Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

Prueba del Software

- La etapa de Prueba no es la primera instancia donde se encuentran defectos.
- Se ha visto que la revisión de requerimientos y diseño del sistema contribuyen a descubrir problemas en etapas tempranas.

➢ ¿Qué significa que el software ha fallado?
  - No hace lo que especifican los requerimientos del cliente.

➢ Posibles razones:
  - Especificación de requerimientos errónea.
  - Requerimientos imposibles de aplicar con las estructuras previstas.
  - Defectos en el diseño del sistema.
  - Defectos en el diseño del programa.
  - Defectos en el código.

Tipos de defecto

➢ Algorítmico
  - Ej: no inicializar variables.

➢ De sintaxis
  - Ej: confundir un 0 con una O (tenés que ser muy idiota para esto).

➢ De precisión
  - Ej: fórmulas matemáticas implementadas erróneamente.

➢ De documentación
  - Ej: la documentación no es acorde a lo que hace el software.

➢ De sobrecarga
  - Ej: el sistema funciona bien con 100 usuarios pero no con 110.

➢ De capacidad (de almacenamiento i guess)
  - Ej: el sistema funciona bien con cantidad de ventas < 1.000.000.

➢ De coordinación o sincronización
  - Ej: comunicación con fallas entre procesos.

➢ De rendimiento
  - Ej: tiempo de respuesta inadecuado.

➢ De recuperación
  - Ej: no volver a un estado normal luego de una falla.

➢ De relación hardware-software
  - Ej: Incompatibilidad entre los componentes.

➢ De estándares
  - Ej: no cumplir con definiciones estandarizadas y procedimientos.

Clasificación ortogonal de Defectos

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

- Los defectos se han organizado en categorías.
- Primero, identificamos cual de los dos defectos siguientes es:
  - Defecto por omisión -> cuando falta un aspecto clave del código.
    Por ej: una variable no inicializada.
  - Defecto de cometido -> cuando algún aspecto del código es incorrecto.
    Por ej: variable que se inicializa con un valor erróneo

Tipo de defecto	Significado
Función	Afecta la capacidad, interfaces.
Interfaz	Afecta a la interacción con otros componentes.
Comprobación	Afecta la lógica del programa.
Asignación	Afecta la estructura de datos.
Sincronización	Involucra sincronización de recursos compartidos y de tiempo real.
Construcción	Ocurre debido a problemas en repositorios, gestión de cambios o control de versiones.
Documentación	Afecta a publicaciones.
Algoritmo	Involucra la eficiencia o exactitud de un algoritmo.

Prueba del Software

- ¿Objetivo primario?
  Diseñar pruebas que saquen a la luz diferentes clases de errores, haciéndolo en la menor cantidad de tiempo y esfuerzo.
- ¿Cuándo una prueba tiene éxito?
  Cuando descubre errores.

Objetivos y Beneficios de estas pruebas del software

- Descubrir errores antes de que el software salga del ambiente de desarrollo.
- Detectar un error que no haya sido descubierto hasta ese momento.
- Bajar los costos de corrección de errores en la etapa de mantenimiento.

Principios de la Prueba

- A todas las pruebas se les debe poder hacer un seguimiento hasta los requisitos del cliente.
- Las pruebas deberían ser planificadas mucho antes de que empiecen a entrar en acción.
- Se aplica el principio de Pareto:

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

- “El 80% de los errores de un software es generado por un 20% del código de dicho Software, mientras que el otro 80% del código genera tan sólo el 20% restante de los errores”.

➢ Las pruebas deben empezar por lo pequeño y progresar a lo grande.
➢ Es importante asegurarse que se han probado todas las condiciones a nivel de componente.
➢ Para lograr más eficacia, las pruebas deberían ser realizadas por un equipo independiente al desarrollo.

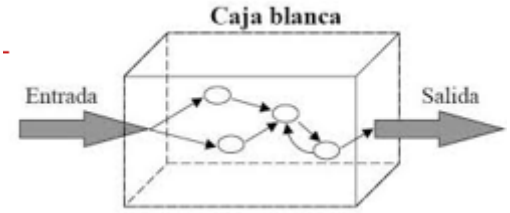
¿Quién realizará las pruebas?

- Varios factores justifican la existencia de un equipo independiente que haga el testing:
  - Evita el conflicto entre la responsabilidad por los defectos y la necesidad de descubrir defectos.
  - Llevar a cabo las pruebas de forma concurrente con la codificación.
  - Los desarrolladores colaboran y corrigen los errores.

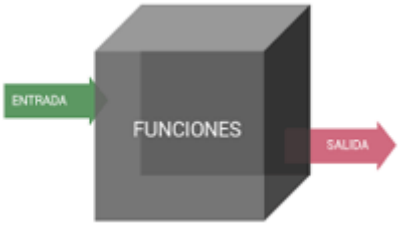
➢ La prueba no asegura la ausencia de defectos.
➢ Por eso se buscan casos de prueba que permitan hallar errores.

