Tutorial: pruebas funcionales y trabajo en equipo

Luis Fernández Sanz
Universidad Europea de Madrid
Coordinador grupo de Calidad de Software de ATI

Objetivos

- Lógicamente limitados
- Algunos de ellos:
  - Conciencia de los distintos aspectos y la situación de las pruebas en general
  - Revisar definiciones y conceptos básicos
  - Diseño básico funcional
  - Conciencia de las especificaciones como clave
  - Conciencia del trabajo en equipo
  - Conocer las posibilidades de generar casos funcionales

Pruebas

Si tienen interés en ahorrar costes,

ENHORABUENA

han elegido la mejor opción:

LAS PRUEBAS DE SOFTWARE

Pruebas de software

- Gasto medio sobre proyecto de pruebas y depuración sobre costes de proyecto:
  - 33% (The Original Software Group): 130 compañías
  - 35,1% (*ISE-TR-06-03*): 12 compañías
  - 33% de gasto y 85% de eficiencia de detección (C.Jones, 1998)
- NIST 7007-011:
  - Coste de pruebas ineficientes USA: 21.200 M$
  - Reducción de costes potencial: 10.700 M$
- Eficiencia:
  - 13,5 % eficientes, 19% automatizan

Sondeo sobre prácticas de pruebas

Rellenar el cuestionario adjunto anónimo

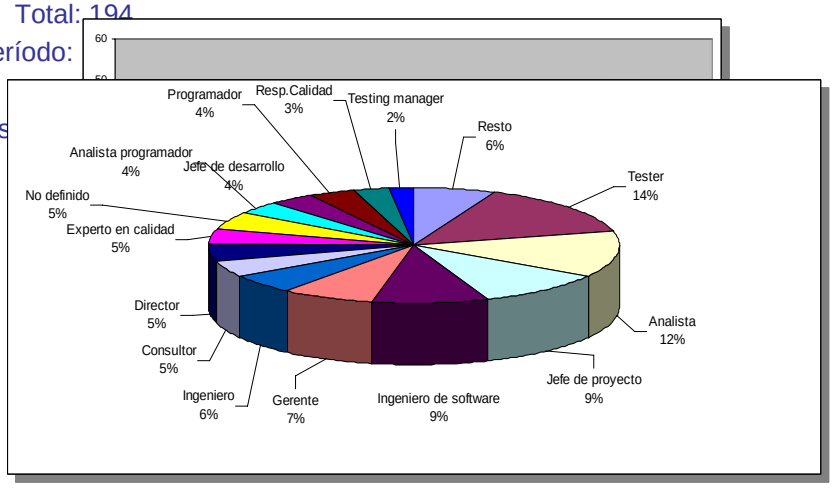
Aprovechar a meditar sobre nuestro proceso

