U.T.3: Diseño y realización de pruebas.

[Fuente: Entornos de Desarrollo, Alicia Ramos, Ed.Garceta]
[Fuente: Análisis y diseño de Aplicaciones Informáticas de Gestión, Mario Piattini, Ed.RA-MA]

Diseño y realización de pruebas

- Técnicas de diseño de casos de prueba
- Estrategias de prueba del software.
- Documentación para las pruebas
- Herramientas de depuración

- La prueba del software consiste en verificar y validar un producto software antes de su puesta en marcha.

Diseño y realización de pruebas

- La ejecución de las pruebas involucra una serie de etapas

Figura 1.16. Flujo de proceso para probar el software.

Técnicas de diseño de casos de prueba

- Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados.
- Existen dos enfoques para diseñarlos:
  - ***Prueba de la caja blanca***: se centran en validar la estructura interna del programa
  - ***Prueba de la caja negra***: se centran en validar los requisitos funcionales del mismo.
- Ambos enfoques se combinan para obtener el mayor número de errores.

Estrategias de prueba del software

- El proceso de prueba se puede ver como una espiral con diferentes etapas:

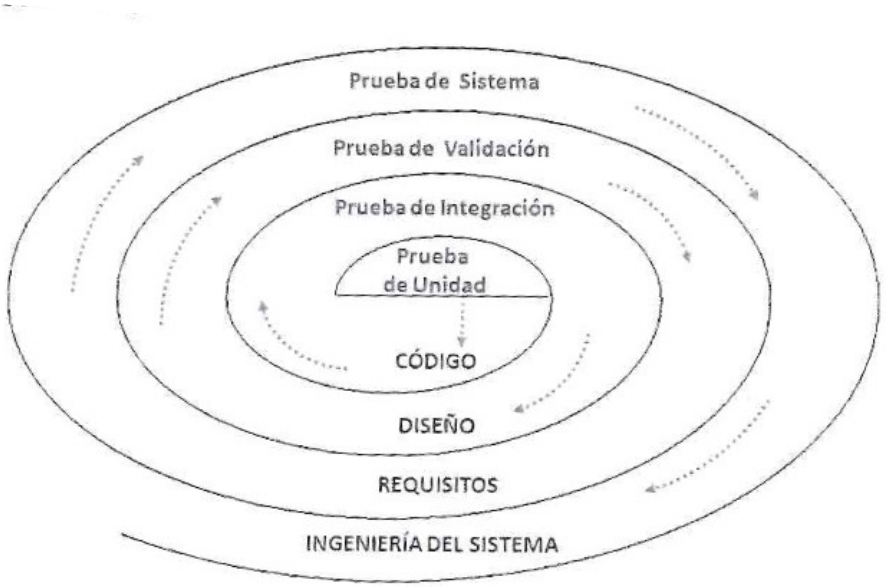


Figura 3.2. Estrategia de prueba.

Estrategias de prueba del software:
Prueba de unidad

- Examina el código fuente a nivel de módulo.
- Utiliza técnicas de caja blanca y caja negra.
- Se realizan pruebas sobre:
  - La interfaz del módulo
  - Las estructuras de datos locales
  - Las condiciones límite
  - Todos los caminos de ejecución independientes
  - Todos los caminos de manejo de errores
- Una herramienta para automatizar esta prueba es JUnit de Eclipse.

Estrategias de prueba del software:
Prueba de integración

- Examina la interacción entre los distintos módulos del programa.
  - ***Integración no incremental***: La prueba sólo se ejecuta después de la integración de todos los módulos.
  - ***Integración incremental***: El programa completo se va construyendo y probando en pequeños segmentos.

Estrategias de prueba del software:
Prueba de validación

- La validación se consigue cuando el programa funciona según los requisitos recogidos en la ERS.
- Se aplican técnicas de prueba de caja negra.
  - ***Prueba Alfa***: La prueba se realiza por el cliente en el lugar de desarrollo con supervisión del desarrollador.
  - ***Prueba Beta***: La prueba la realizan usuarios finales en el lugar de trabajo y sin supervisión del desarrollador.

Estrategias de prueba del software:
Prueba del sistema

- Su misión es validar aspectos del software relativos al rendimiento.
- Se aplican técnicas de prueba:
  - ***Prueba de recuperación***: Se fuerza el fallo del sistema para comprobar su correcta recuperación.
  - ***Prueba de seguridad***: Intenta verificar la protección contra accesos ilegales.
  - ***Prueba de resistencia (Stress)***: Enfrentan al sistema a situaciones de gran demanda de recursos.

