ENTORNOS DE DESARROLLO

IES Santiago Hernández
Curso 2017-2018
Ignacio Agudo Sancho

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Diseño y Realización de Pruebas

- Las pruebas de software consisten en verificar y validar un producto software antes de su puesta en marcha.
- Consiste en probar la aplicación construida.
- Se integran dentro de la vida del software.
- Se han considerado como una etapa dentro del desarrollo de software aunque la tendencia está cambiando puesto que se integran en la etapa de desarrollo.

Diseño y Realización de Pruebas

- La ejecución de pruebas involucra una serie de etapas:
  - Planificación de las pruebas
  - Diseño y construcción de los casos de prueba
  - Definición de los procedimientos de prueba
  - Ejecución de las pruebas
  - Registro de resultados obtenidos
  - Registro de errores encontrados
  - Depuración de los errores
  - Informe de los resultados obtenidos

6. Diseño y Realización de Pruebas

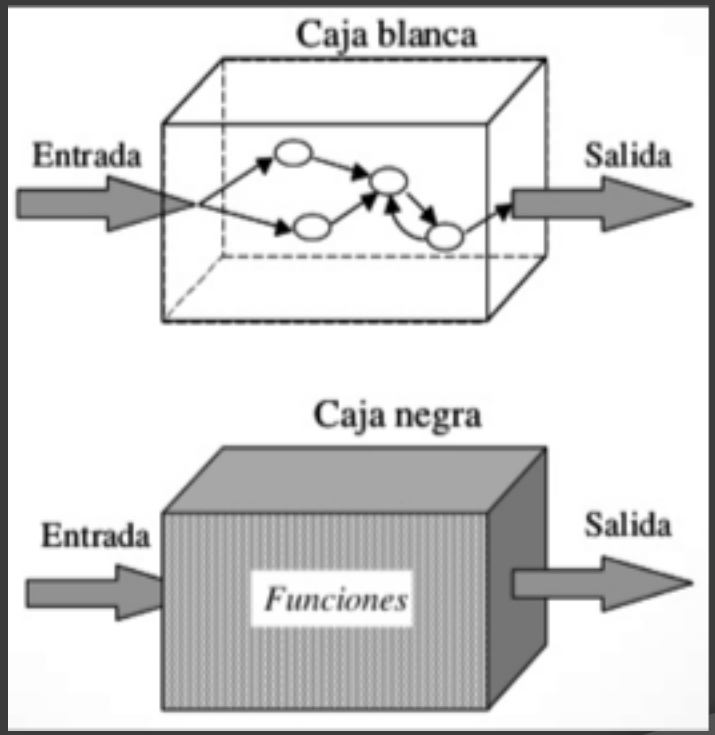
- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Técnicas de Diseño de Casos de Prueba

- Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba.
- Para llevar a cabo un caso de prueba es necesario definir las precondiciones y post condiciones, identificar unos valores de entrada y conocer el comportamiento que debería tener el sistema ante dichos valores.
- Tras realizar ese análisis e introducir los datos, se observa si el comportamiento es el previsto o no y por qué. Determinamos con esto si el sistema ha pasado la prueba o no.

Técnicas de Diseño de Casos de Prueba

- Existen dos técnicas para diseñar casos de prueba:



Técnicas de Diseño de Casos de Prueba

- Pruebas de caja blanca
  - Se basan en examinar minuciosamente los detalles procedimentales del código de la aplicación.
  - Mediante esta técnica se pueden obtener casos que:
    - Garanticen que se ejecutan al menos una vez todos los caminos independientes de cada módulo.
    - Ejecuten todas las sentencias al menos una vez.
    - Ejecuten todas las decisiones lógicas en su parte verdadera y en su parte falsa.
    - Ejecuten todos los bucles en sus límites.
    - Utilicen todas las estructuras de datos internas para asegurar su validez.

Técnicas de Diseño de Casos de Prueba

- Pruebas de caja negra
  - Se llevan a cabo sobre la interfaz del software. No hace falta conocer la estructura interna del programa ni su funcionamiento.
  - Pretende obtener casos de prueba que demuestren que las salidas que devuelve la aplicación son las esperadas.
  - El sistema se considera como una caja negra cuyo comportamiento solo se puede determinar estudiando las entradas y salidas que devuelve en función de las entradas suministradas.

Técnicas de Diseño de Casos de Prueba

- Pruebas de caja negra
  - Con este tipo de pruebas se pretende encontrar errores como por ejemplo:
    - Funcionalidades incorrectas o ausentes.
    - Errores de interfaz.
    - Errores en estructuras de datos o en accesos a bases de datos externas.
    - Errores de rendimiento.
    - Errores de inicialización y finalización.