Obtener una autoevaluación comparativa

Si me lo permiten, ampliaré el número de respuestas del sondeo

Muestra

- Muestra:
  - Total: 194
- Período:
  -
- Dis
  -
  -
  -
  -
  -
  -
  -

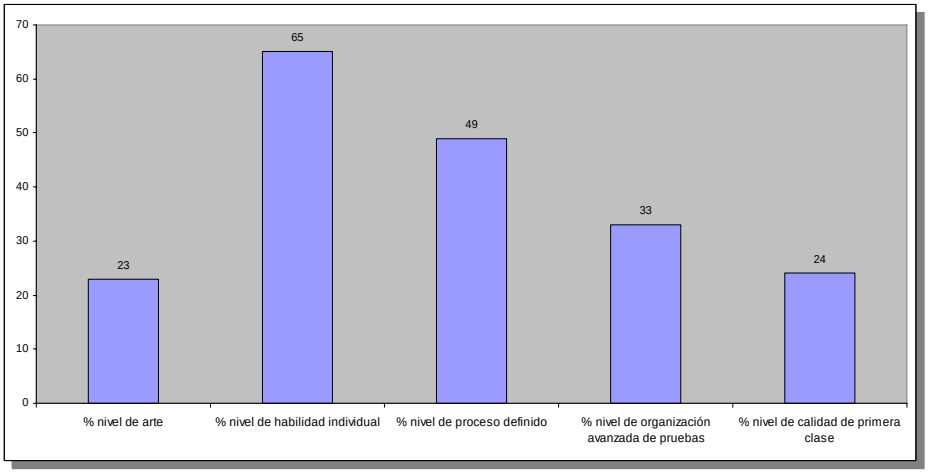


Procesos de prueba

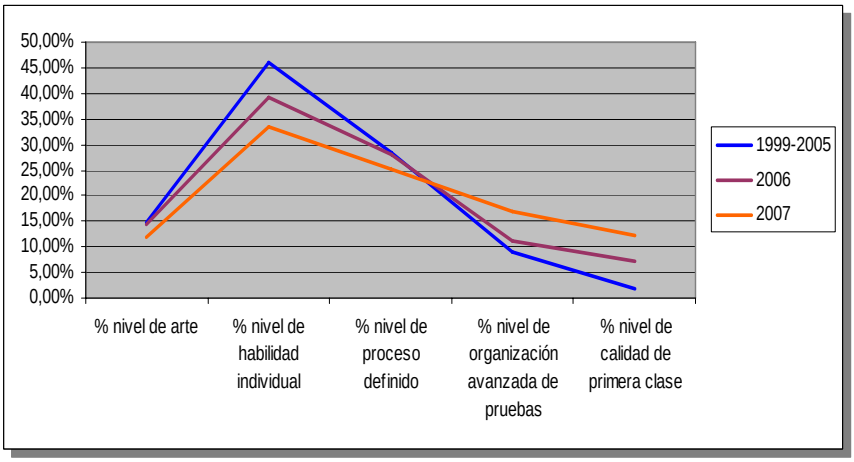
- El cuestionario se basa en un modelo del QAI
- Media de resp. positivas: 8,45
  - Varianza: 4,07

Proceso como arte: 15,13%
Habilidad individual: 42,76%
Proceso definido: 32,24%
Organización avanzada: 21,71%
Calidad de primera clase: 15,79%

Análisis de procesos



Evolución



Preguntas detalladas

Aspectos más implantados	Hasta 2006	Hasta 2007
7. (¿Se valida que, además de bien implementadas, se cumplan las expectativas del cliente?)	53,6%	61,84%
9. (¿Personal de pruebas informa de defectos al equipo de desarrollo y no a otros como dirección?)	40%	52,63%
6. (¿Se valida que las especificaciones están correctas?)	40%	48,68%

Preguntas detalladas

Aspectos menos implantados	Hasta 2006	Hasta 2007
18. (¿Se usan métricas (p.ej., defectos/KLOC) para planear y evaluar los procesos de pruebas?)	6,4%	9,87%
20. ¿Supone el uso de herramientas automatizadas de prueba un componente significativo del proceso de pruebas?	18,37%	12,50%
10. (¿El personal de pruebas identifica los riesgos de negocio antes de desarrollar el plan de pruebas?)	12,8%	13,16%

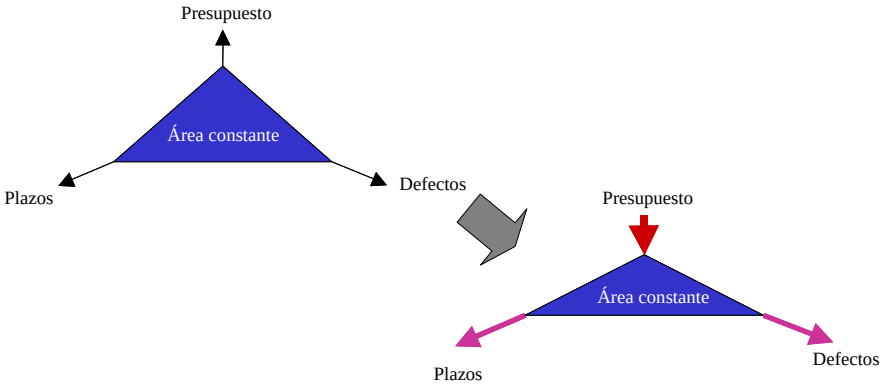
Formación

- Profesionales recibieron formación específica de pruebas:
  - 26,97% (23,08% hasta 2006)
  - 6,74% no contesta
- Estudio desde 2003:
  - Preguntas de conceptos básicos