Tipos de Prueba del Software

➔ La prueba de **caja blanca** es basada en el minucioso examen de los detalles de los procedimientos. De forma que se comprueban los caminos lógicos de ejecución del software proponiendo casos de prueba que ejercitan conjuntos específicos de condiciones y/o bucles.



➔ La prueba de **caja negra** se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Es menos minuciosa.



Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

Prueba de caja negra (o cerrada)

- También como prueba de comportamiento, se centran en los requisitos funcionales del software.
- Intenta descubrir diferentes tipos de errores que los métodos de caja blanca.
- ¿Qué errores busca?
    - Funciones incorrectas o ausentes.
    - Errores de la interfaz.
    - Errores en estructuras de datos o en accesos a BD.
    - Errores de rendimiento.
    - Errores de inicialización y de terminación.
    - Concentra la prueba en el dominio de información: por ejemplo pruebas de partición equivalente.

Prueba de la partición equivalente

- El diseñado de casos de prueba consiste en evaluar las clases de equivalencia para una condición de entrada.
- Clase de equivalencia: representa un conjunto de estados válidos o no válidos para condiciones de entrada.
- Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores que se relacionan o una condición lógica.

Partición equivalente – definición

- ➔ Si una condición de entrada es un **rango**, se define una clase de equivalencia válida y dos no válidas.
- ➔ Si una condición de entrada requiere un **valor específico**, se define una clase de equivalencia válida y dos no válidas.
- ➔ Si una condición de entrada especifica un elemento de un **conjunto**, se define una clase de equivalencia válida y una no válida.
- ➔ Si una condición de entrada es **lógica** , se define una clase de equivalencia válida y una no válida.
- ➔ Resumiendo:
    - Rango o valor específico: 1 eq válida, 2 no válidas.
    - Conjunto o lógica: 1 eq válida, 1 no válida.

Ejemplo de Partición Equivalente, Alta de juguete

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

Dato	Tipo
Código	entero positivo
Nombre	string 20
descripción	string 256
Recomendaciones	string 512
Genero	enumerativo (masculino, femenino)
Edad	rango 0..120
Marca	string 25
Stock	entero
stock mínimo	entero positivo
Estado	enumerativo (normal, oferta, novedoso)

Condición de entrada	*Clases de equivalencia válidas*	Clases de equivalencia inválidas
código	0<= código <= 9999	código <0 código > 9999
nombre	1 a 20 caracteres	0 caracteres; mas de 20 caracteres;
descripción	0 a 256 caracteres	mas de 256 caracteres
recomendaciones	0 a 512 caracteres	mas de 512 caracteres
genero	masculino femenino	otra cadena de caracteres;
stock	Número entero	Caracteres no dígitos;

Análisis de Valores Límite (AVL)

- Los errores tienden a darse más en los límites del campo de entrada que en el centro.
- Por ello se ha desarrollado el análisis de valores límite como una técnica de prueba.
- **Dicho análisis complementa la partición equivalente**. En lugar de elegir cualquier elemento de una clase de equivalencia, el ALV selecciona los casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.

Caja negra: AVL

- Casos de prueba en los bordes de las clases:

- Para condición de entrada de rango entre valor a y valor b, probar: a, b, < a, > b.
- Para una salida de rango entre a y b: utilizar casos de prueba que generen un valor de salida a y b.
- Probar las estructuras de datos internas en sus límites.

Caja blanca (o cristal o abierta)

- Deriva casos de prueba de la estructura de control para verificar detalles del procedimiento.
- El ing de software obtendrá casos de prueba que le garanticen que se ejercita por lo menos una vez, todos los caminos independientes de cada módulo.
  - Ejercitación de todas las decisiones lógicas.
  - Ejecución de todos los búcles en sus límites.
  - Ejercitación de todas las estructuras internas de datos para asegurar su validez.
- El flujo lógico de un programa a veces no es para nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos puede derivar a tener errores de diseño que sólo se encuentran cuando comienza la prueba del camino de (ya veremos que verga es esto).
- ¿Por qué realizar estas pruebas?
  - Los casos especiales son los más factibles de error.
  - Los errores tipográficos con aleatorios.
  - El flujo de control intuitivo es distinto del real.