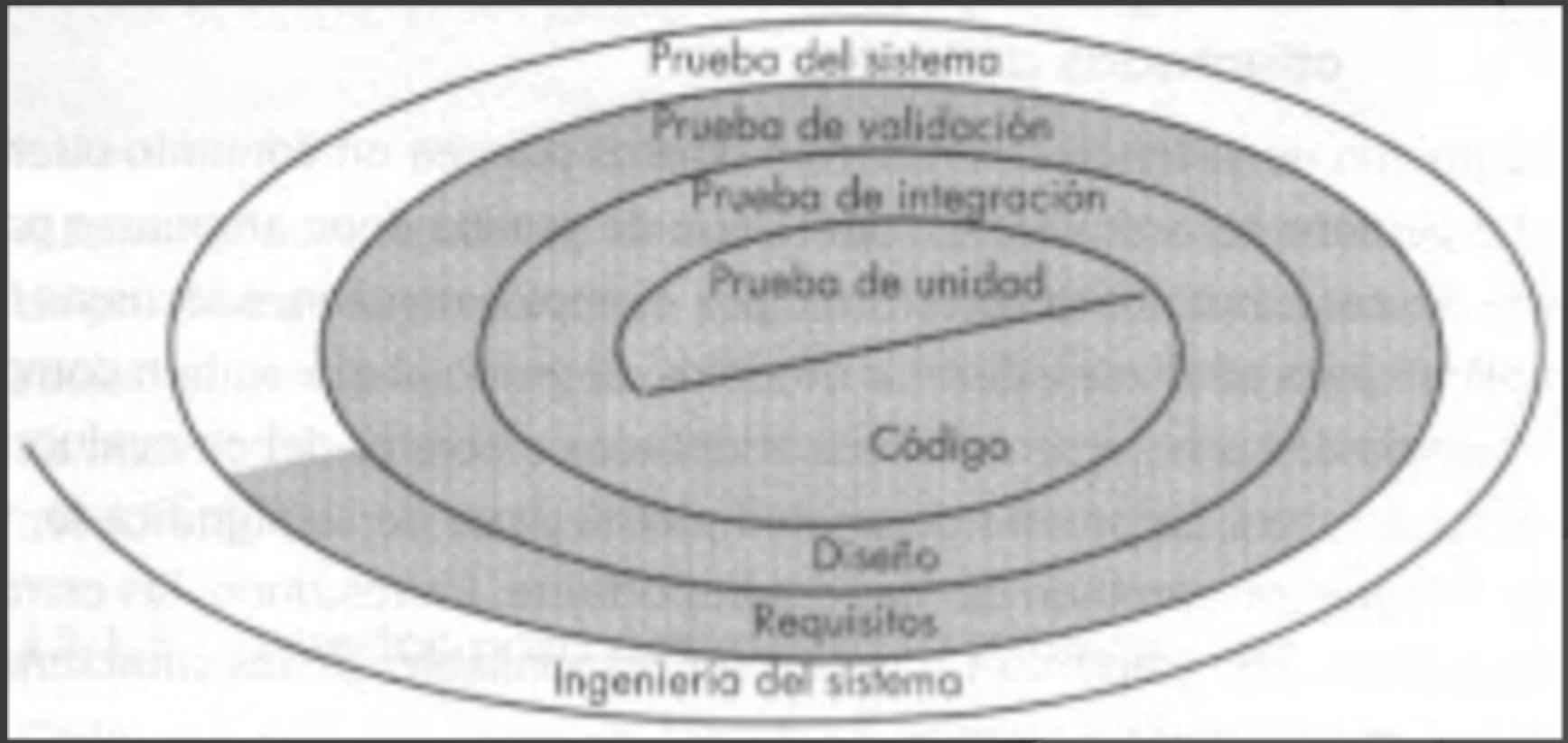
6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Estrategias de Pruebas de Software

- Pruebas de unidad: se centran en la unidad más pequeña de software, el módulo.
- Pruebas de integración: se construye con los módulos una estructura de programa tal como dicta el diseño.
- Prueba de validación o aceptación: prueba del software en el entorno real de trabajo por el usuario final. Se validan los requisitos establecidos.
- Prueba del sistema: verifica que cada elemento encaja de forma adecuada y se alcanza la funcionalidad y rendimiento total.

Estrategias de Pruebas de Software



Estrategias de Pruebas de Software

- PRUEBA DE UNIDAD (unitaria)
  - Se trata de probar cada módulo para eliminar errores en la interfaz y en la lógica interna.
  - Utiliza técnicas de caja blanca y caja negra.
  - Se realizan pruebas sobre:
    - La interfaz del módulo, para asegurar el flujo correcto.
    - Las estructuras de datos locales, para asegurar que mantienen la integridad.
    - Las condiciones límite.
    - Todos los caminos independientes, para asegurar que todas sentencias se ejecutan al menos una vez.
    - Todos los caminos de manejo de errores.

