

## TEMA 5

## PRUEBAS Y ENTREGA

### PRUEBAS DE CAJA BLANCA

Se centra en probar el comportamiento interno y la estructura del programa examinando la lógica interna:

Para ello

- se ejecutan todas las sentencias (al menos una vez)
- se recorren todos los caminos indep. de cada módulo.
- se comprueban todas las decisiones lógicas
- se comprueban todos los bucles.

Se buscan provocar situaciones extremas.

Se dividen en → TÉCNICAS

- pruebas de interfaz.
- pruebas de estructuras de datos locales.
- pruebas del camino básico
- pruebas de condiciones límite

### PRUEBAS DE INTERFAZ

Analizan el flujo de datos que pasa a través de la interfaz (tanto interna como externa) del módulo.

- Interfaces internas entre funciones
  - args de llamadas a func
  - consistencia var. globales

- Interfaces externas
  - se declaran los ficheros correctamente
  - se abren los ficheros correctamente
  - se abren antes / cierran después
  - condiciones fin de fichero
  - manejo condiciones E/S.

### PRUEBAS DE ESTRUCTURAS DE DATOS LOCALES

Aseguran la integridad de los datos durante todos los pasos de la ejecución del módulo.

- Ref. de datos: utilización de var. no inicializadas, salirse del límite de matrices/vectores.
- Declaración de datos: comprobar que las declaraciones de datos locales son correctas (longitud, tipo, ...)
- Cálculo: localiza errores de uso de variables (overflow, underflow, div. zero).
- Comparación: comprobar que no se hacen comparaciones entre variables de distinto tipo.

## PRUEBAS DE CAMINO BÁSICO

Definen un conjunto básico de caminos usando la complejidad ciclomática (complejidad del módulo)

Número de caminos básicos a probar:  $\text{Aristas} - \text{Nodos} + 2$   
 $\text{Nº reg. cerradas} + 1$

1. Dibujar grafo de flujo
2. Calcular Complejidad Ciclomática.
3. Determinar conjunto base de caminos lin. indep.
4. Preparar casos de prueba de cada camino

## PRUEBAS DE CONDICIONES LÍMITE

Validar la construcción de los bucles.

- Bucle simples (n es el máximo de pasos)

- pasar por alto el bucle
- pasar una sola vez
- pasar dos
- pasar m ( $m < n$ )
- pasar  $n-1$  y  $n+1$ .

- Bucle anidados

→ Comenzar con el bucle más interno (el resto a los valores mínimos)

## DISEÑO DE CASOS DE PRUEBA

Objetivo: crear el subconjunto de todos los posibles casos de prueba que tiene la mayor probabilidad de detectar el mayor nº posible de errores.

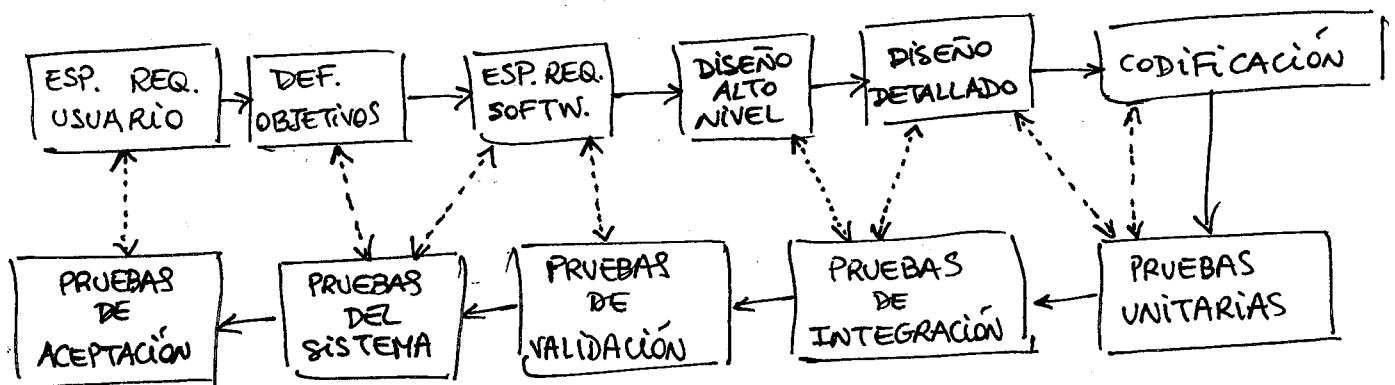
Se usan pruebas de caja negra y después examinar la lógica del programa (caja blanca).

