Ingeniería del Software II

Tema 1: Verificación del Software

A. Goñi, J. Iturrioz

Índice

- Introducción. Definición, principios y objetivos.
- Tipos de testing.
- Técnicas de testing.
  - Caja Blanca.
  - Caja Negra.
  - Valores Límite.

Qué es el software testing

- Según el IEEE [1] “el software testing se basa en un conjunto de actividades dirigidas a facilitar la identificación y/o evaluación de propiedades en el software”.

- Una de las propiedades más interesantes consiste en **analizar un producto** software para detectar las diferencias entre el comportamiento real con el esperado.

- Es decir, **comparar “lo que es”** con **“lo que debería ser”**

[1] ISO/IEC/IEEE 29119-1: Concepts & Definitions. https://ieeexplore.ieee.org/document/6588537/

Verificación vs. Validación

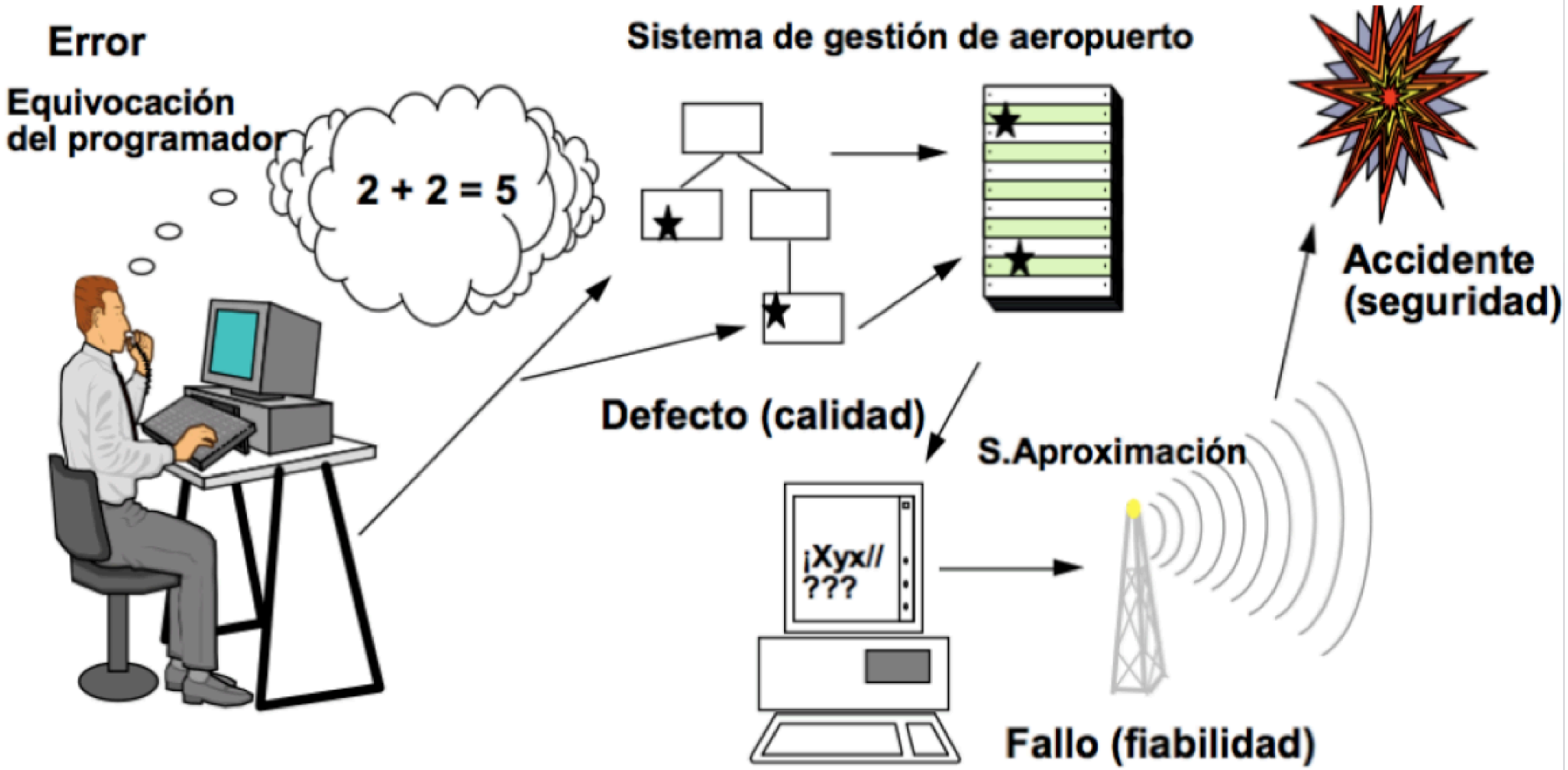
Verificación: El proceso de evaluación de un sistema (o de uno de sus componentes) para determinar si los productos de una fase dada, satisfacen las condiciones impuestas al comienzo de dicha fase **¿Estamos construyendo correctamente el producto?**

Validación: El proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario. **¿Estamos construyendo el producto correcto?**

Error, defecto y fallo

- **Error**: equivocación realizada por una persona (en la actividad del desarrollo software).
- **Defecto**: resultado de introducir un error en un artefacto software.
- **Fallo**: representación del defecto en el comportamiento del sistema.

Relación entre error, defecto y fallo



Error (del programador) —Se plasma en→ **Defecto** (en el software) —Da lugar a→ **Fallo** (el sistema no se comporta como debería) —Que provoca→ **Efectos negativos** (dependiendo de la criticidad del sistema)

Principios del testing

- El objetivo del **testing** consiste en la detección de defectos en el software. Una vez identificados, el proceso de **debugging**, está relacionado con la búsqueda de dónde ocurren los defectos.
- Representa la última revisión de la especificación de los requisitos, el diseño y el código.
- Es uno de los métodos utilizados para asegurar la calidad del software.
- Muchas organizaciones consumen entre el 40-50% del tiempo de desarrollo en el testing.
- Aunque se utilicen los mejores métodos de revisión, el testing es siempre necesario.