Estrategias de Pruebas de Software

- PRUEBA DE INTEGRACIÓN
  - Se observa cómo interaccionan los distintos módulos.
  - Dos enfoques para estas pruebas:
    - Integración no incremental o big bang: Se combinan todos módulos a la vez y se prueba el programa completo. Generalmente se encuentran muchos errores y la corrección es difícil.
    - Integración incremental: El programa completo se va construyendo y probando en pequeños segmentos, así es más fácil localizar los errores. Puede ser de dos tipos:
      - Ascendente: la construcción y prueba empieza desde los niveles más bajos de la estructura del programa.
      - Descendente: la integración comienza en el módulo principal.

Estrategias de Pruebas de Software

- PRUEBA DE VALIDACIÓN
  - La validación se consigue cuando el software funciona de acuerdo con las expectativas y requisitos.
  - Se llevan a cabo una serie de pruebas de caja negra:
    - Prueba Alfa: por el usuario o cliente en el lugar de desarrollo. El cliente usa el software y el desarrollador mira, apuntando los errores y problemas de uso.
    - Pruebas Beta: por los usuarios finales en su lugar de trabajo. Sin el desarrollador delante. El usuario registra los problemas que encuentra (reales o imaginarios) e informa al desarrollador para que lo modifique y arregle.

Estrategias de Pruebas de Software

- PRUEBA DEL SISTEMA
  - Es un conjunto de pruebas para ejercitar el software:
    - Prueba de recuperación: se fuerza el fallo y se verifica que la recuperación se lleva a cabo apropiadamente.
    - Prueba de seguridad: intenta verificar que el sistema está protegido contra accesos ilegales.
    - Prueba de resistencia (stress): trata de enfrentar al sistema con situaciones que exigen gran cantidad de recursos (máximo de memoria, gran frecuencia de datos de entrada, problemas en un sistema operativo virtual…)

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Documentación para las pruebas

- Existen una serie de documentos que se han de producir durante el proceso de pruebas:
  - Plan de pruebas: describe el alcance, el enfoque, los recursos y el calendario de las actividades de prueba. Identifica los elementos a probar, las características a probar, las tareas que se van a realizar, quién las va a hacer, y los riesgos asociados al plan.
  - Informes de pruebas: cuatro documentos: un informe que identifica los elementos que están siendo probados, un registro de las pruebas (donde se registra lo que ocurre durante la ejecución de la prueba), un informe de incidencias de prueba (cualquier evento que se produzca durante la ejecución de la prueba que requiera mayor investigación) y un informe resumen de las actividades de prueba.

Documentación para las pruebas

- Especificaciones de prueba: formadas por tres documentos:
  - La especificación del diseño de la prueba: requisitos, casos de prueba y procedimientos necesarios para llevar a cabo las pruebas, y los criterios de pasa no-pasa
  - La especificación de los casos de prueba: documenta los valores reales utilizados para la entrada, junto con los resultados previstos.
  - La especificación de los procedimientos de prueba: se identifican los pasos necesarios para hacer funcionar el sistema y ejecutar los casos de prueba especificados.

