T.3 - DISEÑO Y REALIZACIÓN DE PRUEBAS

PLANIFICACIÓN DE PRUEBAS → proceso que permite verificar y revelar la calidad de un producto SFTW

↳ Conj de pruebas en las que se expresan detalladamente los datos de entrada, el resultado que se espera obtener y el resultado obtenido.

<u>Uso</u>: id. posibles fallos de implantación
calidad
usabilidad

- <u>Buena prueba</u> → probar si <u>SW hace</u> lo que tiene que hacer
  → probar si <u>SW no hace</u> " "
- <u>Errores humanos en desarrollo SW</u> → incorrecta especificación de los obj
  → errores producidos durante el proc. de diseño
  → errores q aparecen en la fase de desarrollo

↳ Implantar estrategia de pruebas

↳ PRUEBA DE UNIDAD → se analiza el cod. implementado

↳ PRUEBA DE INTEGRACIÓN → se presta atención al diseño y la construcción de la arq. de SW.

↳ PRUEBA DE VALIDACIÓN → se comprueba q el sist. construido cumple con lo establecido en el análisis de req. de SW.

↳ PRUEBA DE SISTEMA → verifica el funcionamiento total del SW y otros elementos del sist.

TIPOS → FUNCIONALES → evalúan cumplimiento de los requisitos
→ NO FUNCIONALES → " " aspectos adicionales → rendimiento ...
→ seguridad ...

PRUEBAS UNITARIAS

- OBJ → probar el correcto funcionamiento de un módulo de cod.
- FIN → cada modulo funciona bn x separado

1 UNIDAD = parte + peq. de la app. que se puede probar.

↳ prog. procedural = función / procedimiento

↳ prog. orientada obj = método

REQUISITOS → Automatizable
→ Completas (cobrir mayor cant cod)
→ Reutilizables
→ Independientes
→ Profesionales

las pruebas individuales nos dan SUTILAS → fomentan el cambio
→ simplifican la integración
→ documenta el código (se puede ver como utilizar el cod)
→ Separación interfaz - implementación
Errores + acotados y fáciles de localizar

PRUEBAS DE INTEGRACIÓN

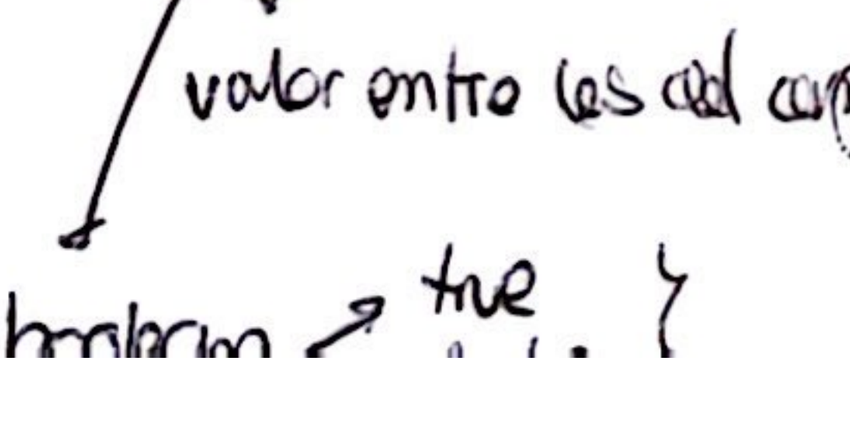
⇒ prueban finalmente todo el SW. en conj.

JUnit → framework → automatización de pruebas → unitarias
↳ herramientas, métodos... → integración

ESTRATEGIAS DE PRUEBA

↳ CAJA NEGRA / FUNCIONALES → sin saber estructura / funcionamiento interno

↳ solo se sabe entradas aceptadas / salidas



↳ clases de equivalencia → probar sólo un caso de cada clase, ya que los demás datos de la misma clase son equivalentes.

↳ identificación → rango = 3 clases → x encima / dentro / x debajo
valor concreto = 2 clases → ese valor / otro valor
valor entre los del conjunto → dentro / fuera
booleano → true

Frontera / valores Límite

↳ CAJA BLANCA / ESTRUCTURALES → OBJ → probar estructura interna del cod

↳ Cobertura → % de cod probado con las pruebas

↳ cod no usado
ejecutar todas las instrucciones

↳ comprobar caminos lógicos

segmentos → casos de prueba para que cada sentencia (sin condición) sea probada al menos una vez

De ramas → casos de prueba suficientes para que cada decisión se ejecute una vez con resultado ⊂ +

de condición/decisión → trocear las expresiones booleanas complejas en sus componentes e intentar cubrir todos los posibles valores de cada uno de ellos.

de bucles → ver si se entra 0/1/+ veces, y con las salidas break.

OTROS TIPOS DE PRUEBA
- RECORRIDOS (desarrolladores ven y comprenden)
- ALEATORIAS (estadística)
- REGRESIÓN (respecto a actualizaciones = NUEVOS errores)
- CARGA → SOLIDEZ → INTERACCIONES
  → AGUANTE → CONFORMIDAD / HOMOLOGACIÓN
  → INTEROPERABILIDAD
- ESTRÉS
- ESTABILIDAD
- RIESGOS

PRUEBAS DE VALIDACIÓN → integrar de forma secuencial el cliente

↳ se centra → descubrir errores → perspectiva de requisitos

↳ pruebas de caja negra

↳ PLAN DE PRUEBAS = pruebas realizadas ↳ PROCEDIMIENTO DE PRUEBAS → especifica casos

cumplir req. funcionales, rendimientos, portabilidad, compatibilidad...

↳ No se suele cumplir esta lista de requerimientos antes de la terminación planificada.

CONCLUSIONES

1) Probar = ejecutar programa xra encontrarle fallos. Jamás se deberá probar un programa con el ánimo de mostrar que funciona.
2) Caso de prueba tiene éxito cuando encuentra un fallo
3) Pruebas ↗ diseño ↘ ejecución y persona distinta
4) Diseñar empezar a pensarlas antes de terminar el programa
5) Módulo con muchos fallos = insistir sobre él (incluso reescribirlo)
6) Las pruebas no pueden demostrar que no hay fallos
7) Las pruebas también tienen fallos.

HERRAMIENTAS DE DEPURACIÓN

Una vez el cod. no pasa las pruebas → podemos utilizar el depurador (modo Debug)

↳ y así paso a paso ir corrigiendo

↳ Recomendable → breakpoints