Tema 3

Diseño y Realización de Pruebas

Entornos de Desarrollo

Objetivos

- Identificar los tipos y estrategias de pruebas.
- Definir casos de prueba.
- Utilizar herramientas de depuración en un entorno de desarrollo.
- Realizar pruebas unitarias utilizando la herramienta JUnit.
- Implementar pruebas automáticas.

Introducción

Durante todo el proceso de desarrollo de software, desde la fase de diseño, en la implementación y una vez desarrollada la aplicación, es necesario realizar un **conjunto de pruebas** (proceso que permite verificar y revelar la calidad de un producto software).

Se utilizan para identificar posibles fallos de implementación, calidad o usabilidad de un programa, que permitan verificar que el software que se está creando, es correcto y cumple con las especificaciones impuesta por el usuario.

Fundamentos de la prueba del software

En la etapa de prueba del software se crean una serie de casos de prueba que intentan "destruir" el software desarrollado.

Objetivos de la prueba

• La prueba es un proceso de ejecución de un programa con la intención de 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.

El objetivo es diseñar casos de prueba que, sistemáticamente, saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo.
Las pruebas no pueden asegurar la ausencia de errores; sólo puede demostrar que existen defectos en el software cuando se detectan.

Fundamentos de la prueba del software

Proceso de prueba

El proceso de prueba tiene dos entradas:

- **Configuración del software:** Incluye la especificación de requisitos del
software, la especificación del diseño y el código fuente.
- **Configuración de prueba:** Incluye un plan y un procedimiento de prueba.

Si el funcionamiento del software parece ser correcto y los errores encontrados son fáciles de corregir, podemos concluir con que:

- La calidad y la fiabilidad del software son aceptables, o que
- Las pruebas son inadecuadas para descubrir errores serios

Fundamentos de la prueba del software

Diseño de casos de prueba

Se trata de diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y de tiempo.

Cualquier producto de ingeniería se puede probar de dos formas:

- **Pruebas de caja negra:** Realizar pruebas de forma que se compruebe que cada función es operativa.

- **Pruebas de caja blanca:** Desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada.

Fundamentos de la prueba del software

Diseño de casos de prueba

En la prueba de la caja negra, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

En la prueba de caja blanca se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos.
A primera vista, la prueba de caja blanca profunda nos llevaría a tener "programas 100 por cien correctos", pero esto supone un **estudio demasiado exhaustivo**, que prolongaría excesivamente los planes de desarrollo del software, por lo que se hará un estudio de los caminos lógicos importantes.

Pruebas de Caja Blanca

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.

Las pruebas de caja blanca **intentan garantizar que:**

• Se ejecutan al menos una vez todas las sentencias.
• Se ejecutan todas las decisiones lógicas en su parte verdadera y en su parte falsa.
• Se ejecuten todos los bucles en sus límites.
• Se utilizan todas las estructuras de datos internas para asegurar su validez.

Pruebas de Caja Negra

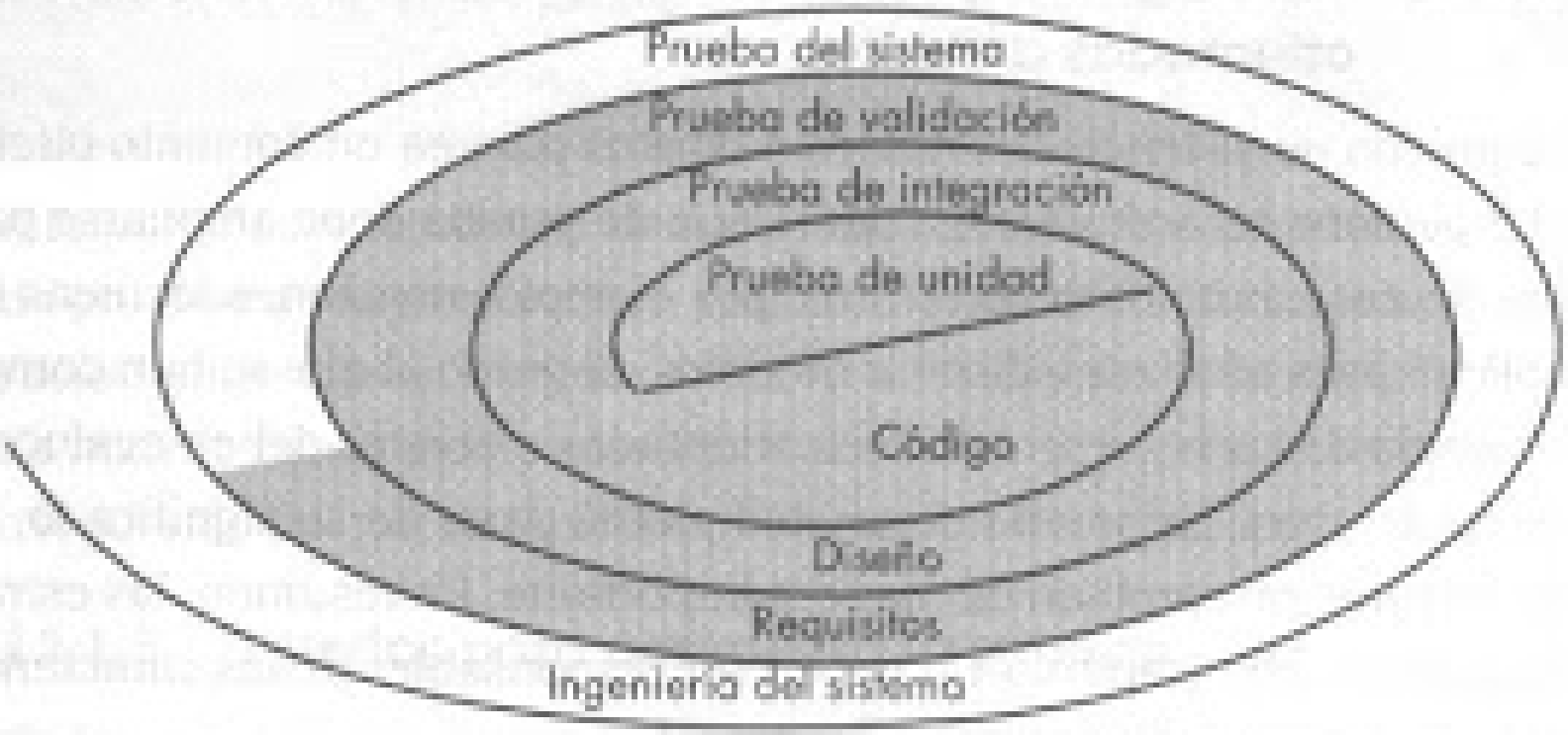
Las pruebas de caja negra se llevan a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra **pretenden encontrar:**