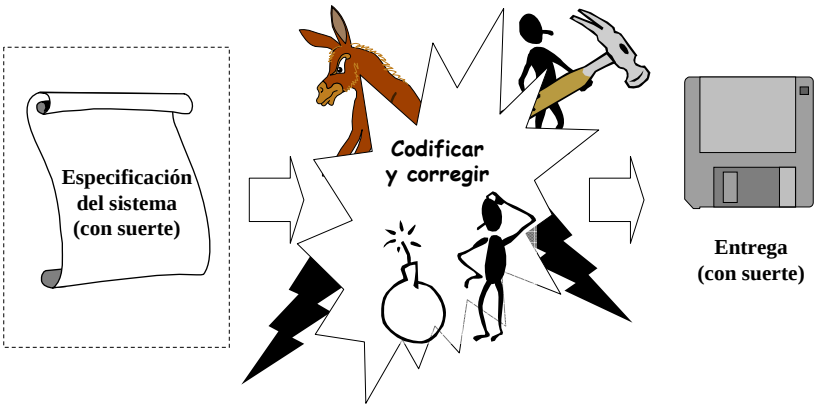
Preguntas	Sin formación	Con formación
P1	35,59%	45,83%
P2	25,42%	41,67%
P3	28,81%	54,17%
P4	10,17%	12,50%
P5	20,34%	33,33%

Equilibrio

Principio del triángulo (¿McConnell, 1997?)



Huir del codificar y corregir



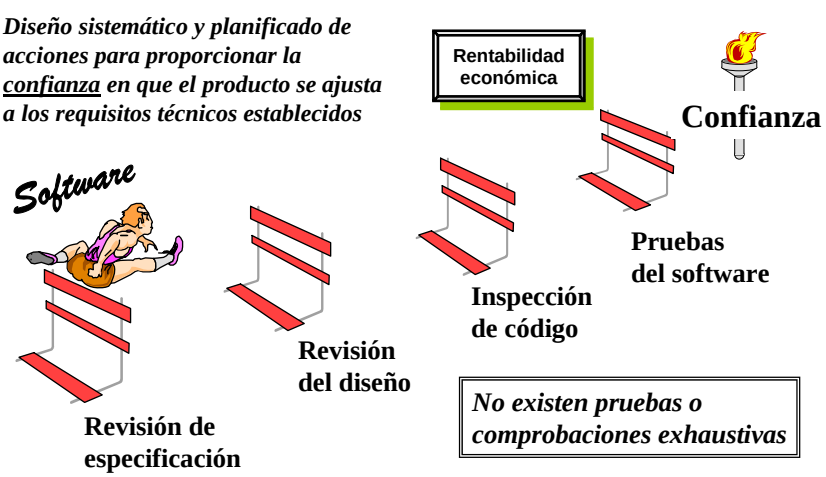
Evaluación de software (II)

- Es muy complejo centrarse sólo en el producto final

- Controlar en todo el proceso de producción:
  - Aseguramiento de calidad
    - Control de productos
    - Control de procesos
  - IEEE 830:
    - Gestión de configuración, V&V (revisiones y pruebas), medición

Enfoque integral de aseguramiento de calidad de software

Diseño sistemático y planificado de acciones para proporcionar la <u>confianza</u> en que el producto se ajusta a los requisitos técnicos establecidos



Peter Neumann: Inside Risks

- Barcla
- Un gr
  uno d
- Un de
  encar
- El nue
  a inoc
- Un de
- Un pe
  ganad
- Un fa
  aleatoriamente pero el 9 nunca sale (1994)

Defecto de Ariane: 10 años y 7000 millones de $

39 sg. después del lanzamiento, a una altura de 4,5 km, un mecanismo de autodestrucción acabó con el Ariane 5 y su carga de 4 caros y no asegurados satélites científicos. Cuando el ordenador intentó convertir un dato de velocidad lateral del cohete de 64 bits a 16 bits hubo error de *overflow*. Se había decidido que este número nunca sería tan grande porque no lo había sido con el Ariane 4. Pero el Ariane 5 era más rápido.

Algo absurdo adicional: el cálculo contenía un defecto que tumbó el sistema de guía (que confundió al ordenador de a bordo, que forzó al cohete fuera de ruta) sólo servía para alinear el cohete antes del lanzamiento pero se decidió, mucho antes (en versiones anteriores de Ariane) que siguiera funcionando los primeros 40 segundos de vuelo.

