

# Encontrar fallos de software

Gestión de la Calidad del Software

La mayoría de los programadores comienzan con una definición falsa del término:

- ✓ Testing es el proceso consistente en demostrar que no hay errores.
- ✓ El propósito del testing es mostrar que un programa lleva a cabo sus funcionalidades correctamente.
- ✓ Testing es el proceso de establecer una cierta confianza en que el programa hace lo que se supone que debe hacer.

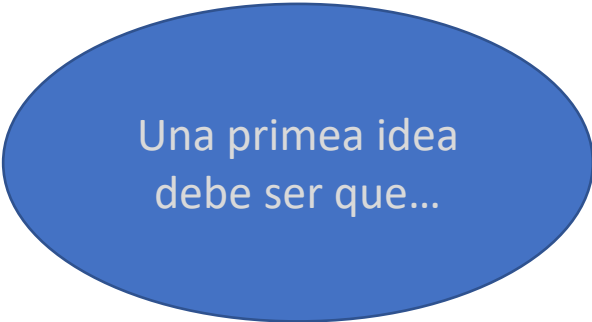
Estas definiciones no son adecuadas para describir el proceso de testing.

# Testing de software

Testing debe aumentar la **calidad** y **fiabilidad** del programa.

Aumentar la fiabilidad significa que debemos **encontrar** y **corregir** errores.

Al comenzar a testear un programa hay que empezar asumiendo que el programa contiene errores y testearlo con el objetivo de encontrar los más posibles.



Una primea idea  
debe ser que...

*“Testing consiste en ejecutar  
un programa con la intención  
de encontrar errores”*

*Manuel Núñez*

Si nuestro **objetivo** es demostrar que un **programa no tiene errores**, entonces tenderemos a seleccionar **tests** que tengan una **probabilidad baja** de causar que el **programa falle**.

Si nuestro **objetivo** es demostrar que un **programa tiene errores**, entonces tenderemos a seleccionar **tests** que tengan una **probabilidad alta** de **encontrar errores**.

Obviamente, la segunda aproximación será más útil que la primera.

- *Testing consiste en ejecutar un programa con la intención de encontrar errores.*

# Testing de software

Demostrar que no hay errores en un programa es, virtualmente, imposible, incluso para programas triviales.

Si nos quedamos con una definición en la que testing se asocia con el proceso de descubrir errores entonces conseguimos que el testing sea una tarea realizable.

# Testing de software

Incluso los programas que *hacen lo que se supone que deben hacer* pueden tener errores.

Hay errores si un programa hace algo que se supone que no debe hacer.

# Testing de software

En resumen:

Tenemos, en principio, una versión del testing como un proceso destructivo consistente en encontrar errores (cuya presencia es asumida) en un programa.

La aplicación de un test es exitosa si hace que el programa falle (e.g. devuelva un valor inesperado, produzca una excepción no prevista, etc).

Pero como ya hemos dicho, esta definición puede ser matizada y mejorada si consideramos una definición del término basada en nivel de pensamiento.

# Definición de testing por niveles

Nivel 0: No hay diferencia entre testing y debugging.

Nivel 1: El propósito de testing es mostrar corrección.

Nivel 2: El propósito de testing es mostrar que el software no funciona.

Nivel 3: El propósito de testing no es probar nada específico, sino reducir los riesgos asociados con la utilización del software.

Nivel 4: Testing es una disciplina mental que ayuda a que los profesionales desarrollen software con más calidad.



# Nivel 0 de pensamiento

**No hay diferencia entre testing y debugging.**

Testing es lo mismo que *debugging*.

**No** se distingue entre comportamiento incorrecto y errores en el programa.

**No** ayuda a desarrollar software fiable y/o seguro.

## **El propósito de testing es mostrar corrección.**

- Desgraciadamente, la corrección es imposible de conseguir.
- Si no detectamos fallos, ¿tenemos buen software o malos tests?

# Nivel 2 de pensamiento

**El propósito de testing es mostrar que el software no funciona.**

El propósito es mostrar fallos.

Buscar fallos es una actividad con negatividad.

Enfrenta a los testers con los desarrolladores.

¿Qué ocurre si no hay fallos?

# Nivel 3 de pensamiento

**El propósito de testing no es probar nada específico, sino reducir los riesgos asociados con la utilización del software.**

El testing solo puede mostrar la presencia de fallos.