Principios del testing

- Se debe hacer un seguimiento hasta ver si se cumplen los requisitos del cliente.
- El principio de Pareto es aplicable a las pruebas del software.
  - El 80% de los errores está en el 20% de los módulos.
  - Hay que identificar esos módulos y probarlos muy bien.
- Empezar por lo pequeño y progresar hacia lo grande.
- No son posibles las pruebas exhaustivas.
- Son más eficientes las pruebas dirigidas por un equipo independiente al que ha creado el código.

Objetivos de una prueba

- La prueba es un proceso de ejecución con la intención de descubrir defectos.
- Un buen caso de prueba es aquel que tiene una probabilidad muy alta de descubrir un nuevo defecto.
  - Un caso de prueba no debe ser redundante.
  - Debe ser el mejor de un conjunto de pruebas de propósito similar.
  - No debe ser ni muy sencillo ni excesivamente complejo: es mejor realizar cada prueba de forma separada si se quiere probar diferentes casos.
- Una prueba tiene éxito si descubre un nuevo defecto.

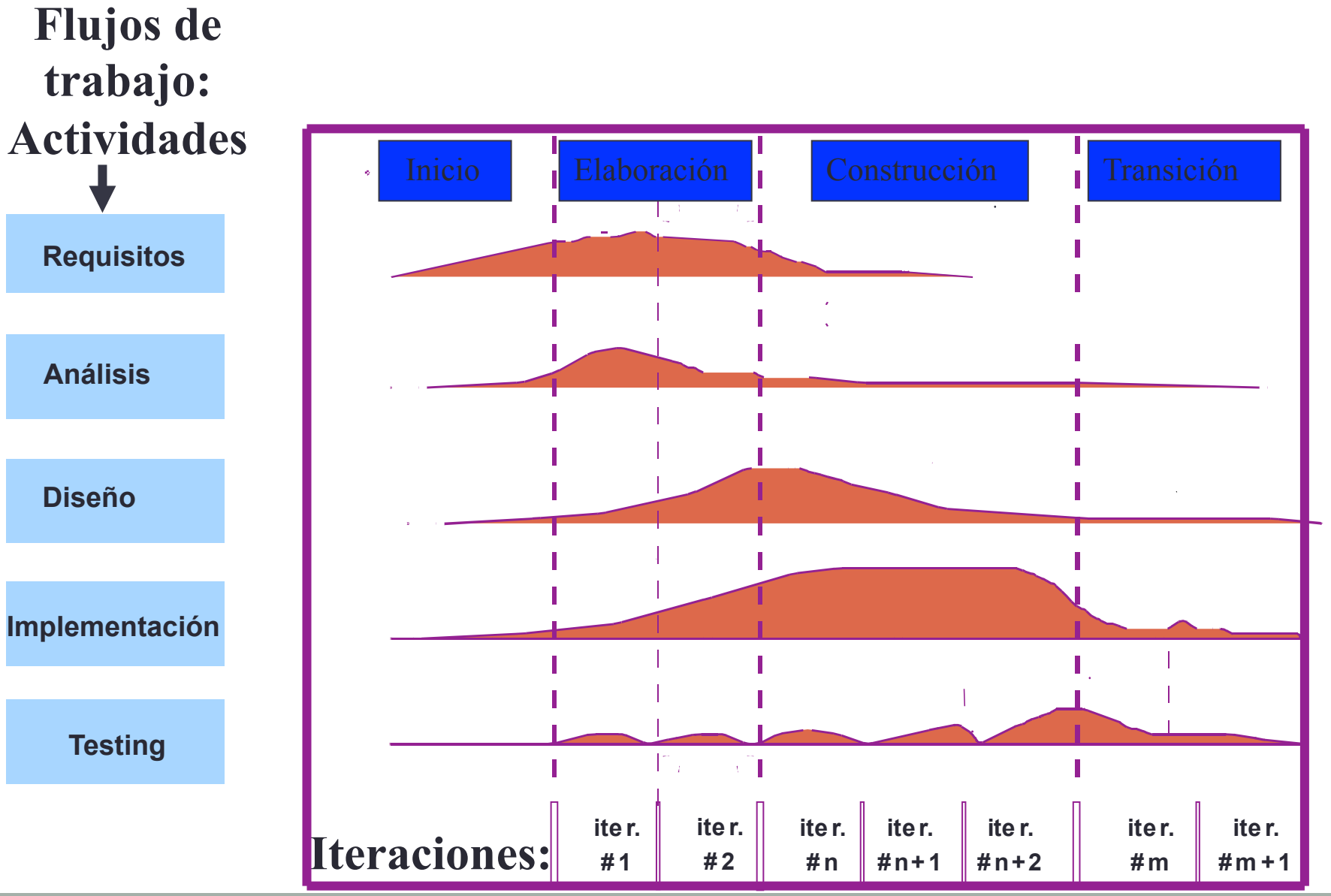
Axioma del testing

"La prueba de un programa sólo puede mostrar la presencia de defectos, no su ausencia"