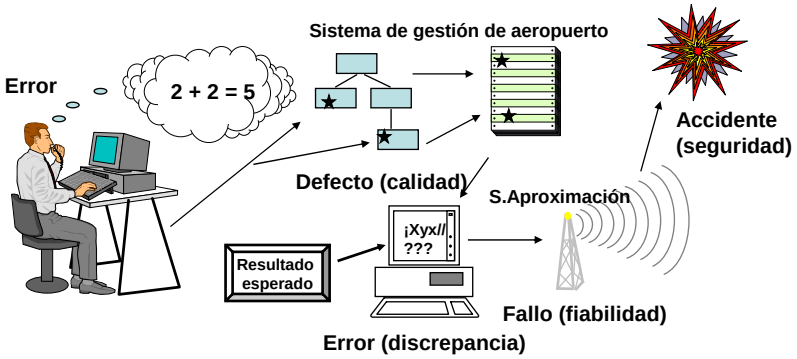
http://www.csl.sri.com/users/neumann/illustrative.html

Definiciones

- Pruebas
  - Ejecución de software con datos controlados (casos) con el fin de descubrir defectos
- Caso de prueba
  - Conjunto específico de entrada, procedimientos y salida esperada para una situación de operación
    - Salida esperada: especificada (¿diseñada?¿esperada por el usuario?)
- Pruebas como una técnica más de V y V
  - Verificar: ¿construir correctamente el producto? Comprobar coherencia en el ciclo de vida
  - Validar ¿construir el producto correcto? Comprobar si satisface los requisitos

Definiciones

- Definición y relación de error (*error*, *mistake*), defecto (*fault*, *defect*) y fallo (*failure*)



Filosofía

- Objetivo
  - Descubrir defectos
- Buen caso de prueba
  - Gran probabilidad de detectar defectos no encontrados antes
- Éxito
  - Descubrir un defecto no detectado

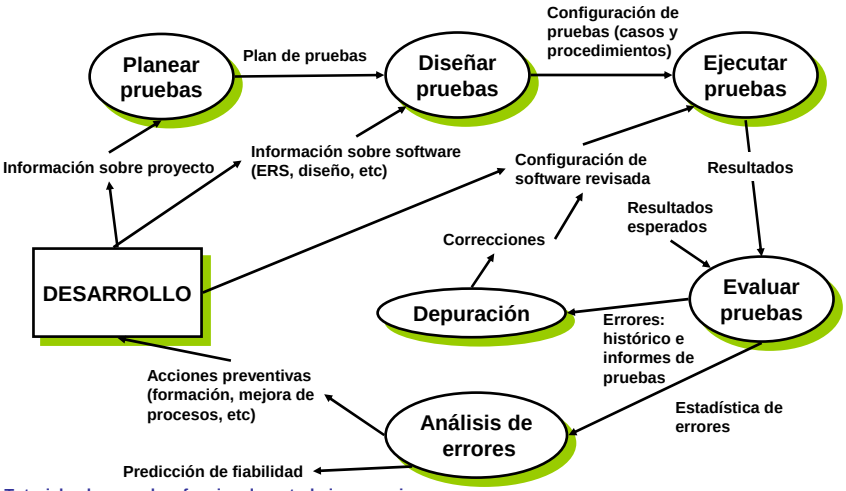
- **Dijkstra: las pruebas no pueden asegurar la ausencia de defectos, sólo demostrar que (los que localizamos) existen en el software**