Siempre que usamos un software debemos asumir un cierto riesgo.

Este riesgo puede ser pequeño y sus consecuencias irrelevantes.

Sin embargo, este riesgo también puede ser grande y tener consecuencias catastróficas.

Testers y desarrolladores deben cooperar para reducir riesgos.

# Nivel 4 de pensamiento

**Testing es una disciplina mental que ayuda a que los profesionales desarrollen software con más calidad.**

- El testing es solo un método para aumentar la calidad.
- Los ingenieros de testing pueden llegar a liderar proyectos.
- La principal responsabilidad es medir y mejorar la calidad del software.
- Su experiencia debería ayudar a los desarrolladores.

- *Una parte necesaria a la hora de definir un test consiste en proporcionar el resultado esperado.*

Por tanto, cada test debe tener dos componentes:

- Una enumeración de los valores de los inputs del programa.
  - Una definición precisa del output correcto para esos valores de los inputs.
- *Un programador debería evitar testear sus propios programas.*

Normalmente no estamos preparados (psicológicamente) para buscar errores en nuestro propio trabajo.

En particular, el programa puede tener errores debido a que el programador no ha comprendido el enunciado del problema o su especificación.

# Nociones a tener en cuenta a la hora de testear

- *La organización que desarrolla un programa no debería testear sus propios programas.*

El argumento es similar al anterior.

Además, el proceso de testing se puede llegar a ver como responsable de no completar el producto a tiempo y aumentar su coste.

- *Estudia detalladamente los resultados de cada test.*

Hay errores que aparecen en tests que no se percibieron en tests que se habían aplicado con anterioridad.

- *Los test se deben diseñar tanto para cubrir inputs válidos y esperados como para llevar valores inválidos o inesperados.*

# Nociones a tener en cuenta a la hora de testear

- *Testear un programa para ver si no hace lo que se supone que debe hacer es solo la mitad del proceso; la otra mitad consiste en ver si el programa hace algo que no debería hacer.*
- *Evita utilizar tests desechables.*

Si tenemos que testear el programa de nuevo entonces tenemos que reinventar los tests.

El *retesting* de un programa no suele ser tan riguroso como la primera vez de forma que, si una modificación causa un error en una funcionalidad anterior del programa, es fácil que el error no se detecte.

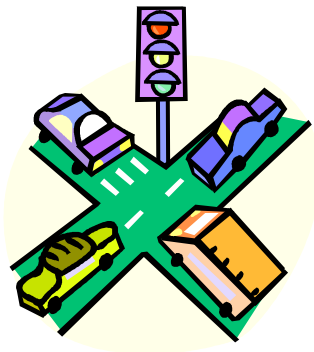
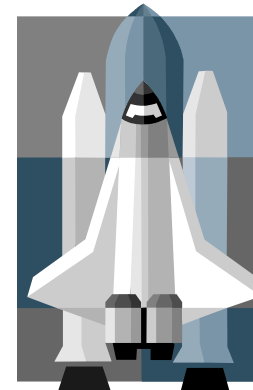
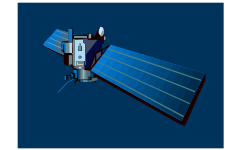
Guardar tests y ejecutarlos tras cambios en otras componentes del programa se conoce como regression testing.



- *No planee un proceso de testing asumiendo que no encontrará errores.*
- *La probabilidad de encontrar más errores en una componente es proporcional al número de errores que ya se han encontrado en esa componente.*

- Las metodologías de desarrollo ágiles ponen una presión adicional a los testers
- Los programadores deben realizar testing unitario (unit testing) sin haber recibido ni entrenamiento ni formación.
- Los tests son fundamentales para comprobar los requisitos funcionales (¿qué hace el sistema). Pero, ¿Quién escribe estos tests?

# Software está presente en todas partes



# Faults, errors & failures

Estos tres conceptos son fundamentales:

**Software Fault (defecto):** Un defecto estático en el software.

**Software Error (error):** Un estado interno incorrecto que es la manifestación de un defecto.

**Software Failure (fallo):** Comportamiento externo incorrecto con respecto a los requisitos o descripción del comportamiento esperado.

# Faults, errors & failures: Ejemplo

- Un paciente le enumera al médico una lista de síntomas.
  - Failures.
- El médico intenta diagnosticar la causa de la dolencia.
  - Fault.
- El médico puede buscar condiciones anómalas internas (presión arterial alta, arritmias, bacterias en el torrente sanguíneo).
  - Errors.