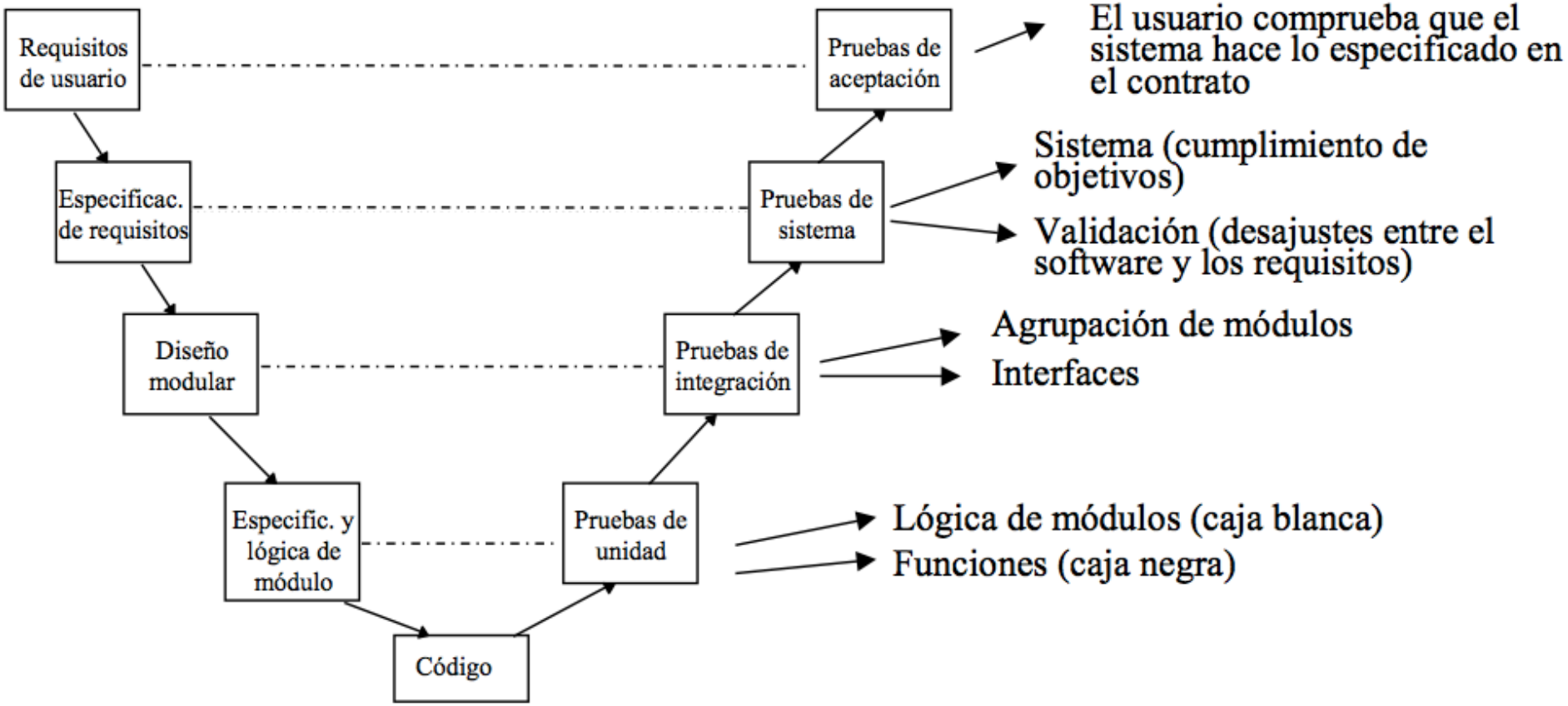
Edsger Wybe Dijkstra

http://www.cs.utexas.edu/users/EWD/

El testing en el ciclo de vida software



¿Qué se puede probar?



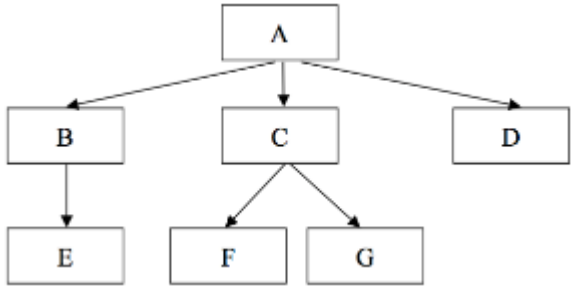
Existe una correspondencia entre cada nivel de prueba y el trabajo realizado en cada etapa del desarrollo

Pruebas de unidad

Se trata de las pruebas formales que permiten declarar que un módulo está listo y terminado (no las informales que se realizan mientras se desarrollan los módulos).

- Las pruebas de unidad pueden abarcar desde un módulo hasta un grupo de módulos (incluso un programa completo).
- Estas pruebas suelen realizarlas el propio personal de desarrollo, pero evitando que sea el propio programador del módulo.

Pruebas de integración



- Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo.
- Tipos fundamentales de integración:
  - **Integración incremental**. Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados
    - ascendente. Se comienza por los módulos hoja.
    - descendente. Se comienza por el módulo raíz.
  - **Integración no incremental**. Se prueba cada módulo por separado y luego se integran todos de una vez y se prueba el programa completo.
- Habitualmente las pruebas de unidad y de integración se solapan y mezclan en el tiempo.

Pruebas de sistema

Es el proceso de prueba de un sistema integrado de hardware y software para comprobar lo siguiente:

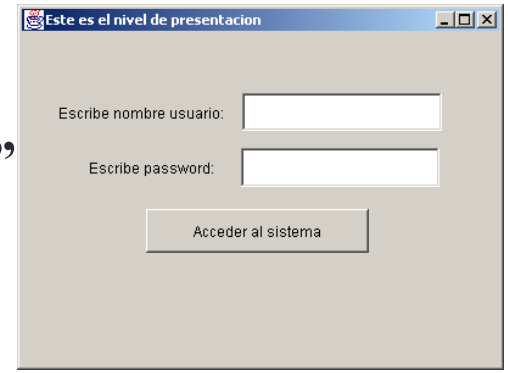
- Cumplimiento de todos los requisitos funcionales, considerando el producto software final al completo en un entorno de sistema.
- El funcionamiento y rendimiento en las interfaces hardware, software, de usuario y de operador.
- Adecuación de la documentación de usuario.
- Ejecución y rendimiento en condiciones límite y de sobrecarga.

Pruebas de aceptación

- Es la prueba planificada y organizada formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente.
- Sus características principales son las siguientes:
  - Participación del usuario.
  - Enfocadas hacia la prueba de los requisitos de usuario especificados.
- Está considerada como la fase final del proceso para crear una confianza en que el producto es el apropiado para su uso en explotación.