Estrategias de prueba del software:
Pruebas de código

- Prueba del camino básico
  - Técnica de caja blanca
  - Obtiene una medida de la complejidad lógica del código.
  - Garantiza que durante la prueba se ejecuta al menos una vez cada sentencia del programa
- Clases de equivalencia
- Análisis de valores límite

Estrategias de prueba del software:
Pruebas de código

- Prueba del camino básico: Ejemplo

```
 static Scanner leer=new Scanner(System.in);

static void paresImpares(){
   int c=0,par=0,impar=0;
   int n;
   while (c!=10){
         c++;
         n=leer.nextInt();
         if (n%2==0)
              par++;
         else
        impar++;
   }
   System.out.println(par,impar);
}
```

Estrategias de prueba del software:

Pruebas de código

- Prueba del camino básico: Ejemplo

Camino	Caso de prueba	Resultado esperado
1	Escoger algún valor de C tal que NO se cumpla la condición C!=10 **C=10**	Visualizar el número de pares y de impares
2	Escoger algún valor de C tal que SÍ se cumpla la condición C!=10 Escoger algún valor de N tal que NO se cumpla la condición N%2==0 **C=1 , N=5**	Contar números impares
3	Escoger algún valor de C tal que SÍ se cumpla la condición C!=10 Escoger algún valor de N tal que SÍ se cumpla la condición N%2==0 **C=2 , N=4**	Contar números pares

Estrategias de prueba del software:
Pruebas de código

- Prueba del camino básico: Ejemplo 2

```
static void visualizarMedia(float x, float  y){
   float resultado=0;
   if (x < 0 ||  y < 0)
       System.out.println("X e Y deben ser positivos");
   else{
       resultado = (x+y) / 2;
       System.out.println("La media es: "+resultado);
   }
}
```

Estrategias de prueba del software:

Pruebas de código

Prueba del camino básico: Ejemplo

Camino	Caso de prueba	Resultado esperado
1	Escoger algún X e Y tal que NO se cumpla la condición X<0 || Y<0 **X=4, Y=5** **visualizarMedia(4,5)**	Visualiza: La media es : 4.5
2	Escoger algún X tal que SÍ se cumpla la condición X<0 (Y puede ser cualquier valor) **X=-4, Y=5** **visualizarMedia(-4,5)**	Visualiza: X e Y deben ser positivos
3	Escoger algún X tal que NO se cumpla la condición X<0 y escoger algún Y que SÍ cumpla la condición Y<0 **X=4, Y=-5** **visualizarMedia(4,-5)**	Visualiza: X e Y deben ser positivos

Estrategias de prueba del software:
Pruebas de código

- Clases de equivalencia
  - Es un método de caja negra
  - Divide los valores o datos de entrada en clases de equivalencia o grupos:
    - Clases válidas
    - Clases no válidas
  - Los casos de prueba cubren tantas clases de equivalencia válidas como sea posible y sólo una clase no válida.
  - Se añaden casos de prueba hasta que todas las clases de equivalencia hayan sido cubiertas.

Estrategias de prueba del software:

Pruebas de código

- Resumen de las clases de equivalencia a definir

Condiciones de Entrada	Nº de clases de equivalencia válidas	Nº de clases de equivalencia NO válidas
1. Rango	1 CLASE VÁLIDA Contempla 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 Contempla dicho 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 miembro del conjunto	1 CLASE NO VÁLIDA Un valor que no pertenece al conjunto
4. Lógica	1 CLASE VÁLIDA Una clase que cumple la condición	1 CLASE NO VÁLIDA Una clase que no cumpla la condición

Estrategias de prueba del software:
Pruebas de código

- Ejemplo:

Se va realizar una entrada de datos de empleados por teclado. Se definen 3 campos de entrada, uno de ellos es una lista para elegir el oficio. La aplicación acepta los datos de esta manera:

- *Empleado*: número de tres dígitos que no empiece por 0.
- Departamento: en blanco o número de dos dígitos.
- Oficio: Analista, Diseñador, Programador o Elige oficio.

Si la entrada es correcta el programa asigna un salario (que se muestra en pantalla) a cada empleado según estas normas:

- S1 si el Oficio es Analista se asigna 2500
- S2 si el Oficio es Diseñador se asigna 1500
- S3 si el Oficio es Programador se asigna 2000

Si la entrada no es correcta el programa muestra un mensaje indicando la entrada incorrecta:

- ER1 si el Empleado no es correcto
- ER2 si el Departamento no es correcto
- ER3 si no ha elegido Oficio

Estrategias de prueba del software:

Pruebas de código

- Ejemplo:

Condición de Entrada	Clases de Equivalencia	Clases Válidas	COD	Clases no válidas	COD
Empleado	Rango	100>=Empleado<=999	V1	Empleado < 100 Empleado > 999	NV1 NV2
Departamento	Lógica (puede estar o no)	En blanco	V2	No es un número	NV3
	Valor	Cualquier número de dos dígitos	V3	Número de más de 2 dígitos Número de menos de 2 dígitos	NV4 NV5
Oficio	Miembro de un conjunto	Oficio="Programador"	V4	Oficio="Elige oficio"	NV6
		Oficio="Analista"	V5		
		Oficio="Diseñador"	V6		