Problemática

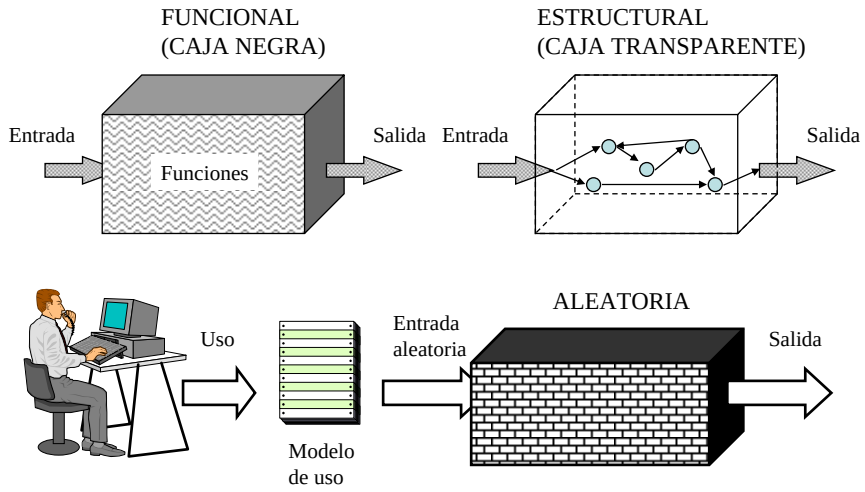
- Actitud frecuente
  - Mito: buen trabajo = cero defectos
  - Defecto = culpa por mal trabajo
  - Descubrir defecto = fracaso
- Actitud apropiada
  - Siempre existen defectos. Tasas tras entrega de software (defects/KLOC)
  - Causa de defecto no es siempre negligencia. Comunicación humana defectuosa
  - Descubrir defectos es un éxito (diagnóstico médico)
    - Evitamos problemas mayores
    - "Pay now or pay much more later"

Ciclo completo de pruebas

- Ciclo básico: planear, especificar/diseñar, ejecutar, comprobar

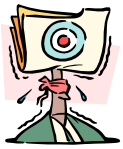


Diseño de pruebas



Enfoque funcional

- Centradas en dominio de entrada y de salida
- Distintas técnicas:
  - Mejores resultados: combinando varias y complementando sus ventajas
- Prueba exhaustiva impracticable
  - A+B (sumandos 2 dígitos y signo) ≅ 40.000 casos
  - Quedan todas las entradas no válidas
- Buscar subconjuntos de casos bien elegidos



Particiones de equivalencia

- Caso bien elegido:
  - Invocar máximo número de entradas (mínimo nº de casos)
  - Partir dominio en clases equivalentes (probar un valor supone lo mismo que probar cualquier otro)
- Etapas:
  1. Identificar clases de equivalencia (tabla de clases):
     - Válidas (entradas válidas)
     - No válidas (errores o entradas no válidas)
  2. Crear los casos de prueba (entrada y salida)

Particiones de equivalencia

- Cada condición o restricción de entrada divide el dominio en dos o más grupos
- Sugerencias para clases:
  - Rango (“identificador entre 1 y 99”):
    - clase válida ($1 \leq id \leq 99$)
    - dos clases no válidas ($id < 1$, $id > 99$)
  - Número de valores (“de 1 a 3 beneficiarios”)
    - clase válida ($1 \leq$ nº benefic. ≤ 3)
    - dos clases no válidas (nº ben. < 1, nº ben. > 3)
  - Situación booleana (“primer carácter es letra”)
    - clase válida (car. = letra)
    - clase no válida (car. $\neq$ letra)

Particiones de equivalencia

- Sugerencias para clases:
  - Conjunto de valores (“vehículo puede ser moto, coche, camión”) con distinto tratamiento (“tipo de seguro”):
    - una clase válida para cada uno
      - moto
      - coche
      - camión
    - una clase no válida (p.ej., “trailer”)
  - Regla genérica:
    - todos los elementos con mismo tratamiento
    - si no, dividir la clase en otras menores

Particiones de equivalencia

- Reglas para diseño de casos:
  - Asignar identificador a cada clase
  - Hasta incorporar todas las clases válidas:
    - crear un nuevo caso que cubra tantas como sea posible
  - Hasta incorporar todas las clases no válida:
    - crear un nuevo caso que cubra una y sólo una clase no válida

Se trata de evitar que unos errores enmascaren a otros o invaliden otros controles similares