Estrategia:

1. Valores límite
2. Partición de equivalencia
3. Combinar entradas + casos típicos de error.
4. Caja blanca.

## ESTRATEGIA DE PRUEBAS

- Pruebas unitarias: comprueban la lógica, funcionalidad y si es correcta la especificación de cada módulo.
- Pruebas de integración: comprueban la unión de módulos y su flujo de datos.
- Pruebas de validación: comprueban concordancia req. software.
- Pruebas del sistema: integración con su entorno hardware y software.
- Pruebas de aceptación: comprueban que el producto se ajusta a los req. de usuario.



PRUEBAS UNITARIAS : prueba de cada módulo y la realiza el programador en su entorno de trabajo.

- Ventajas
- el error está más localizado
  - se pueden probar simultáneamente varios módulos
- Enfoque de caja blanca
  - La prueba de unidad se simplifica cuando se ha diseñado un módulo con alto grado de cohesión.
  - se crean módulos para las pruebas
    - conductores o impulsores : especif. creado para la prueba que llama al módulo a probar.
    - resguardo o auxiliar : especif. creado para la prueba que es llamado por el módulo a probar.

PRUEBAS DE INTEGRACIÓN : consiste en integrar los módulos centrándose en probar sus interfaces.

- Enfoque de caja negra.
- Hay que determinar la manera en la que se combinan los distintos módulos

- no incremental (big bang)
- incremental

- ascendente : se integra de abajo arriba.  
se utilizan mód. conductores.
- descendente : se integra de arriba a abajo.  
se utilizan mód. de resguardo.
- sandwich : combina ascendente y descendente

• Obs : los módulos críticos son aquellos que está dirigido a varios req. software, nivel de control alto, complejo, propenso a errores o tiene req. de alto rendimiento.

PRUEBAS DEL SISTEMA : prueba el sistema integrado e su entorno hardw. y softw.

- Verificación de que el sistema integrado cumple los requisitos especificados.
- Consta de varias subpruebas: de interfaces externas, de volumen, de recuperación, de seguridad...

PRUEBAS DE VALIDACIÓN : comprueba que se cumplen los req.

- Los criterios de validación se han acordado en la fase de def. de req.
- Técnicas de caja negra.
- Dos partes  $\begin{cases} \rightarrow \text{validación por parte del usuario} \\ \rightarrow \text{utilidad, usabilidad y ergonomía.} \end{cases}$
- Dos tipos  $\begin{cases} \rightarrow \text{alfa: cliente en lugar de desarrollo} \\ \rightarrow \text{beta: cliente en su entorno} \end{cases}$

PRUEBAS DE ACEPTACIÓN : último paso antes de la entrega.

- Consiste en la aceptación por parte del cliente del softw.
- El usuario aporta los casos de prueba.

## TEMA 6 MANTENIMIENTO

DEF: El mantenimiento de un producto software comprende la modificación de dicho producto después de haber sido entregado a los clientes con el fin de corregir defectos, mejorar su rendimiento o atributos, o adaptarlo a un cambio en su entorno.

Conjunto de actividades que se realizan sobre el software una vez que este está operativo.

Llevar a cabo el mantenimiento de forma planificada facilita el mismo y reduce su coste.

### TIPOS DE MANTENIMIENTO

- CORRECTIVO ( $\approx 20\%$ ): corrige errores
- ADAPTATIVO ( $\approx 25\%$ ): acomodar a nuevo entorno
- PERFECTIVO ( $\approx 50\%$ ): mejorar y expandir req. implementados
- PREVENTIVO ( $\approx 5\%$ ): prevenir errores
- ESTRUCTURAL: modificar arquitectura interna

MANT. CORRECTIVO: modificar el sistema tras la detección de defectos, ambigüedades o errores.

Se realiza: diagnóstico de errores y corrección de errores

Tipos: de emergencia o planificado.

MANT. ADAPTATIVO : modificar el sistema para acomodarlo a cambios físicos del entorno.

Incluye:

- actividades para adaptar el software a un entorno nuevo (hw, sw)
- actividades para añadir nuevos periféricos
- actividades para adaptar el software a cambios en fuentes externas.

MANT. PERFECTIVO : mejorar el sistema para cumplir con las nuevas necesidades/requisitos de los usuarios o negocio.