Estrategias de prueba del software:
Pruebas de código

- Ejemplo:

Caso de prueba	Clases de Equivalencia	CONDICIONES DE ENTRADA			Resultado Esperado
		Empleado	Departamento	Oficio	
CP1	V1,V3,V4	200	20	Programador	S3
CP2	V1,V2,V5	250		Analista	S1
CP3	V1,V3,V6	450	30	Diseñador	S2
CP4	V1,V2,V4	220		Programador	S3
CP5	NV1,V3,V6	90	35	Diseñador	ER1
CP6	V1,NV3,V5	100	AD	Analista	ER2
CP7	V1,V2,NV8	300		Elige oficio	ER3
CP8	V1,NV4,V6	345	123	Diseñador	ER2
…			…		

Estrategias de prueba del software:
Pruebas de código

- Análisis de valores límite
  - Los errores suelen producirse con mayor probabilidad en los extremos o límites de los campos de entrada
  - Técnica complementaria a las clases de equivalencia
  - Los casos de prueba ejercitan los valores justo por encima y por debajo de los límites de la clase de equivalencia.

Estrategias de prueba del software:
Pruebas de código

- Análisis de valores límite: ejemplo

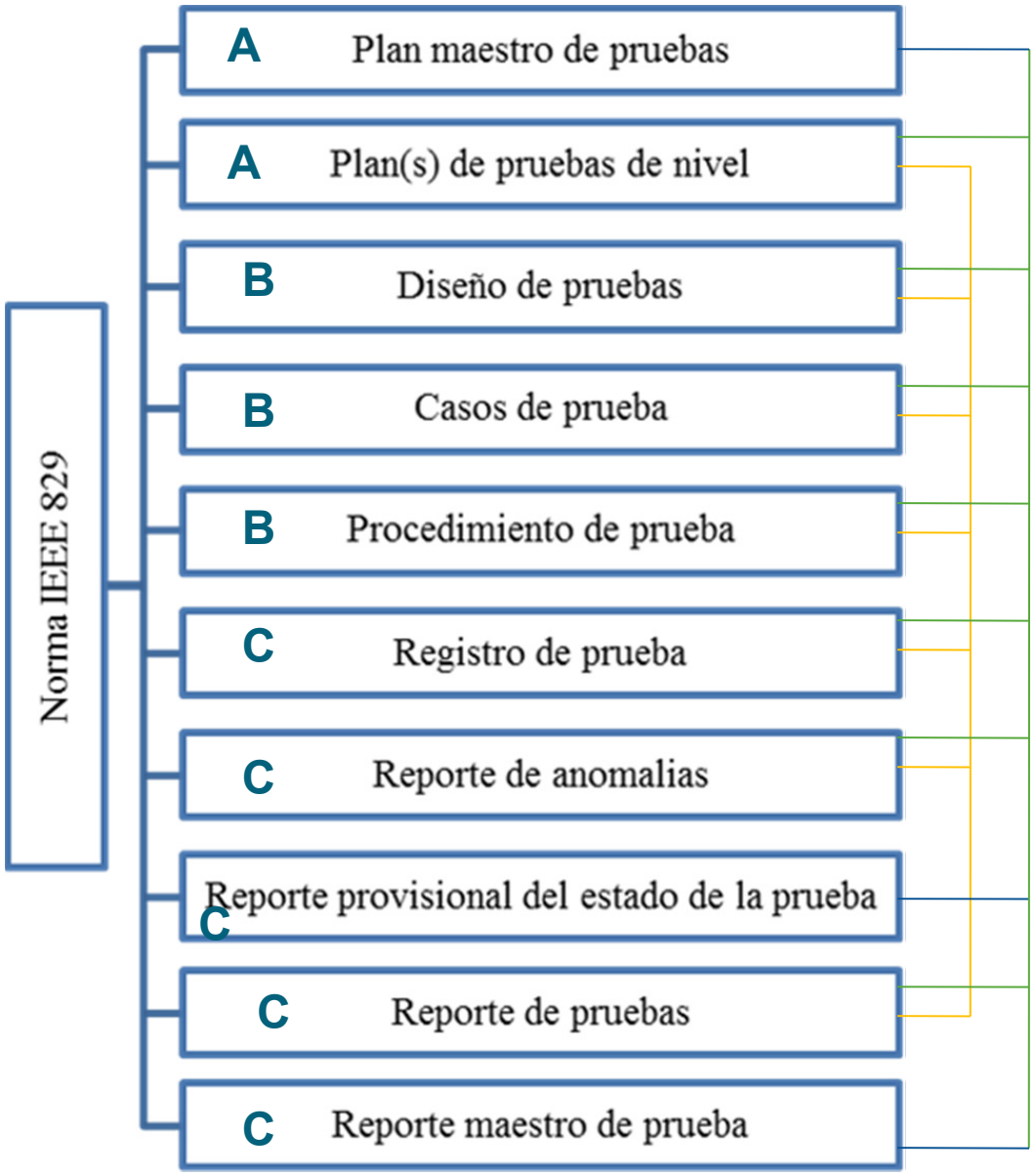
	Condiciones de Entrada y Salida	Casos de prueba
Código	Entero 1 a 100	Valores: 0, 1, 100, 101
Puesto	Alfanumérico de hasta 4 caracteres	Longitud de caracteres: 0, 1, 4, 5
Antigüedad	De 0 a 25 años (Real)	Valores: 0, 25, -0.1, 25.1
Horas semanales	De 0 a 60	Valores: 0, 60, -1, 61
Fichero de entrada	Tiene de 1 a 100 registros	Para leer 0, 1, 100 y 101 registros
Fichero de salida	Podrá tener de 0 a 10 registros	Para generar 0, 10 y 11 registros
Array interno	De 20 cadenas de caracteres	Para el primer y último elemento