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Pruebas de código

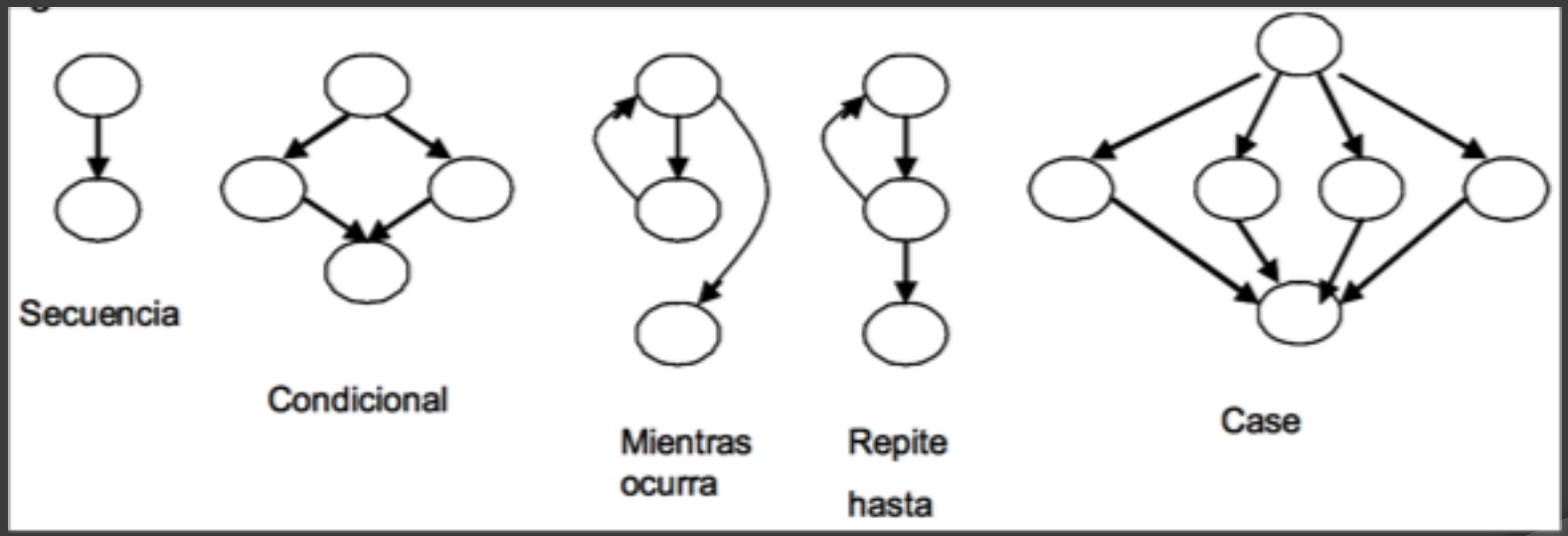
- Consiste en ejecutar el programa o parte de él con el objetivo de encontrar errores.
- Se parte de un conjunto de entradas y una serie de condiciones de ejecución.
- Se observan y registran los resultados y se comparan con los resultados esperados.
- Se observará si el comportamiento del programa es el previsto o no y por qué.
- A continuación vemos algunas técnicas para las pruebas de código:

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - Es una técnica de prueba de caja blanca que permite al diseñador de las pruebas:
    - Obtener una medida de la complejidad lógica de un diseño procedimental.
    - Usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.
  - Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.
  - Para la obtención de la medida de complejidad lógica (o ciclomática) emplearemos una representación del flujo de control denominada grafo de flujo o del programa.

Pruebas de código

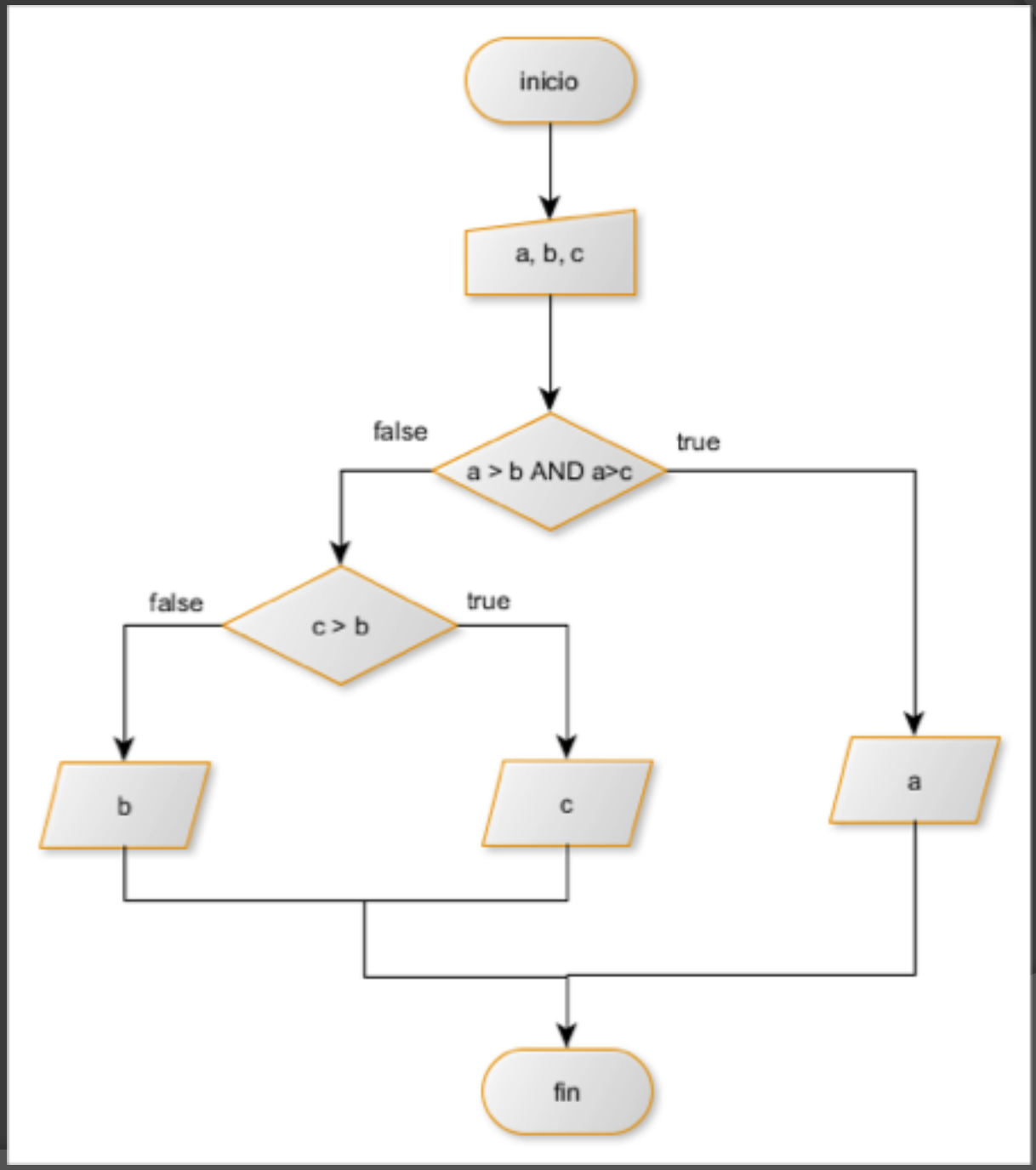
- PRUEBA DEL CAMINO BÁSICO
  - Notación de grafo de flujo