La mayoría de los problemas médicos provienen o de ataques externos (bacterias, virus) o de la degradación natural debida al envejecimiento.

Software faults estaban allí desde el principio y no “aparecen” cuando una componente se “gasta”.

# Otro ejemplo

**Fault: La búsqueda debería empezar en 0, no en 1**

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

## Test 1

[ 2, 7, 0 ]

**Esperado: 1**

**Observado: 1**

## Test 2

[ 0, 2, 7 ]

**Esperado: 1**

**Observado: 0**

**Error: i es 1, no 0, en la primera iteración**

Failure: ninguno

**Error: i es 1, no 0**

El error se propaga a la variable count

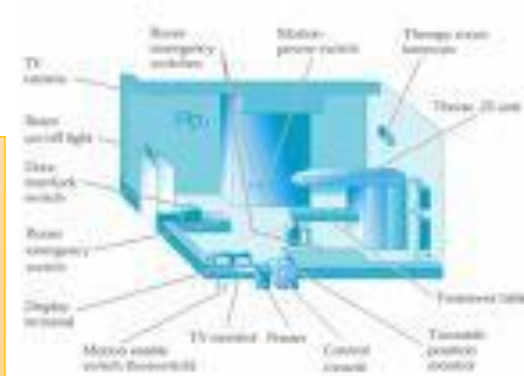
Failure: count vale 0 en el return

El Mars lander de la NASA. Septiembre de 1999, se estrelló debido a un fault (defecto) al integrar unidades.

**Therac-25** (máquina de radioterapia). Testing deficiente de *safety-critical* software puede costar vidas: 3 pacientes murieron.

Necesitamos que el software sea *fiable* (dependable).

Testing es una forma de evaluar la fiabilidad de un sistema.



## Diseño de THERAC-25

# Apagón Noreste de Norteamérica (2003)

Se apagaron 508 generadores y 256 centrales eléctricas.

Afectó a 10 millones en Ontario, Canadá.

Afectó a 40 millones en 8 estados de USA.

Se perdieron \$6.000.000.000.



El sistema de alarma del sistema de gestión energética falló debido a un error de software y los operadores no fueron informados de la sobrecarga del sistema.



El informe del NIST (*National Institute of Standards and Technology*) “*The Economic Impacts of Inadequate Infrastructure for Software Testing*” (2002) afirmaba:

- Procesos inadecuados de testing le cuestan a USA entre \$22 y \$59 miles de millones al año.
- Mejores aplicaciones podrían reducir esta cifra a la mitad.

Pérdidas enormes debidas a fallos en aplicaciones web:

- Servicios financieros: \$6.5 millones por hora (solo en USA).
- Aplicaciones dependientes de compras con tarjeta de crédito: \$2.4 millones por hora (de nuevo, considerando solo USA).

En diciembre de 2006, una oferta BOGO (Buy One Get One) de amazon.com dio lugar a un descuento doble.

En 2007 Symantec concluyó que la mayoría de las vulnerabilidades del software se deben a software con fallos.

- La seguridad está en la actualidad asociada con fallos de software.
  - Un software seguro es un software fiable.
- La web ofrece una nueva plataforma de implantación.
  - Muy competitiva y disponible para muchos usuarios.
  - Las aplicaciones web apps están distribuidas y deben ser muy fiables.

# Validación y verificación (IEEE)

Complementando al término testing conviene conocer, y distinguir adecuadamente entre ellos, estos dos términos.

**Validación:** El proceso de evaluar el software al final de su desarrollo para garantizar conformidad con respecto al uso deseado.

**Verificación:** El proceso de decidir si el producto, en una determinada fase del ciclo de desarrollo del software, cumple los requisitos establecidos durante la fase anterior.

Testing se considera habitualmente una actividad de validación.

# Objetivos tácticos. ¿Por qué usamos un test?

Los objetivos de test y los requisitos deben estar bien documentados.

¿Cuánto testing es suficiente?