Documentación para las pruebas

- El estándar IEEE 829-1998 establece diez documentos básicos que deben ser generados durante el proceso de pruebas.
- El nivel de integridad del software determina cuales de ellos son obligatorios en cada caso:
  1. Insignificante
  2. Marginal
  3. Crítico
  4. Catastrófico

Documentación para las pruebas

- Insignificante
- Catastrófico



Documentación para las pruebas

- Los documentos generados en la fase de pruebas según el estándar 829 se pueden agrupar en tres grupos:

A. Plan de pruebas
   - Describe el alcance, enfoque, recursos y calendario
   - Identifica las características a ser probadas y se asocia el personal implicado en ellas.

B. Especificaciones de prueba.
   - Describe detalladamente los casos de prueba y los pasos a seguir para realizarlas

C. Informes de pruebas.
   - Recoge el registro de los resultados de la prueba y de las incidencias producidas, así como un resumen del proceso

Herramientas de prueba: JUnit

- JUnit es una herramienta integrada en Eclipse para realizar pruebas unitarias automatizadas.
- Es posible crear clases que sirvan para lanzar conjuntos de tests (TestSuite) y de esta manera automatizar más aún el proceso de prueba.
- En la clase de prueba se crea automáticamente un método de prueba por cada método de la clase a probar.
- Más información sobre la herramienta:

Métodos de JUnit

Método assertxxx() de JUnit	***Qué comprueba***
assertTrue(expresión)	que la *expresión* evalúe a true
assertFalse(expresión)	que la *expresión* evalúe a false
assertEquals(esperado,real)	que esperado sea igual a real
assertNull(objeto)	que objeto sea null
assertNotNull(objeto)	que objeto no sea null
assertSame(objeto_esperado,objeto_real)	que objeto_esperado y objeto_real sean el mismo objeto
assertNotSame(objeto_esperado,objeto_real)	que objeto_esperado no sea el mismo objeto que objeto_real
fail()	hace que el test termine con fallo

- Las anotaciones disponibles en JUnit 4 son:

Annotation	***Description***
@Test public void method()	La anotación @Test identifica que un método es un método de test.
@Before public void method()	Este método es ejecutado antes de cada test. Este método puede preparar el entorno de test.
@After public void method()	Este método es ejecutado después de cada test. Este método puede limpiar el entorno de test. También puede recuperar memoria, limpiando estructuras de memoria costosas.
@BeforeClass public static void method()	Este método es ejecutado una vez, antes del comienzo de todos los tests. Se puede usar para realizar actividades intensivas en cuanto al tiempo, por ejemplo para conectarse a una base de datos. Los métodos así anotados necesitan ser definidos como static en JUnit.

- Las anotaciones disponibles en JUnit 4 son:

Annotation	***Description***
@AfterClass public static void method()	Este método es ejecutado una vez, después de que todos los tests hayan sido finalizados. Esto puede usarse para realizar tareas de limpieza, por ejemplo, desconectarse de una base de datos. Los métodos así anotados necesitan ser definidos como static en JUnit.
@Ignore	Ignora el método de test. Esto es útil cuando el código subyacente ha sido cambiado y el caso de testeo no ha sido adaptado. O si el tiempo de ejecución de este test es demasiado largo como para ser incluido.
@Test (expected = Exception.class)	Falla si el método no lanza la excepción esperada.
@Test(timeout=100)	Falla si el método toma más de 100 milisegundos.

Herramientas de depuración

-