Definición: Caso de prueba

- **Es un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados.**
- **Tiene un objetivo concreto (probar algo).**



<u>Ejemplo: CASO de PRUEBA CP1</u> para el CASO de USO "Entrada Sistema"

ENTRADA: usuario "pepe" password "1234".

CONDICIONES DE EJECUCIÓN: no existe en la tabla CUENTA(usuario,pass,intentos) la tupla <"pepe", "123",x> pero sí una tupla <"pepe","321",x>.

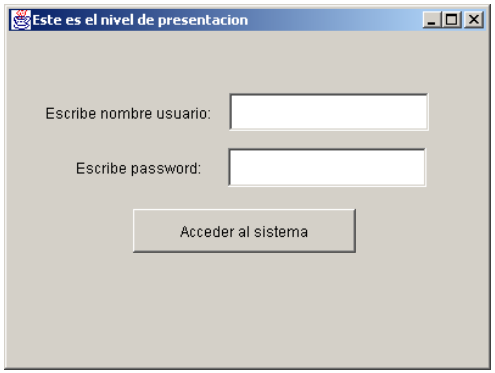
RESULTADO ESPERADO: no deja entrar y cambia la tupla a <"pepe","321",x+1>.

OBJETIVO DEL CASO DE PRUEBA: comprobar que no deja entrar a un usuario existente con un password equivocado, y aumentar el número de intentos.

Definición: Procedimiento de prueba

- **Pasos que hay que llevar a cabo para probar uno (o varios) casos de prueba: ¿Cómo probar el caso de prueba y verificar si ha tenido éxito?**

Ejemplo: Procedimiento de prueba para el CP1



- Ejecutar la clase Presentacion.

- Comprobar que en la BD "passwords.mdb" existe la tupla <"pepe","321",x>.

- Escribir "pepe"en la interfaz gráfica (en el campo de texto etiquetado "Escribe nombre usuario".

- Escribir "123"en la interfaz gráfica (en el campo de texto "Escribe password").

- Pulsar botón "Acceder al sistema".

- Comprobar que no deja entrar al sistema y que en la BD la tupla ha cambiado a <"pepe","321",x+1>.

Definición: Componente de prueba

- **Programa que automatiza la ejecución de uno (o varios) casos de prueba**
- **Una vez escrito, se puede probar muchas veces (cada vez que haya un cambio en el código de una clase que pueda afectarle)**

```
public class ComponentePruebaEntradaSistema {
public void testLoginPassword() {
 InterfaceLogicaNegocio ln; InterfaceOperacionesParaPruebas lp;
 lp.añadirUsuario("pepe","321",3); // Crea usuario con pass y numInt.
 boolean b = ln.hacerLogin("pepe","123");
 int j = lp.comprobarUsuario("pepe","321"); // Devuelve n° intentos
 assertEquals( b==false && j==4); //sino error
} // Fin caso prueba CP1
```

NOTA: se necesitarán otros métodos como **`comprobarUsuario,añadirUsuario`** que pueden pertenecer a la lógica del negocio o no (en este caso se considera que no).

¿Cómo escribir componentes de prueba?

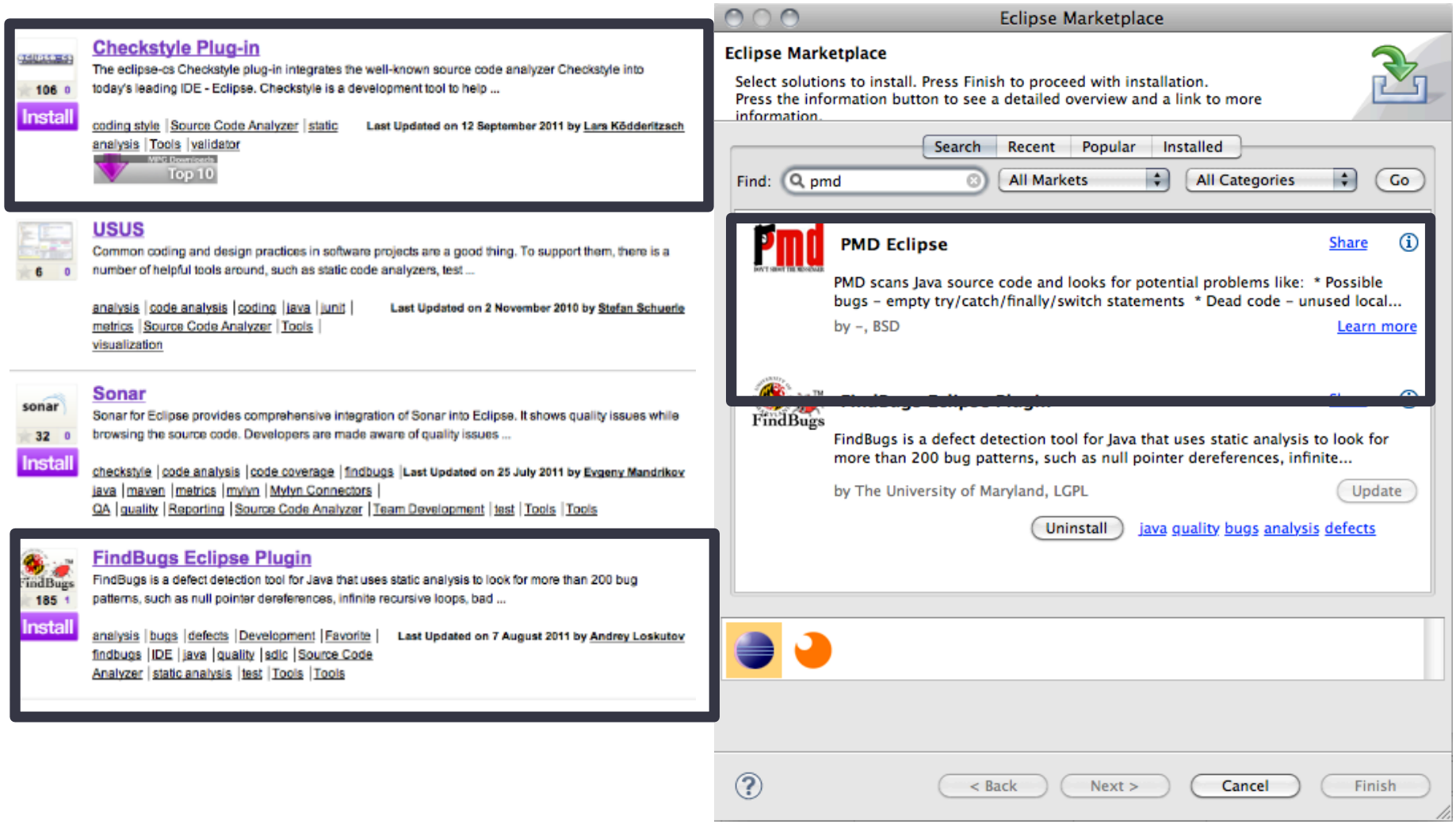
- Se puede escribir “ad hoc”.
  - Por cada caso de prueba, se escribe el código correspondiente. en el componente (cambiaría el código).
- Se pueden usar entornos de trabajo disponibles para pruebas.
  - Ejemplo: JUnit para Java (ver laboratorio 1)