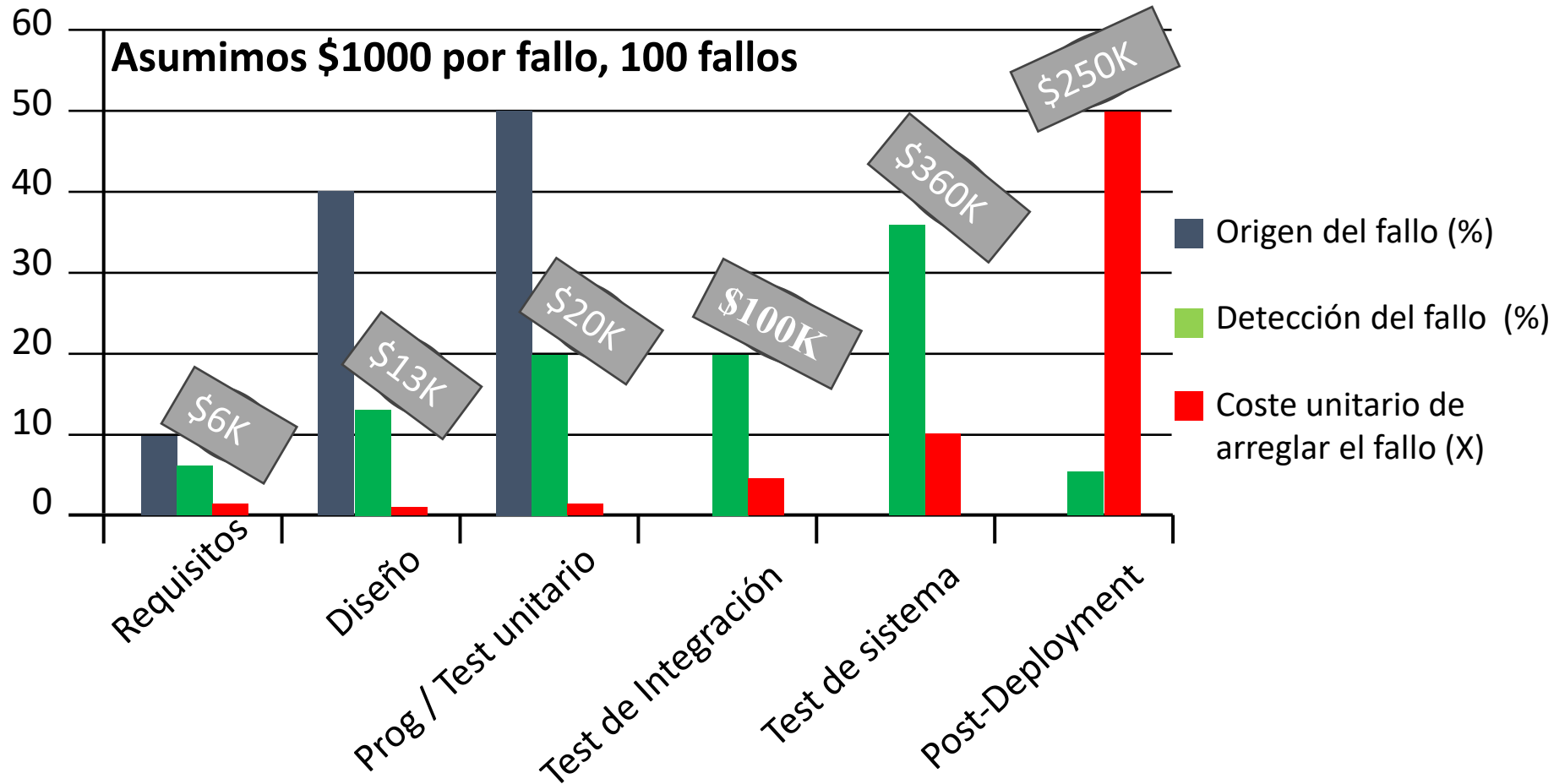
Objetivo común: gastar el presupuesto... testear hasta el último día...

Esta aproximación se suele llamar “criterio de fecha”.

# El precio de no testear

- Testing es la actividad más cara (al menos el 50% del presupuesto) y que más tiempo conlleva durante el desarrollo de software.
- Pero no testear es todavía más caro.
- Si no nos esforzamos en hacer testing al principio, entonces el coste aumenta.
- Planificar el testing para después de finalizar el desarrollo es prohibitivamente caro.

# El precio de testear tarde



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

# Conclusión: ¿Por qué testear el software?

El objetivo del testeador debe ser eliminar los fallos (faults) lo antes posible.

Incrementa la calidad.

Reduce el coste.

Mantiene/incrementa la satisfacción del cliente.

¿Por qué?