Incluye:

- mejorar el software para aumentar la eficiencia, rendimiento...
- actividades necesarias para cumplir nuevos requisitos relativos a fuentes externas.

MANT. PREVENTIVO : modificar el sistema con los cambios necesarios para mantener la eficiencia y fiabilidad del software.

Incluye:

- revisión periódica de equipos, periféricos y protocolos asociados.
- pruebas y revisiones periódicas
- aumentar el tamaño previsto ficheros / BD.

o MANT. ESTRUCTURAL : modificar la arquitectura interna del sistema para mejorar su mantenibilidad.

Incluye:

- mejorar documentación
- reestructurar código
- efectuar reingeniería modularidad para aumentar su

# REINGENIERÍA

## MOTIVOS

- Mejorar el mantenimiento
- Mejorar los sistemas sw actuales
- En sistemas que se usan mucho, se van a seguir usando o se está realizando mantenimiento.

DEF: proceso de examinar un sistema software existente y reconstruirlo con ayuda de herramientas automáticas para mejorar su mantenibilidad y comprensión, incrementar su calidad y aumentar su vida.

## TIPOS

- RE-ESTRUCTURACIÓN: proceso de cambiar el código modificando su forma pero no funcionalidad.
- MIGRACIÓN: proceso de convertir un sistema software de un lenguaje a otro o portarlo de una plataforma a otra.
- INGENIERÍA-INVERSA: proceso de recobrar una descripción de más alto nivel de un sist. sw a partir de una exp. de bajo nivel.  
La ing. inversa no cambia lo que hace el su

¿Cuándo aplicar reingeniería?

- Gran importancia para la empresa.
- Fallos frecuentes
- Problemas de rendimiento
- Difícil de modificar / probar
- Frecuente mantenimiento
- Caro de mantener



## ESTRATEGIAS DE MANTENIMIENTO

- MANT. ESTRUCTURADO / PLANIFICADO : se acumulan los cambios mejoras principalmente, y se realizan todos a la vez en fechas planificadas.
- MANT. NO ESTRUCTURADO : bajo petición ; cada vez que se demanda el mantenimiento.
- COMBINADO : estructurado y cuando hay emergencia. Suele ser la más beneficiosa para la empresa.

## TEMA 7

## GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE

DEF. (Config. del software) : Es el estado actual del sistema software y las interrelaciones entre sus componentes constitutivos (código, datos y documentación).

DEF (Gestión config. del software) : Disciplina cuya misión es identificar, controlar y organizar la evolución de un sistema software. Permite controlar formalmente la evolución y los cambios del software, garantizando la visibilidad en el desarrollo y en el producto, y la trazabilidad en el producto durante todo su ciclo de vida.

La GCS es necesaria debido a la larga vida del software, los cambios del mismo a lo largo de su vida y asegurar mínimo coste en los cambios.

La GCS actúa sobre programas, datos y documentación.

DEF. (Elem. Config. del Software) : Cada uno de los elementos básicos de un producto software sobre los que se realizará un control. Tiene un nombre y puede evolucionar. Cumple que

- evoluciona en el tiempo
- nos interesa su evolución

LÍNEA BASE: Es una configuración de referencia en el proceso de desarrollo del software a partir de la cual las revisiones de los elementos de configuración del software se han de realizar de manera formal.

Objetivo: controlar los cambios del software, sin impedir llevar a cabo aquellos que sean justificados.

Se definen al principio del proyecto, coincidiendo con los hitos marcados. Generalmente se corresponden con los resultados de las fases.

Tipos (más habituales): LB Funcional, LB Diseño, LB Producto, LB Operación.

Objetivos secundarios:


- Identificar resultados de las tareas de cada fase.
- Asegurar que se ha completado la fase.
- Servir como punto de partida para desarrollos posteriores.
- Servir como punto de partida para las peticiones de cambio.

Una vez que se ha desarrollado y revisado un elemento de configuración, pasa a formar parte de la siguiente línea base planificada del proyecto. Es decir, el elemento se convierte en línea base.

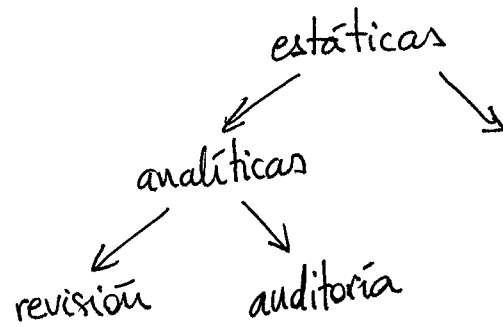
Cuando un ECS se convierte en una LB se introduce en una BD del proyecto. A partir de aquí el ECS se debe modificar siguiendo un procedimiento establecido. El objetivo es que se puedan hacer los cambios necesarios pero de manera controlada y previa autorización.

