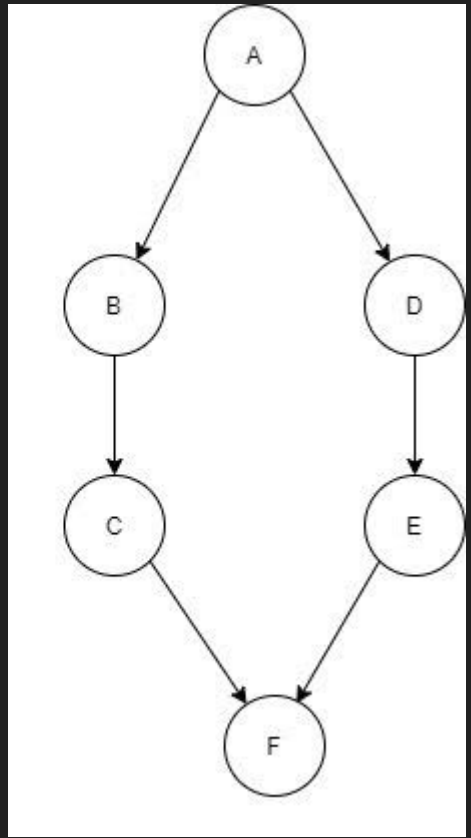
Método de diseño de casos de prueba en el que:

- Se ejercitan al menos una vez todos los caminos independientes de cada módulo
- Todas las decisiones en el camino True y en el False
- Todos los bucles en sus límites
- Todas las estructuras internas de datos



Pruebas de caja blanca

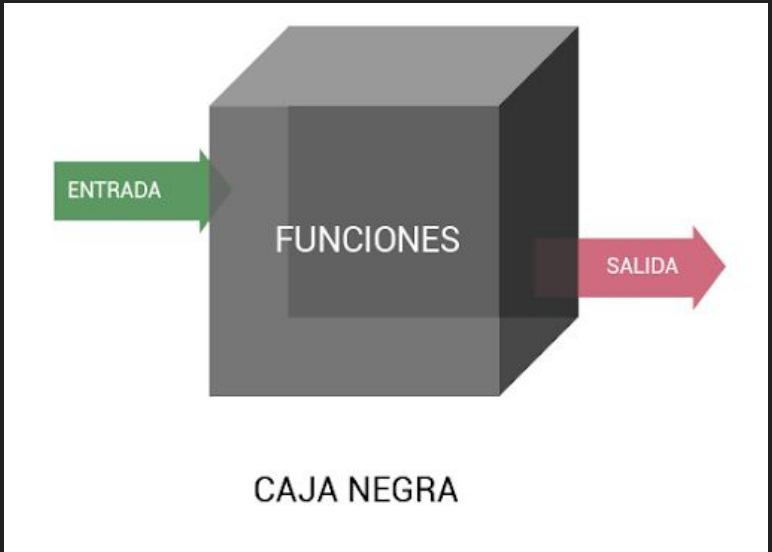
```
public int Max(int x, int y){    //A
     if(x>y){  //B
          print("x es mayor que y")  //C
     }
     else{    //D
          print("y es mayor que x")  //E
     }
} //F
```



Actividad 1: Pruebas con JUnit

Vamos a hacer un ejemplo, abre el pdf del Aula Virtual *JUnit caja blanca.pdf*

Pruebas de caja negra



Los casos de prueba de la caja negra pretenden demostrar que:

- Las funciones del software son operativas
- La entrada se acepta de forma correcta
- Se produce una salida correcta
- La integridad de la información externa se mantiene

Pruebas de caja negra

Las pruebas de caja negra pretenden encontrar estos tipos de errores:

- Funciones incorrectas o ausentes
- **Errores en la interfaz**
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Pruebas de caja negra

Tipos de pruebas de caja negra:

- Prueba de partición equivalente
- Prueba de análisis de valores límites

Prueba de Partición Equivalente

Este método de prueba de caja negra divide el dominio de entrada de un programa en clases de datos, a partir de las cuales deriva los casos de prueba.

Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada.

- Identificación de las clases de equivalencia:

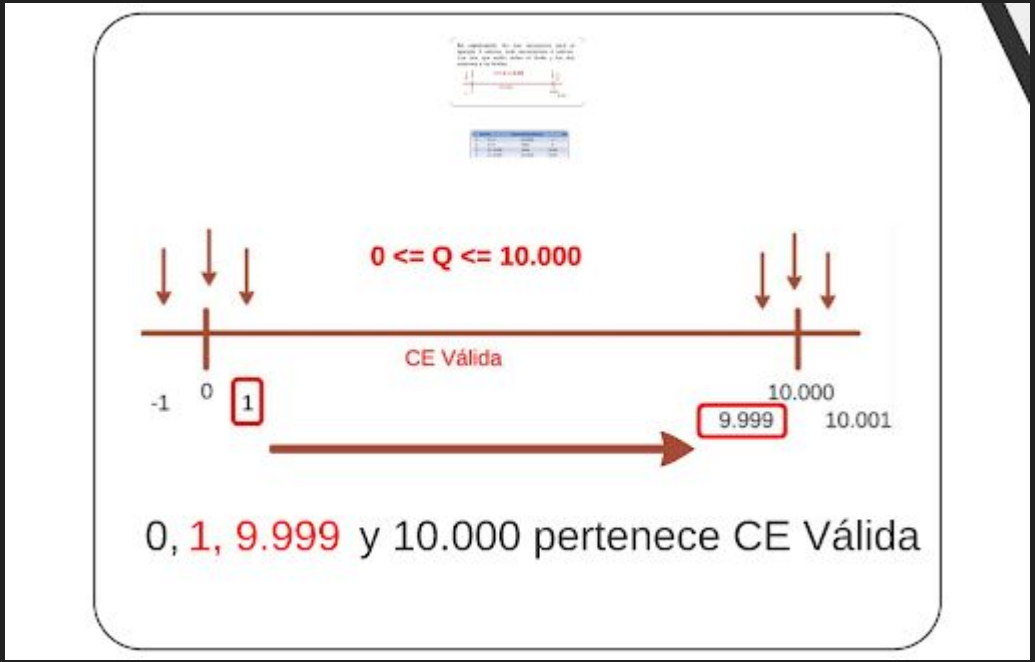
Se identifican clases de equivalencia válidas e inválidas con la siguiente tabla:

Condiciones externas	Clases de equivalencia válidas (CEV)	Clases de equivalencia inválidas (CEI)

Prueba de Partición Equivalente

Condiciones externas	Clases de equivalencia válidas (CEV)	Clases de equivalencia inválidas (CEI)
rango de valores entre 1 y 999	1<=valor<=999	valor <1 && valor >999
el primer caracter debe ser una letra	una letra	no es una letra
vehículos a motor	coche, moto, camión...	bicicleta, patín...
una casa con 1 o 2 propietarios	1 o 2 propietarios	propietarios >2 ó propietarios=0

Prueba de análisis de valores límites



Prueba de análisis de valores límites

Pasos:

3. Crear un caso de prueba por cada valor identificado.

4. Identificar el representante para cada caso.

Nro	Caso de Prueba		Representante
1	Q < 0	No Válido	-1
2	Q = 0	Válido	0
3	Q > 0	Válido	1
4	Q = 10.000	Válido	10.000
5	Q < 10.000	Válido	9.999
6	Q > 10.000	No válido	10.001

Prueba de análisis de valores límites. Ejemplo

Diseñar casos de prueba de partición equivalente para un software que capte estos datos de entrada:

- Código: En blanco o un número de tres dígitos

  99<valor<=999 or “ “

- Prefijo: Número de tres dígitos que no comience por 0 ó 1

  199<valor<=999

- Sufijo: Número de cuatro dígitos

  999>valor<=9999

Tipos de pruebas

1. **Pruebas funcionales** → Lo que hace el sistema. Se basan en la especificación de requisitos. Se centran en verificar que las funciones del software operen de acuerdo a lo especificado. El propósito es comprobar si el sistema realiza las tareas previstas sin tener en cuenta cómo están implementadas. Se usan técnicas de diseño de caja negra.
2. **Pruebas no funcionales** → Cómo funciona el sistema. También estas características están en las especificaciones de producto. Se usan técnicas de diseño de caja negra.
3. **Pruebas estructurales** → Se prueba cómo trabaja cada camino del SW, se usan técnicas de caja blanca.