- Funcionalidades incorrectas o ausentes
- Errores de interfaz
- Errores de estructuras de datos o en acceso a las bases de datos.
- Errores de rendimiento.
- Errores de inicialización o finalización.

Estrategias de pruebas del software

La estrategia de pruebas del software se puede ver en el contexto de una espiral.



Estrategias de pruebas del software

La prueba de unidad es una forma de comprobar el correcto funcionamiento de **un módulo de código**. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

Las pruebas de integración son aquellas que se realizan una vez que se han aprobado las pruebas de unidad. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez.
Consiste en **realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos.**

Estrategias de pruebas del software

Las **pruebas de validación** son el proceso de revisión que verifica que la aplicación cumple con las especificaciones y que logra su cometido, es decir, es realmente lo que el usuario quería.
Se trata de evaluar el sistema o parte de este durante o al final del desarrollo para determinar si satisface los requisitos iniciales.

La prueba del sistema verifica que cada elemento encaja de forma completa y se alcanza la funcionalidad total y el rendimiento esperado. Se prueba como un todo y junto a otros elementos del sistema

Documentación de las pruebas

El estándar IEEE 829-1998 describe el conjunto de documentos que pueden producirse durante el proceso de prueba que son los siguientes:

Plan de Pruebas. Sirve para:

- Establecer el alcance, enfoque, recursos y calendario de las actividades de prueba.
- Identificar los elementos poniendo a prueba, las características para ser analizadas, las tareas de pruebas a realizar, el personal responsable de cada tareas y los riesgos asociados con este plan.

Un plan de pruebas deberá tener la siguiente estructura:

- Identificador del plan de prueba
- Elementos de Prueba
- Criterios Pasa / Falla
- Criterios de suspensión y reanudación requisitos
- Entregas de Prueba
- Tareas de Prueba
- Necesidades ambientales
- Responsabilidades

Documentación de las pruebas

Especificaciones de prueba: Están cubiertas por 3 tipos de documentos:

- Se identifican los requisitos, los casos de prueba y los procedimientos que se han llevado a cabo, así como los criterios pasa / no pasa)

- Documenta los valores utilizados como entrada, así como los valores proporcionados de salida.

Informe de pruebas: Identifica que elementos han sido probados, que ha ocurrido durante la prueba, un informe de incidencias detectadas y un resumen.

Pruebas de Código

La prueba de código consiste en la ejecución del programa o parte de él con el objetivo de encontrar errores.

Se parte para su ejecución 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.

Para las pruebas de código se pueden utilizar diferentes técnicas tanto de caja negra como de caja blanca.

Pruebas de Código

Prueba del Camino Básico

- La prueba del camino básico es una técnica de prueba de caja blanca que define un conjunto básico de caminos de ejecución.
- Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

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 analiza los casos en los que hay una falta de correspondencia entre los resultados obtenidos y los esperados.
El proceso de depuración siempre tiene uno de los dos resultados siguientes:

- Se encuentra la causa del error, se corrige y se elimina.
- No se encuentra la causa del error. En este caso, la persona encargada de la depuración debe sospechar la causa, diseñar casos de prueba que ayuden a confirmar sus sospechas y volver a repetir las pruebas.