Técnicas de testing

- **Estáticas**
  - Buscan fallos sobre el sistema en reposo.
  - Se puede aplicar sobre artefactos de requerimientos, análisis, diseño y código.
  - Técnicas: Revisiones, Inspecciones, Walkthrough y Auditorías de calidad.
- **Dinámicas**
  - Se ejecuta y observa el comportamiento de un producto.

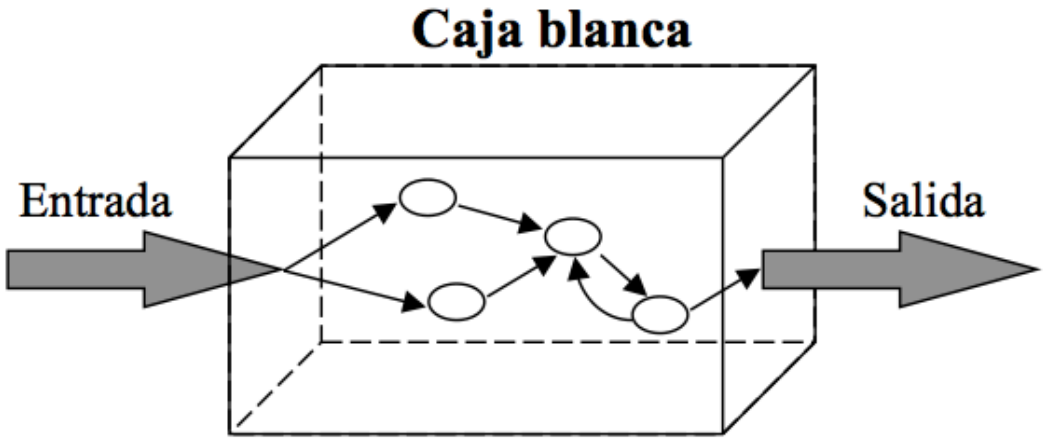
    Estímulo → Proceso → Respuesta
  - Tipos
    - **Caja Negra**: Se centran en los requisitos funcionales.
    - **Caja Blanca**: Se centran en el código.

Herramientas de testing estático

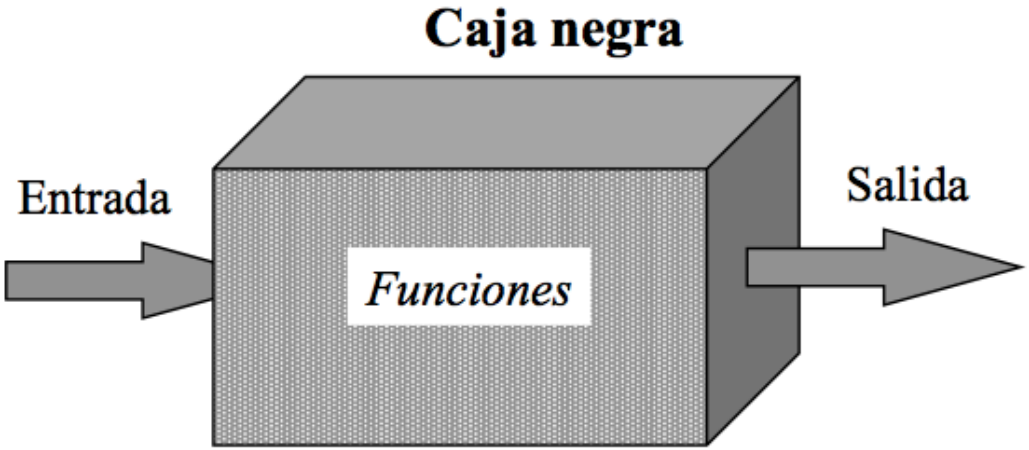


Técnicas de testing dinámico

Estructural



Funcional

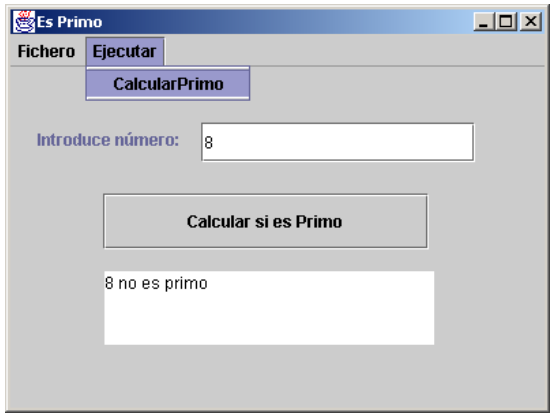


Pruebas Caja Blanca

"Mirando" el código interno

- Intentan garantizar que todos los caminos de ejecución del programa **quedan cubiertos.**
- Usa la estructura de control para obtener los casos de prueba:
  - De condición: Diseñar casos de prueba para que todas las condiciones del programa se evalúen a cierto/falso.
  - De bucles: Diseñar casos de prueba para que se intente ejecutar un bucle 0,1,…,n-1,n y n+1 veces (siendo n el número máximo).