  - Cada círculo representa una o más sentencias, sin bifurcaciones.

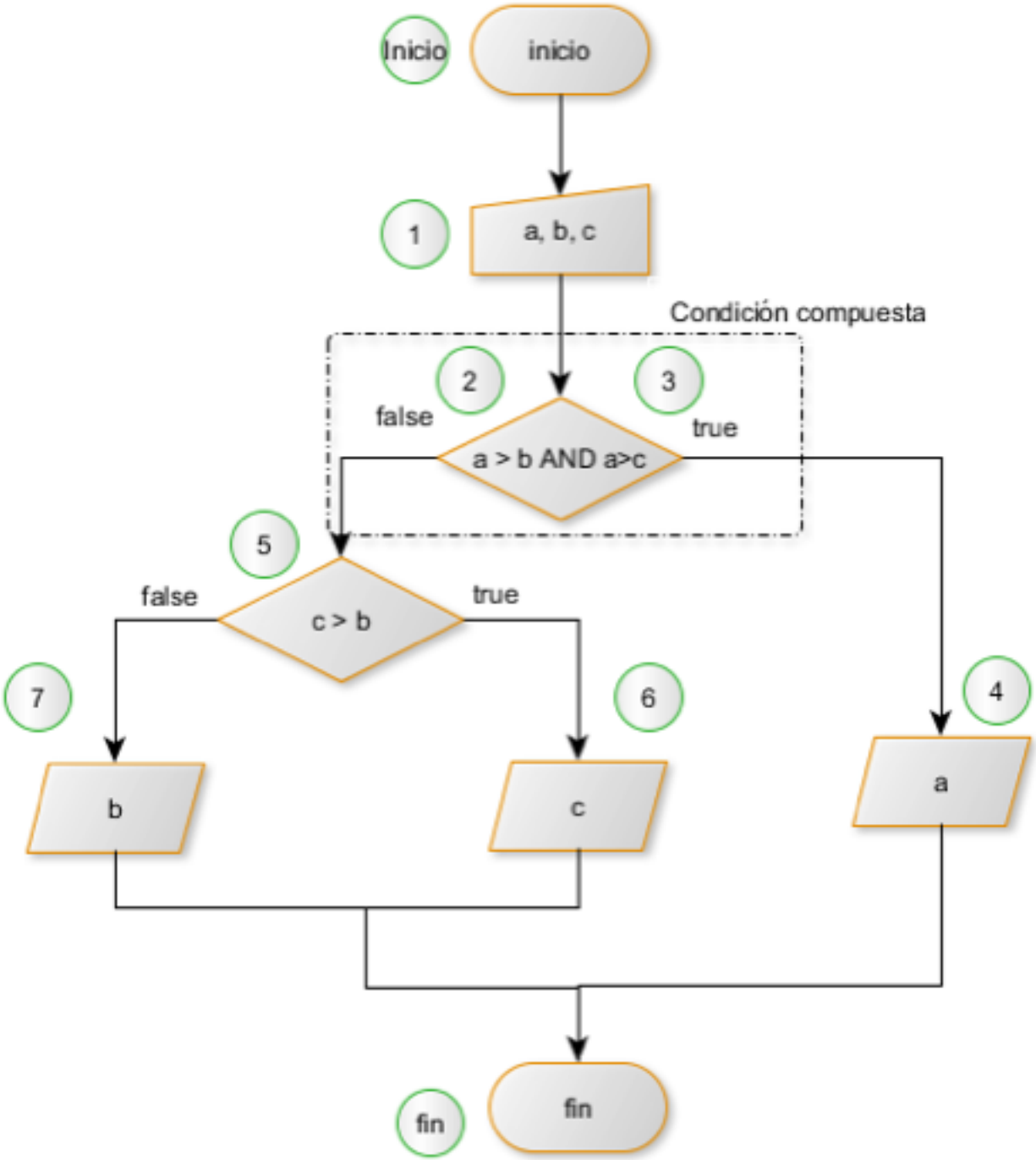
Pruebas de código

- PRUEBA DEL CAMINO BÁSICO (EJEMPLO)
  - En primer lugar necesitamos tener claro el diagrama de flujo de la aplicación.
  - A continuación se muestra un diagrama de flujo de un programa que determina el mayor de tres valores dados:


inicio
a, b, c
false
a > b AND a>c
true
false
c > b
true
b
c
a
fin

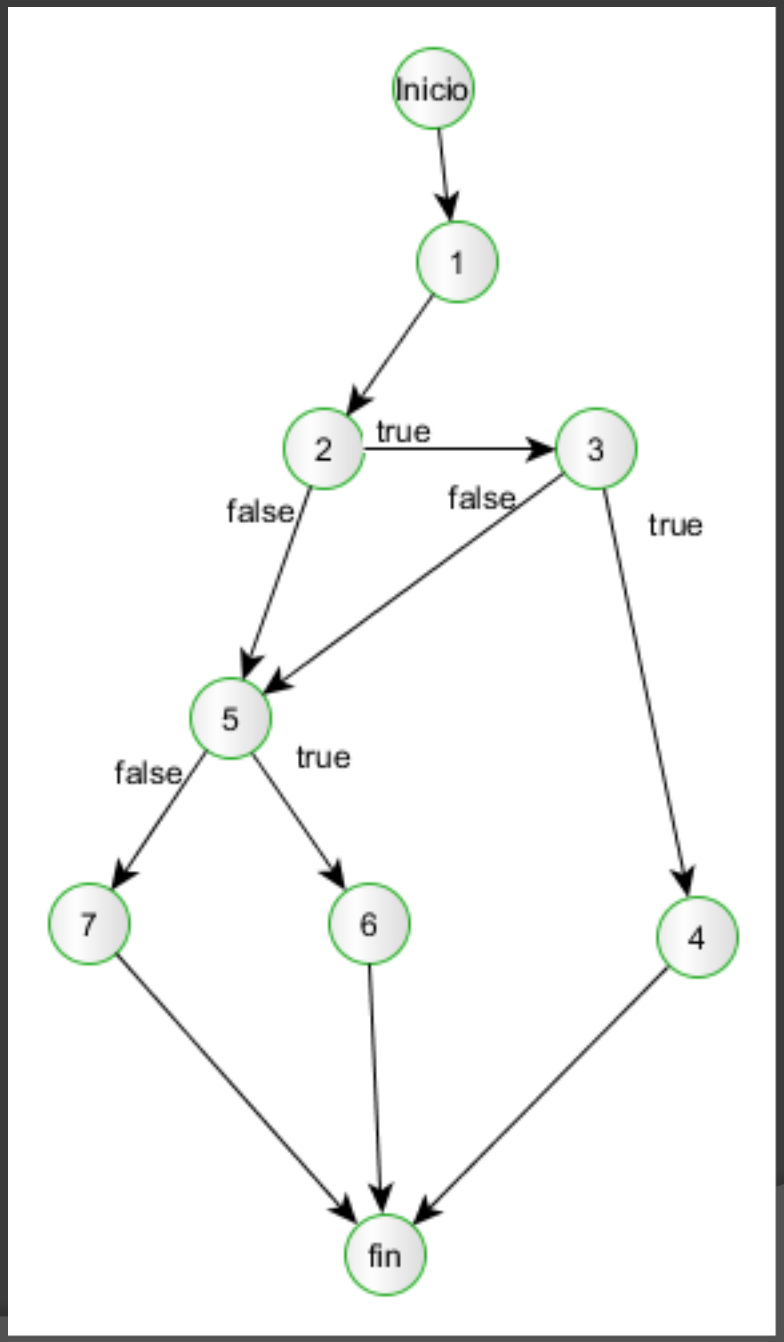
Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - A partir del diagrama de flujo generaremos un GRAFO DE FLUJO para determinar los círculos (nodos) y las flechas (aristas o enlaces) entre ellos.
  - Los NODOS representan una o más sentencias. Un solo nodo se puede corresponder con una secuencia de símbolos del proceso y un rombo de decisión.
  - Las ARISTAS o ENLACES representan el flujo de control. Una arista termina en un nodo.
  - Detectamos los nodos que conformarán el grafo de flujo así como los caminos que se pueden recorrer durante la ejecución del programa:


Inicio
inicio
1
a, b, c
Condición compuesta
2
3
false
true
a > b AND a>c
5
false
c > b
true
7
6
4
b
c
a
fin
fin

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - Si tenemos una condición compuesta, como es nuestro caso (a>b AND a>c), debemos descomponerla creando un nodo para cada una de las condiciones.
  - A continuación dibujamos el grafo de flujo


Inicio
1
2
true
3
false
false
true
5
false
true
7
6
4
fin

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - Una vez obtenido el grafo de flujo, tenemos que calcular la complejidad ciclomática, es decir, el número de caminos independientes del conjunto básico de un programa.
  - Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una condición, es decir, en términos de diagrama de flujo, está constituido por lo menos por una arista que no haya sido recorrida anteriormente.

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - La fórmula para el cálculo de la complejidad ciclomática “V(G)” es:

    V(G) = a –n + 2

    - Donde:
      - a: Es el número de aristas
      - n: Es el número de nodos
  - Nuestro código tiene una complejidad ciclomática de 4, eso quiere decir que debemos realizar 4 pruebas para asegurarnos de que cada instrucción se ejecute por lo menos una vez.

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - Por último, una vez que conocemos la complejidad ciclomática (4 en nuestro ejemplo), debemos formar los caminos independientes que existen observando el grafo de flujo. Compondremos una tabla empezando por el más corto de los caminos:
  -

CAMINO	ENTRADA	PRUEBA	SALIDA
1,2,5,6,F	a>b=FALSE, b>c=TRUE	a=5 b=7 c=9	c
1,2,3,4,F	a>b=TRUE, a>c=TRUE	a=5 b=3 c=4	a
1,2,5,7,F	a>b=FALSE, b>c=FALSE	a=5 b=7 c=6	b
1,2,3,5,6,F	a>b=TRUE, a>c=FALSE, b>c=TRUE	a=5 b=3 c=7	c

Pruebas de código

- PRUEBA DEL CAMINO BÁSICO
  - Se establecen los siguientes valores de referencia en función de la complejidad ciclomática:

Complejidad ciclomática	Evaluación de riesgo
Entre 1 y 10	Programas o métodos sencillos, sin mucho riesgo.
Entre 11 y 20	Programas o métodos más complejos, riesgo moderado.
Entre 21 y 50	Programas o métodos complejos, alto riesgo.
Mayor que 50	Programas o métodos no testeables, muy alto riesgo.

Pruebas de código

- PARTICIÓN O CLASES DE EQUIVALENCIA
  - Es un método de caja negra.
  - Divide los valores de entrada de un programa en clases de equivalencia.
  - Se definen dos tipos de clases de equivalencia: válidas y no válidas.
    - Ej: Número de empleado numérico, 3 dígitos y >0.
      - Clase de equivalencia válida: número entre 100 y 999
      - Clase de equivalencia no válida: número menor que 100

Pruebas de código

- PARTICIÓN O CLASES DE EQUIVALENCIA
  - Las clases de equivalencia se definen según una serie de directrices:

Condiciones de entrada	Nº de clases de equivalencia VÁLIDAS	Nº de clases de equivalencia NO VÁLIDAS
1. Rango	1 clase válida. Cumple los valores del rango	2 clases no válidas. Un valor por encima del rango Un valor por debajo del rango
2. Valor específico	1 clase válida. Cumple ese valor	2 clases no válidas. Un valor por encima Un valor por debajo
3. Miembro de un conjunto	1 clase válida. Una clase por cada uno de los miembros del conjunto	1 clase no válida. Un valor que no pertenece al conjunto
4. Lógica	1 clase válida. Que cumpla la condición	1 clase no válida. Que no cumpla la condición

Pruebas de código

- ANÁLISIS DE VALORES LÍMITE
  - Se basa en que los errores tienden a producirse con más probabilidad en los límites o extremos de los campos de entrada.
  - Complementa a las clases de equivalencia y los casos de prueba elegidos son los valores justo por encima y por debajo de los márgenes de la clase de equivalencia.
  - También se analizan las condiciones de salida definiendo las clases de equivalencia de salida.

Pruebas de código

- ANÁLISIS DE VALORES LÍMITE
  - También tiene una serie de reglas:
    - Rango: diseñar casos de prueba para los límites del rango y para los valores justo por encima y por debajo del rango. Ej. Entrada de valores comprendidos entre 1 y 10 → diseñar pruebas para el valor 1, 10, 0 y 11.
    - Número de valores (específico): casos de prueba para los valores máximo, mínimo, uno justo encima del máximo y uno justo por debajo del mínimo. Ej. Entrada entre 2 y 10 valores → casos de prueba para 2, 10, 1 y 11 datos de entrada.