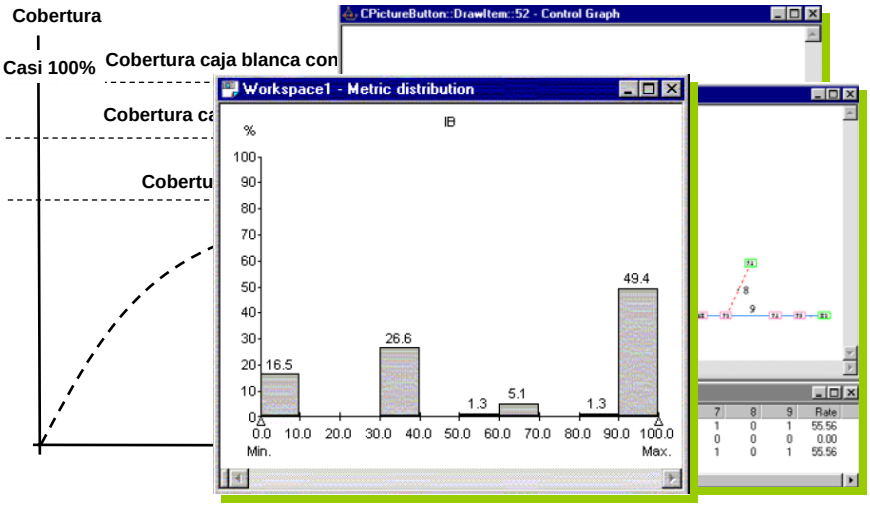
Ejemplo de formateo numérico

- Prueba funcional:
  - Selección de valores límite
  - Tratamiento de combinaciones
- Especificación de entrada y de salida esperada:
  - Entrada: 25 caracteres
    - >1 dígito, ≤ 14 enteros, ≤ 4 decimales
    - Combinaciones de signo, coma y puntos
    - Posibles espacios a dcha o izda (no interemedios)
    - Vale: “+0”, “1234,”, “-,012”, “12.345,6”
  - Salida: PIC S9(14)V9(4)
    - Nº enteros, Nº decimales, signo (T/F), coma (T/F), puntos (T/F)
    - Códigos de error

Enfoque recomendado

- Diseño de caja negra:
  - Combinaciones de entrada
  - Siempre análisis de valores límite
  - Identificar clases válidas y no válidas y completar casos
  - Añadir casos por conjetura de errores
- Ejecutar casos
- Controlar cobertura:
  - Manera de verificar que nos acercamos continuamente al objetivo y que lo alcanzamos
  - Ratio: Nº Objetos Probados/Nº Objetos totales a probar
- Añadir casos para lograr cobertura prevista

Enfoque recomendado



Recomendaciones generales

- ☺ Cada caso debe definir en detalle la salida esperada
- ☺ Evitar que el autor pruebe su software
- ☺ Inspeccionar con detalle la salida obtenida
- ☺ Incluir tanto entradas no válidas e inesperadas como válidas y esperadas
- ☺ Comprobar si el software:
  - No hace lo que debe hacer (fallo de funciones)
  - Hace lo que supone no debe hacer (efectos secundarios)

Recomendaciones generales

☺ Evitar utilizar casos desechables (control y economía)

☺ No planear suponiendo ausencia de defectos

☺ Estudios:

- Probabilidad de nuevos defectos es proporcional al número de defectos ya encontrados

☺ Pruebas:

- Tarea creativa (no destructiva)
- Desafío intelectual

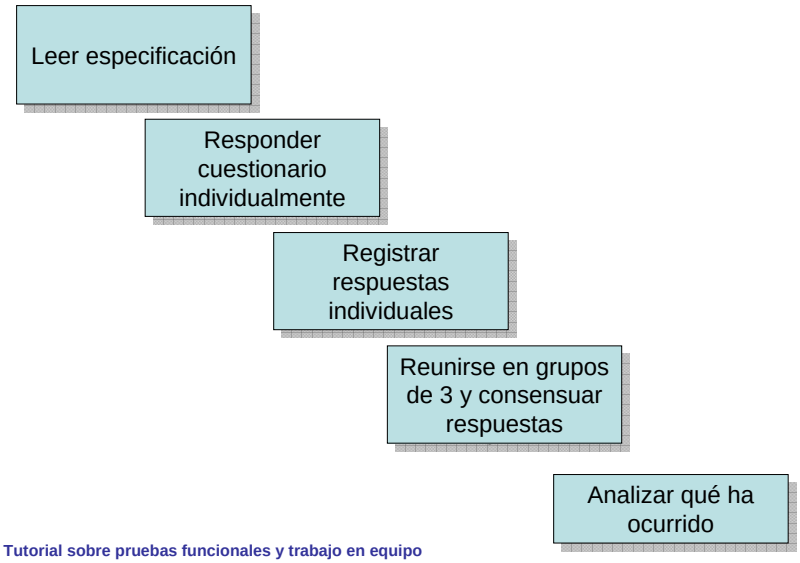
Visión práctica

- El diseño de casos es totalmente dependiente de una buena especificación
  - Muchas veces, el trabajo de pruebas supone hacer el trabajo que no hicieron los analistas
- En cuanto tenemos una buena especificación se pueden diseñar los casos de prueba
  - En especial, si contamos con casos de uso
  - No se diseña justo antes de probar
- Eficiencia y limitación de recursos
  - Sin pruebas perfectas: sólo equilibrio riesgo-coste
  - Coste = 1/ Nº defectos
- El trabajo en equipo puede proporcionar grandes mejoras de eficacia y de eficiencia
- Como última fase, sufren los retrasos de fases previas
  - La planificación de pruebas debe contemplar plazos generosos