Ejemplo: EsPrimo




```
public class Esprimo {

  public static boolean esPrimo (String args[])
      throws  ErrorFaltaParametro,
              ErrorSololParametro,
              ErrorNoNumeroPositivo {
      if (args.length == 0) throw new ErrorFaltaParametro();
      else if (args.length > 1) throw new ErrorSololParametro();
      else {
        try {
          float numF = Float.parseFloat(args[0]);
          int num = (int)numF;
          if (num<=0) throw new ErrorNoNumeroPositivo();
          else {
            for (int i=2;i<num;i++)
              if (num%i==0) {return false;}
            return true;
          }
        }
        catch (NumberFormatException e) { throw new ErrorNoNumeroPositivo();  }
      }
  }
```

El método Esprimo.esPrimo puede ser llamado con un array de Strings

Casos de prueba de Caja Blanca para EsPrimo

A	B	C	D
Condición que se evalúa	**Condición de entrada**	**Dato de entrada**	**Salida esperada**
IF-1 (T)	args.length==0	args={}	Excepción ErrorFaltaparametro
IF-1(F) y IF-2(T)	args.length>1	args={4,12}	Excepción ErrorSolo1Parametro
IF-1 (F) y IF-2(F) y IF-3(T)	num<0	num=-4	Excepción ErrorNoNumeroPositivo
IF-1(F) y IF-2(F) y IF-3(F) y bucle 0 veces	num=2	num=2	primo
IF-1(F) y IF-2(F) y IF-3(F) y bucle 1 vez (IF4(F))	num=3	num=3	primo
IF-1(F) y IF-2(F) y IF-3(F) y bucle 2 veces (IF4(T))	num=4	num=4	no primo
IF-1(F) y IF-2(F) y IF-3(F) y bucleN veces	num>4	num=34	primo
Excepcion ErrorNoNumeroPositivo	tipo args[0] no número	args[0]="patata"	Excepción ErrorNoNumeroPositivo

OBJETIVO A PROBAR

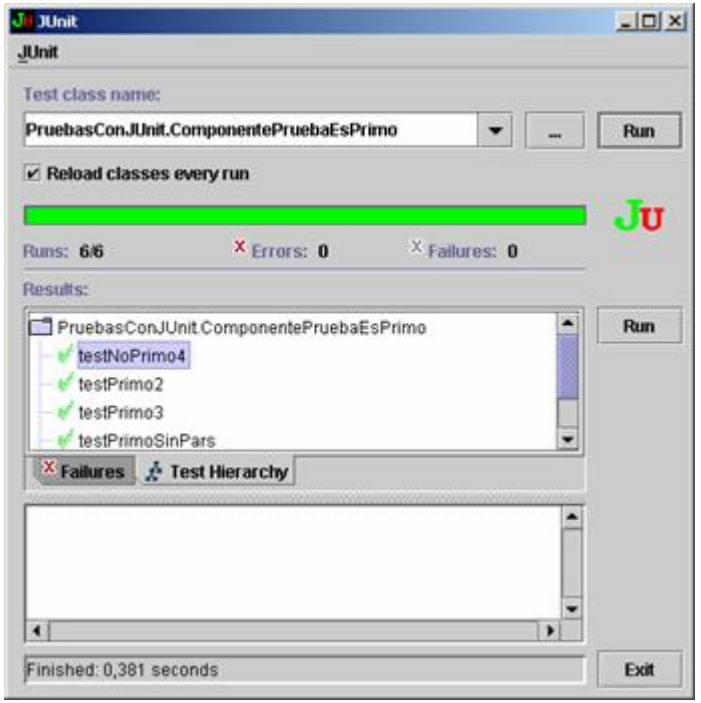
- **Probar todas las condiciones**
- **Probar bucles**

Componente de prueba para EsPrimo

```
public class ComponentePruebaEsPrimo {
 public static void main(String[] args) {
     String[] s1 = new String[0];
     try {boolean b1 = Esprimo.esPrimo(s1);
          System.out.println("CP1 incorrecto");}
     catch (ErrorFaltaParametro e)
         {System.out.println("CP1 correcto");}
     catch (Exception e) {System.out.println("CP1 incorrecto");}
     String[] s2 = new String[2]; s2[0]="xx"; s2[1]="yy";
     try {boolean b1 = Esprimo.esPrimo(s2);
          System.out.println("CP2 incorrecto");}
     catch (ErrorSolo1Parametro e)
          {System.out.println("CP2 correcto");}
     catch (Exception e) {System.out.println("CP2 incorrecto");}
     ...}
```

Componente de prueba para EsPrimo (JUnit)

java junit.swingui.TestRunner PruebasConJUnit.ComponentePruebaEsPrimo




```
package PruebasConJUnit;

public class ComponentePruebaEsPrimo {
 // Un método por cada caso de prueba
 public void testPrimoSinPars() {
        num = new String[0];
        try {result= Esprimo.esPrimo(num);
             assertTrue(false);}
        catch (Exception e)
           {assertTrue(e instanceof ErrorFaltaParametro);}
    }
// RESTO DE CASOS DE PRUEBA...
}
```

Pruebas de Caja Negra

- También denominadas **pruebas de comportamiento**.
- Consideran la **función específica** para la cuál fue creado el producto (qué es lo que hace).
- Las pruebas se llevan a cabo **sobre la interfaz** del sistema.
- No es una alternativa a la prueba de caja blanca.
  - Complementan a las pruebas de caja blanca.