Al desarrollar un programa cometeremos 2 tipos de errores:

- Los errores de compilación: Son fáciles de corregir ya que si utilizamos un **IDE** nos avisa donde se está produciendo el error y normalmente será de sintaxis.
- Los errores lógicos son más difíciles de detectar ya que el programa se puede compilar con éxito y sin embargo su ejecución puede devolver resultados inesperados y erróneos. A este tipo de errores se le llaman «bugs».

Herramientas de Depuración

Los entornos de desarrollo incorporan una herramienta conocida como depurados (**debugger**) para ayudarnos a detectar errores lógicos.

El depurador es un ejecutable cuya misión es permitir la ejecución controlada de un segundo ejecutable. Permite realizar una serie de operaciones específicas para visualizar el entorno de ejecución en cualquier instante.

Más concretamente, el depurador permite:

- Ejecutar un programa línea a línea
- Detener la ejecución temporalmente en una línea de código concreta
- Detener temporalmente la ejecución bajo determinadas condiciones
- Visualizar el contenido de las variables en un determinado momento de la ejecución
- Cambiar el valor del entorno de ejecución para poder ver el efecto de una corrección en el programa

Herramientas de Depuración

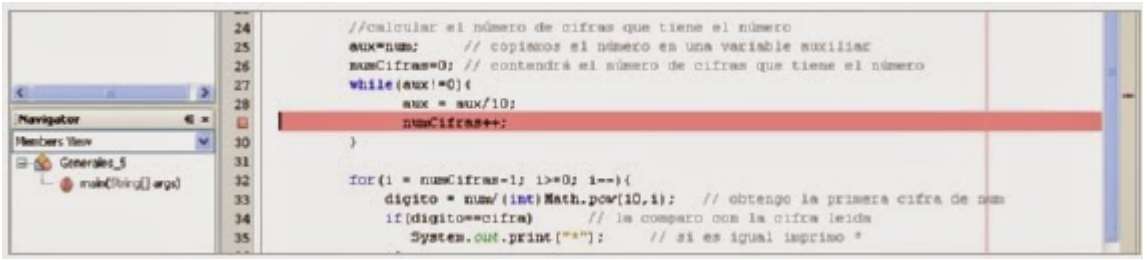
ESTABLECER BREAKPOINTS

Un breakpoint o punto de interrupción es una marca que indica al depurador que debe detenerse cuando la ejecución del programa llegue a ella.

Cuando el programa se detiene en un breakpoint podemos:

- Examinar los valores actuales de las variables.
- Detectar cuando se crea un objeto.
- Continuar la depuración línea a línea del programa.

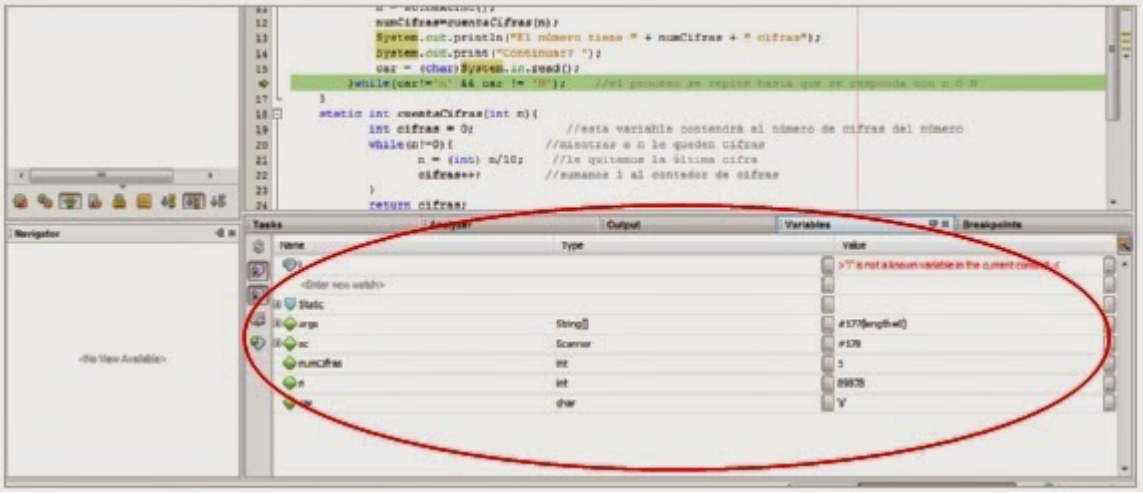
Para fijar un breakpoint se pulsa sobre el número de línea donde se desea colocar. La línea queda resaltada en color rojo con una marca del mismo color en el margen izquierdo.



Herramientas de Depuración

LA VENTANA DE VARIABLES LOCALES.

En esta ventana se muestran las variables, su tipo y su valor actual.



El debugger permite cambiar el valor de una variable local en esta ventana y continuar la ejecución del programa usando el nuevo valor de la variable.

Herramientas de Depuración

Ejercicio:

Poned «breakpoints» en los diferentes métodos de la aplicación realizada y ejecutar el depurador para ver que valores toman las variables en cada momento de la ejecución.

Además utilizad las diferentes opciones estudiadas en las transparencias anteriores.