Tipos de pruebas

4. Pruebas de integración: ascendentes y descendentes

- **Ascendentes:** Se enfocan en probar la integración de componentes desde los módulos más pequeños hacia el sistema completo. Comienzan con las unidades y suben a nivel de integración.
- **Descendentes:** Inician con el sistema completo y luego se desciende hacia los módulos individuales, probando cómo los componentes interactúan entre sí a un nivel más alto.

5. Pruebas de regresión

Se realizan para asegurarse de que los cambios recientes en el software (como nuevas funciones o correcciones de errores) no hayan afectado negativamente las funciones ya existentes. Garantiza la estabilidad del sistema a lo largo del tiempo.

6. Pruebas de sistema: configuración, recuperación, entre otras

- **De configuración:** Validan que el sistema funcione correctamente en diferentes configuraciones (hardware, software, entorno).
- **De recuperación:** Se prueban los mecanismos del sistema para asegurarse de que pueda recuperarse adecuadamente ante fallos o problemas.

Tipos de pruebas

7. Pruebas de capacidad y rendimiento

Se verifican aspectos como la capacidad del sistema para manejar grandes volúmenes de datos o usuarios simultáneos, y el rendimiento en términos de tiempos de respuesta, procesamiento, etc., bajo diferentes condiciones de carga.

8. Pruebas de uso de recursos

Evalúan cuántos recursos (como memoria, CPU y red) utiliza el software durante su ejecución, con el fin de garantizar que no haya fugas de memoria ni un uso ineficiente de los recursos del sistema.

9. Pruebas de seguridad

Se centran en identificar vulnerabilidades en el software, como posibles brechas de seguridad, amenazas externas o la exposición a ataques, asegurando que el sistema sea robusto contra posibles riesgos.

Tipos de pruebas

10. Pruebas manuales y automáticas

- **Manual:** Son realizadas por testers humanos que interactúan directamente con la aplicación para verificar su comportamiento.
- **Automática:** Utilizan herramientas de software para ejecutar pruebas repetitivas o complejas, lo que permite una ejecución más rápida y la repetición continua de pruebas sin intervención manual.
- **Herramientas software para la realización de pruebas:** Existen múltiples herramientas, tanto para pruebas manuales como automáticas, que ayudan a facilitar la ejecución de pruebas. Algunas populares son Selenium, JUnit, TestNG, JIRA, Postman, entre otras.

11. Pruebas de usuario

Se realizan para comprobar cómo interactúan los usuarios finales con el sistema. Pueden implicar desde pruebas de usabilidad hasta pruebas de aceptación. La idea es asegurar que el software sea fácil de usar y cumpla las expectativas del usuario.

12. Pruebas de aceptación

Son pruebas realizadas generalmente al final del ciclo de desarrollo, con el fin de verificar que el software cumpla con los requisitos y criterios de aceptación del cliente. Si el sistema pasa esta prueba, puede ser aceptado y liberado.

Tipos de pruebas

Pruebas Alfa: Se llevan a cabo en un entorno controlado dentro de la empresa desarrolladora. Son realizadas por testers internos o un grupo limitado de usuarios para detectar errores antes de lanzar el software al público.

Pruebas Beta: Se ejecutan en un entorno real con usuarios finales fuera de la empresa. Permiten recopilar comentarios sobre el rendimiento, usabilidad y posibles errores antes del lanzamiento oficial.

Actividad 2: Pruebas de integración

Abre el documento *Prueba de integración JUnit y Mockito.pdf* del aula virtual y estúdialo.