Mejor diseñar los casos de prueba usando los dos tipos de técnicas

El desafío

- Seleccionar los casos de prueba que sean **eficaces.**
  - Que detecten defectos sin requerir un número excesivo de pruebas.

- Buscar las combinaciones de entradas y estados del sistema que tengan la más alta probabilidad de encontrar defectos dentro de un conjunto inmenso que no podemos probar exhaustivamente.

Técnicas de prueba de Caja negra

- Partición equivalente.
- Análisis de valores límite.

Caja Negra: Prueba de partición equivalente

- Se basa en la **división de los parámetros de entrada** en un conjunto de clases de datos denominadas **clases de equivalencia.**

- Se parte de la premisa de que cualquier elemento de una clase de equivalencia es representativo del resto del conjunto.

- A partir de las clases de equivalencia se obtienen las clases de equivalencia válidas e inválidas.

Definición: clase de equivalencia

- **Clase de equivalencia**: conjunto de datos de entrada donde el comportamiento del software es igual para todos los datos del conjunto. Las clases pueden ser:
  - Válida: genera un valor esperado.
  - Inválida: genera un valor inesperado.
- Las clases de equivalencia se obtienen de las **condiciones de entrada** descritas en las especificaciones del software a desarrollar.

Ejemplo: Condiciones de entrada

Especificación entrada	Ejemplo
Un valor específico	“..Introducir **cinco** valores..”
Un rango de valores	...Valores **entre 0 y 10**…
Un valor enumerado	Introducir un día de la semana
Un tipo de valor	Introducir un número **entero**
Condición lógica	Introducir un número **par**

Identificación de las clases de equivalencia

- Por cada condición de entrada se identifican clases de equivalencia válidas y no válidas.
- Este proceso es heurístico.
- Sin embargo existen un conjunto de criterios que ayudan a su identificación.

Criterios para la identificación de las clases de equivalencia

Condición entrada	Ejemplo	Clases de equivalencia válidas	Clases de equivalencia no válidas
Un valor específico	"..Introducir **cinco** valores.."	1 clase que contemple dicho valor	2 clases que representen un valor por encima y otro por debajo
Un rango de valores	...Valores **entre 0 y 10**…	1 Clase que contemple los valores del rango	2 clases fuera del rango, una por encima y otra por debajo
Un valor enumerado	Introducir un día de la semana	1 Clase que contemple un valor del enumerado	1 clase que contemple un valor fuera del enumerado
	Introducir un número **entero**	1 clase que contemple los elemento de ese tipo	N clases que representen tipos distintos(Nulo, alfanuméricos, fraccionarios)
Condición lógica	Introducir un número **par**	**1 clase que cumpla la condición**	**1 clase que no cumpla**
Elementos de un conjunto tratados diferente por el programa	**Las personas menores de 25 años tendrán una bonificación del 10%**	**1 clase que cumpla la condición**	

Tabla resultado de clase de equivalencia

Las clase de equivalencia se definen en la siguiente tabla de cara a identificar los casos de prueba.

Condición entrada	Clases de equivalencia válida	Clases de equivalencia no válida

Derivación de los casos de prueba a partir de las clases de equivalencia

- Generar el menor número de casos de prueba que solapen todas las clases de equivalencia válidas.
- Generar un caso de prueba por cada una de las clases de equivalencia inválida.
- A partir de estos caso se deberá construir la siguiente tabla.

Valor(es) de entrada	Clases de equivalencia cubiertas	Resultado

Ejemplo 1

Construcción de una batería de pruebas para detectar posibles errores en la construcción de los identificadores de un hipotético lenguaje de programación. Las reglas que determinan sus construcción sintáctica son:

- No debe tener más de 15 ni menos de 5 caracteres.
- El juego de caracteres utilizables es:
  - Letras (Mayúsculas y minúsculas)
  - Dígitos (0,9)
  - Guión (-)
- El guión no puede estar ni al principio ni al final, pero puede haber varios consecutivos.
- Debe contener al menos un carácter alfabético.
- No puede ser una de las palabras reservadas del lenguaje.

Tabla resultado de clases de equivalencia

Condición entrada	Clases de equivalencia válidas	Clases de equivalencia no válidas
Entre 5 y 15 caracteres	5≤nº caracteres Ident. ≤15 (1)	nº caracteres Id<5 (2) 15< nº caracteres (3)
El identificador debe estar formado por letras, dígitos y guión	Todos los caracteres del Ident. ∈{letras, dígitos, guión} (4)	Alguno de los caracteres del Ident. ∉{letras, dígitos, guión} (5)
El guión no puede estar al principio, ni al final Puede haber varios seguidos en el medio	Identificador sin guiones en los extremos y con varios consecutivos en el medio (6)	Identificador con guión al principio (7) Identificador con guión al final (8)
Debe contener al menos un carácter alfabético	Al menos un carácter del Ident. ∈{letras} (9)	Ningún carácter del Ident. ∈{letras} (10)
No puede usar palabras reservadas	El Identificador ∉{palabras reservadas} (11)	un caso por cada palabra reservada (12,13,14….)