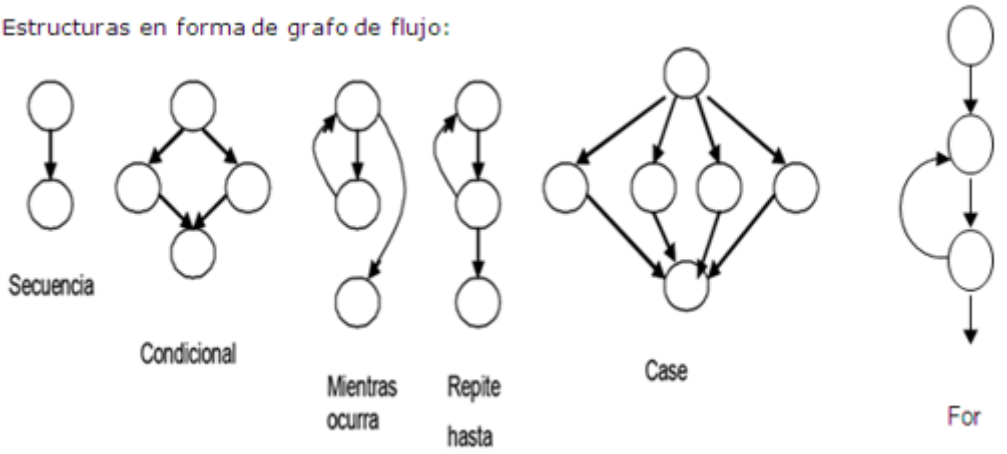
Prueba del camino básico

- Técnica propuesta por Tom McCabe. Permite al diseñador de casos de pruebas obtener una medida de la complejidad lógica y usarla como guía para la definición de caminos de ejecución.
- Los casos de prueba que se obtienen garantizan que se ejecuta al menos una vez cada sentencia del programa.

Camino básico – Notación de Grafo

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

Estructuras en forma de grafo de flujo:

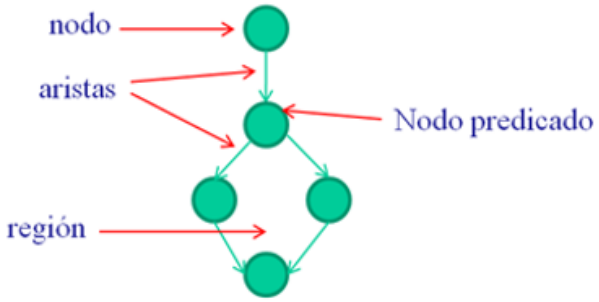


»Cada círculo, denominado **nodo** del grafo de flujo, representa una o más sentencias procedimentales.

»Las flechas del grafo de flujo, denominadas **aristas** o representan flujo de control.

»Una arista debe terminar en un nodo.

Cada nodo que contiene una condición se denomina nodo predicado y está caracterizado porque dos o más aristas emergen de él.



Las áreas delimitadas por aristas y nodos se denominan regiones. Cuando contabilizamos las regiones incluimos el área exterior del grafo, contándolo como otra región más.

»La **complejidad ciclomática**

» es una **métrica del software** que proporciona una medición cuantitativa de la complejidad lógica de un programa.

»**define el número de caminos independientes** del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

»**Un camino independiente** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

»Complejidad ciclomática :

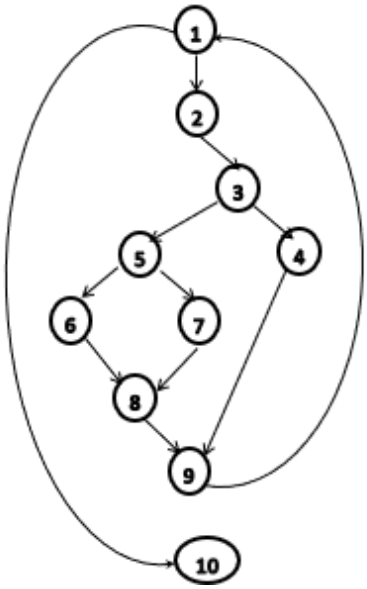
1. *V(g)=Cantidad de regiones del grafo ó*
2. *V(g)= A - N + 2 ó*
3. *V(g)= P + 1*

»La complejidad ciclomática **debe medirse con las tres formulas** de manera de verificar su exactitud.

Pasos para crear la prueba

1. Dibujar el grafo de flujo correspondiente.
2. Determinar la complejidad ciclomática.
3. Determinar un conjunto básico de caminos independientes.
4. Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto.
5. Ejecutar cada caso de prueba y comparar los resultados obtenidos con los esperados.

```
Procedure Ordenar
(1) do while not eof() begin
 (2)   leer registros;
(3)   if (campo 1 de registro = 0)
(4)     then procesar registro
            Guardar en buffer;
            Incrementar contador;
(5)     else  if (campo 2 de registro = 0)
(6)                 then reiniciar contador
(7)  else
            procesar registro;
            Guardar en archivo;
         (8)  endif
 (9)  endif
(10)   end;
```



Fuente:

Ingeniería de Software 2 : Clase 10 – Tipos de Pruebas

Regiones : 4
Nodos : 10
Aristas : 12
Nodos Predicados : 3

V(G) : A-N+2 = 12-10+2 = 4
V(G) : NP+1 = 3+1 =4
V(G) : R = 4

P = Predicado
Rx = Región X