Los cambios sobre un elemento de una LB producen la creación de una nueva versión del elemento.

Diferencias entre:  
pruebas de condiciones límite  
pruebas de condición  
pruebas de decisión  
pruebas de condición/decisión

7/7/17 | How | Pa  


Revisión } med. estáticas  
Auditoría }



Revisión (interna): detectar defectos  
pueden ser en cualquier momento (normalm. hitos)

Auditoría (externa): certificar conformidad (todo se ha hecho  
lo que se ha dicho)  
suelen ser periódicas

decisión: dos caminos

decisión/condición: una o muchas condiciones

b)

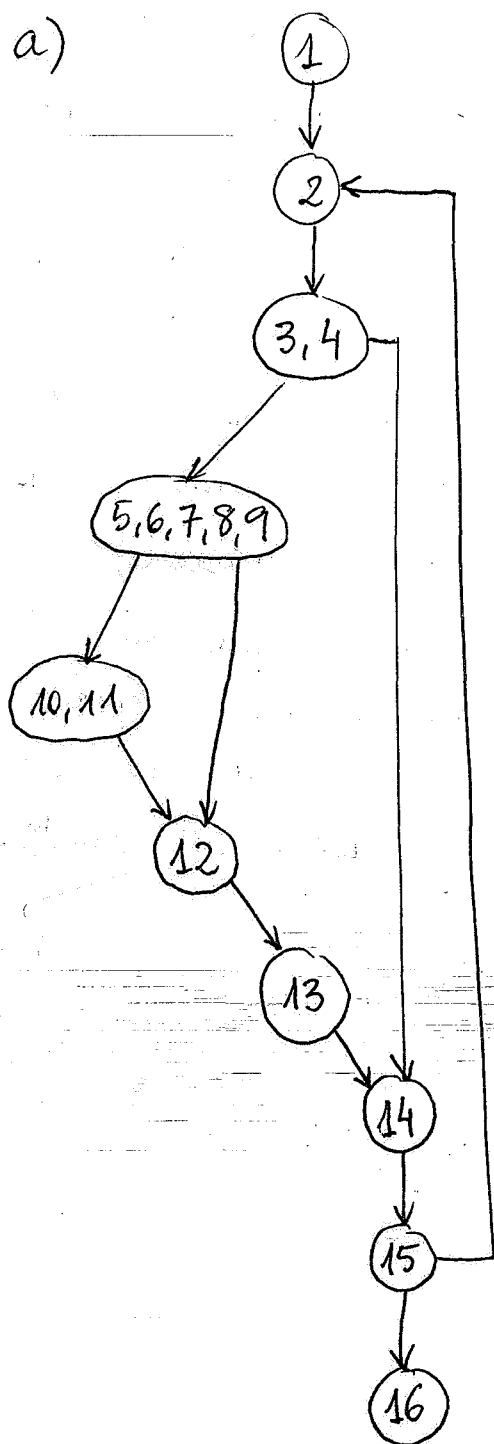
ATRIBUTO	CLASES VÁLIDAS	CLASES NO VÁLIDAS
Usuario	<p>1. Cadena de caracteres que valide la siguiente expresión regular:</p> $^{\wedge}[a-zA-Z][a-zA-Z ]{8}\#$ <p style="text-align: center;"> <math>\uparrow</math>              030! espacio en blanco           </p>	<p>2. Cadena de más de 10 caracteres.</p> <p>3. Cadena de menos de 10 caracteres.</p> <p>4. Cadena con caracteres no permitidos (signos de puntuación)</p> <p>5. Cadena que no valide la expresión regular aún teniendo 10 caracteres y todos ellos permitidos</p>

CASOS DE PRUEBA CAJA NEGRA:

prueba_id	Usuario	Salida
1	abcdefghijkl#K	-1
2	abcdefghi	-3
3	abcde.fgi#	-2
4	abcdefghij	-3
5	acdefghi#	0

4.

a)



complejidad ciclomática : aristas - nodos + 2

$$\begin{array}{c} 12 \\ 12 \end{array} - 10 + 2 = \boxed{4}$$

$\Rightarrow$  complejidad ciclomática = 4