Derivación de los casos de prueba

Identificador	Clases de equivalencia cubiertas	Resultado
Num-1---d3	1,4,6,9, 11 (todas las válidas)	El sistema acepta el identificador
Nd3	2	Mensaje error
Num-1-letr3--d32	3	Mensaje error
Nu%m-1---d3	5	Mensaje error
-um-1---d3	7	Mensaje error
num-1---d3-	8	Mensaje error
456-1---23	10	Mensaje error
Integer	12	Mensaje error
El resto de palabras reservadas	13,14..	Mensaje error

Ejemplo 2

- El pago retrasado de ciertas facturas mensuales conlleva los siguientes recargos:
  - Si se paga entre el día 1 y 10 no tiene ningún recargo.
  - Si se paga entre el día 11 y 20 tiene un recargo del 2%.
  - Si se paga el día 21 o posterior tiene un recargo del 4%.
- Se ha realizado un programa que a partir de un número entero que representa a un día, visualiza el recargo a aplicar:

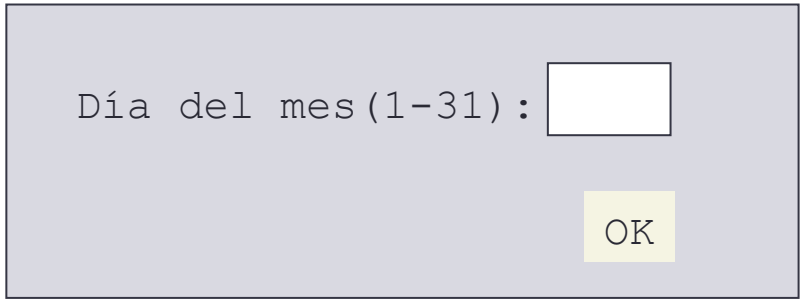


Tabla resultado de clases de equivalencia

Condición entrada	Clases de equivalencia válidas	Clases de equivalencia no válidas
Nº de parámetros	{n=1} (1)	{n<1 (nulo)} (2)
Tipo de parámetros	{dia∈N} (entero) (3)	{dia∈Numérico no natural} (4.1) {dia∈Caracteres} (4.2) (alfabéticos y especiales)
Sin recargo	{dia>0, dia<=10} (5)	{dia<0} (negativo) (6.1) {dia=0} (cero) (6.2)
Recargo 2%	{dia>=11, dia<=20} (7)	
Recargo 4%	{dia>=21, dia<=30} (8)	{dia>30} (9)

Casos de prueba

Entrada	Clases de equivalencia cubiertas	Resultado
4	1, 3, 5	SR
18	1, 3, 7	2%
26	1, 3, 8	4%
	2	Mensaje error
3.8	4.1	Mensaje error
‘ab’	4.2	Mensaje error
-1	6.1	Mensaje error
0	6.2	Mensaje error
32	9	Mensaje error

Prueba de Valores Límite (VL)

- Se basa en la **evidencia experimental** de que **los errores** suelen aparecer con **mayor probabilidad** en los **extremos de los campos de entrada**.
- Un análisis de las condiciones límite de las clases de equivalencia aumenta la eficiencia de las pruebas.
  - **Condiciones límite**: valores justo por encima y por debajo de los márgenes de la clase de equivalencia.

Derivación de las casos de prueba VL

- Generar tantos casos de prueba como sean necesarios para ejercitar las **condiciones límite de las clases de equivalencia (válidas).**
- Proceso heurístico.
- Como en el caso anterior, se pueden seguir unos criterios que faciliten su obtención.

Obtención de los casos de prueba (VL)

Condición entrada	Ejemplo	Caso de prueba
Un valor específico	"..Introducir **tres** valores.."	1 caso que ejercite el valor numérico (15,3,4) 1 caso que ejercite el valor justo por encima (15,3,4,4) 1 caso que ejercite el valor justo por debajo(15,3)
Un rango de valores	...Valores **entre 0 y 10**…	1 caso que ejercite el valor mínimo (0) 1 caso que ejercite por encima del mínimo (1) 1 caso que ejercite por debajo del mínimo (-1) 1 caso que ejercite el valor máximo (10) 1 caso que ejercite un valor por encima del máximo (11) 1 caso que ejercite un valor por debajo del máximo (9)
Elementos de un conjunto tratados diferente por el programa	**Las personas menores de 25 años tendrán una bonificación del 10%**	1 caso que cumpla la condición (25) 1 caso que ejercite el valor justo por encima (26) 1 caso que ejercite el valor justo por debajo(24)

Ejemplo VL ejercicio “Identificadores”

Condición entrada	Clases de equivalencia válidas	Clases de equivalencia no válidas	Valores límite
Entre 5 y 15 caracteres	5≤nº caracteres Ident. ≤15 (1)	nº caracteres Id<5 (2) 15< nº caracteres Id (3)	**1** caso con nº de caracteres 15 (4) **1** caso con nº de caracteres 16 (5) **1** caso con nº de caracteres 14 (6) **1** caso con nº de caracteres 5 (7) **1** caso con nº de caracteres 6 (8) **1** caso con nº de caracteres 4 (9)
El identificador debe estar formado por letras, dígitos y guión	Todos los caracteres del Ident. ∈{letras, dígitos, guión} (10)	Alguno de los caracteres del Ident. ∉{letras, dígitos, guión} (11)	
El guión no puede estar al principio, ni al final Puede haber varios seguidos en el medio	Identificador sin guiones en los extremos y con varios consecutivos en el medio (12)	Identificador con guión al principio(13) Identificador con guión al final (14)	
Debe contener al menos un carácter alfabético	Al menos un carácter del Ident. ∈{letras} (15)	Ningún carácter del Ident. ∈{letras} (16)	
No puede usar palabras reservadas	El Identificador ∉{palabras reservadas} (17)	un caso por cada palabra reservada (18,19,20….)	

Derivación de los casos de prueba ejercicio “Identificadores”

Identificador	Clases de equivalencia cubiertas	Resultado
Num-1---d3	1,10,12,15, 17 (todas las validas)	El sistema acepta el identificador
Nd3	2	Mensaje error
N123456789ABCDE1234	3	Mensaje error
N123456789ABCDE	4	El sistema acepta el identificador
N123456789ABCDEF	5	Mensaje error
N123456789ABCD	6	El sistema acepta el identificador
N1234	7	El sistema acepta el identificador
N12345	8	Mensaje error
N123	9	El sistema acepta el identificador
Nu%m-1---d3	11	Mensaje error
-um-1---d3	13	Mensaje error
num-1---d3-	14	Mensaje error
456-1---23	16	Mensaje error
Integer	18	Mensaje error
El resto de palabras reservadas	19,20..	Mensaje error