Pruebas de código

- ANÁLISIS DE VALORES LÍMITE
  - Reglas (continuación):
    - Aplicar la regla 1 para la condición de salida. Ej. Aplicar un descuento a la salida entre 10% y 50% → pruebas para 9,99%, 10%, 50% y 50,01%.
    - Usar la regla 2 para la condición de salida. Ej. Si la salida es una tabla de 1 a 10 elementos → casos de prueba para 0, 1, 10 y 11 elementos.
      - NOTA: Para las dos reglas anteriores no siempre se podrán generar resultados fuera del rango de salida.
    - Si las estructuras de datos internas tienen límites (un array de 100 elementos), diseñar casos de prueba que ejerciten la estructura de datos en sus límites, primer y último elemento.

Pruebas de código

- PRUEBAS DE BUCLES
- Se deben configurar casos de prueba en los que, siendo N el número máximo permitido de repeticiones, la ejecución:
  - No entre en el bucle
  - Entre una sola vez
  - Entre dos veces
  - Entre M veces, siendo M<N
  - Entre N-1, N y N+1.
- Para bucles anidados:
  - Comenzar por el más interior.
  - Configurar los demás con sus valores límites.
  - Ir aumentando valores y progresar hacia fuera.

Pruebas de código (actividades)

- PRUEBA DEL CAMINO BÁSICO
  - Calcula la complejidad ciclomática del siguiente ejemplo y realiza los casos de prueba necesarios para los caminos indicados en el resultado obtenido:
  -

```
public int metodo(boolean a, boolean b, boolean c){
    int ret = 0;
    if (a && b) {
        ret = 1;
    } else {
        ret = 2;
    }
    return ret;
}
```

Pruebas de código (actividades)

- PRUEBA DEL CAMINO BÁSICO
  - Calcula la complejidad ciclomática del siguiente ejemplo y realiza los casos de prueba necesarios para los caminos indicados en el resultado obtenido:

```
int contar_letra(char cadena[10], char letra){
    int contador, n, longitud;
    n = 0; contador = 0;
    longitud = strlen(cadena);
    if (longitud > 0){
        do{
            if (cadena[contador] == letra){
                n++
            }
            contador++;
            longitud--
        }while (longitud > 0);
    }
    return n;
}
```

Pruebas de código (actividades)

- Determina las clases de equivalencia para los siguientes datos de entrada de una aplicación bancaria:
  - Código de área: número de tres dígitos que no empieza por 0 ni por 1.
  - Identificador de transacción: 6 caracteres.
  - Órdenes posibles: “cheque”, “depósito”, “pago factura”, ”retirada de fondos”.

Datos de entrada	Clases Válidas	Clases no válidas
Código de área		
Identificador de transacción		
Orden		

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración

Herramientas de depuración

- El proceso de depuración comienza con la ejecución de un caso de prueba.
- Se evalúan los resultados de la ejecución y se comprueba que hay una falta de correspondencia entre los resultados esperados y los obtenidos.
- Qué podemos obtener con la depuración:
  - 1. Se encuentra el error, se corrige y elimina.
  - 2. No se encuentra, y hay que diseñar nuevos casos de prueba que ayuden a encontrarlo.

Herramientas de depuración

- Al desarrollar un programa cometeremos dos tipos de errores:
  - De compilación: fáciles de corregir ya que el IDE nos informa de la localización del error y cómo solucionarlo.
  - Lógicos: más difíciles de detectar ya que el programa compila bien pero la ejecución devuelve resultados inesperados y erróneos. Son los bugs.
- El depurador del IDE nos permite analizar el código mientras se ejecuta:
  - Poner puntos de ruptura
  - Suspender la ejecución
  - Ejecutar el código paso a paso
  - Examinar el contenido de las variables

Herramientas de depuración

- Elementos más importantes de la depuración:
  - Resume (F8): continuar ejecución hasta el siguiente breakpoint.
  - Step Into (F5): si el depurador encuentra una llamada a un método o una función, se irá a la primera instrucción dentro de dicho método.
  - Step Over (F6): no entrará al método sino que irá a la siguiente instrucción.
  - Step Return (F7): si nos encontramos dentro de un método, el depurador sale del método actual.
  - Botón “Watch”: permite crear nuevas expresiones basadas en las variables del programa.
  - Botón “Inspect”: crea una nueva expresión para la variable seleccionada y la agrega a la vista de inspección.

6. Diseño y Realización de Pruebas

- Introducción
- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software
- Documentación para las pruebas
- Pruebas de código
- Herramientas de depuración