Buena especificación → buenas pruebas funcionales (y no funcionales)

- Una buena especificación facilita el diseño de pruebas
  - IEEE 830: no ambigua, completa, fácil de verificar, coherente, fácil de modificar, ordenación por prioridades y/o estabilidad, trazabilidad directa e inversa
- El lenguaje natural es ambiguo:
  - Uso de modelos, técnicas, etc.
- Reaprovechamiento del esfuerzo en modelos para generar pruebas (*model driven testing*)
- Pensar en pruebas como ayuda al análisis y especificación
  - XP, etc: pruebas es parte de la especificación

Especificaciones y trabajo en equipo

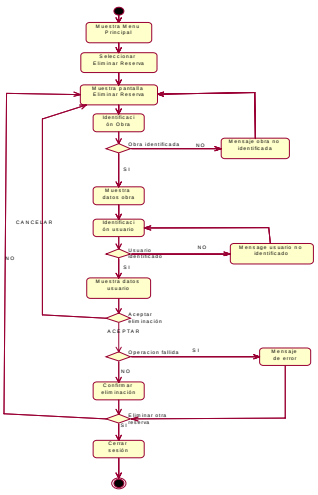


Casos de uso

- Describen interacciones actor-sistema
  - Eficaces para vincular especificación y pruebas
  - Recogen combinaciones y opciones no válidas (flujos alternativos)
- Generar casos de prueba:
  - Caminos/escenarios de ejecución de caso
    - Diagrama de interacción o de estados
  - Sobre cada escenario, diseño de caja negra:
    - Clases de equivalencia y límites
    - Tratamiento de combinaciones
    - Centrado en datos de E/S manejados en el caso

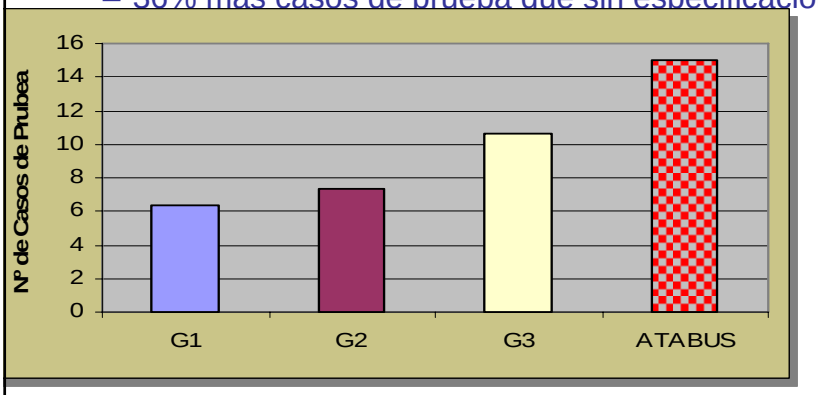
Ejemplo de caso de uso

Bibliotecario	Sistema
	1.- Muestra por pantalla el menú principal
2.- Selecciona la opción *Eliminar Reserva*	3.- Muestra la pantalla de *Eliminar Reserva*
4.- Introduce el *idObra* <extends Buscar Título>	5.- Muestra los datos de la obra y pide el DNI del usuario.
6.- Introduce el DNI <extends Buscar Usuario>	7.- Muestra los datos del usuario y pide confirmar la eliminación reserva
8.- Pulsa *ACEPTAR*	9.- Muestra mensaje de confirmación y pregunta si desea eliminar otra reserva.
10.- Pulsa *NO*	11.- Cierra sesión y muestra menú principal



Diseño de casos

- Estudio previo:
  - Uso de casos de usos y diagramas de actividad
  - Estudiantes y profesionales
  - 36% más casos de prueba que sin especificaciones



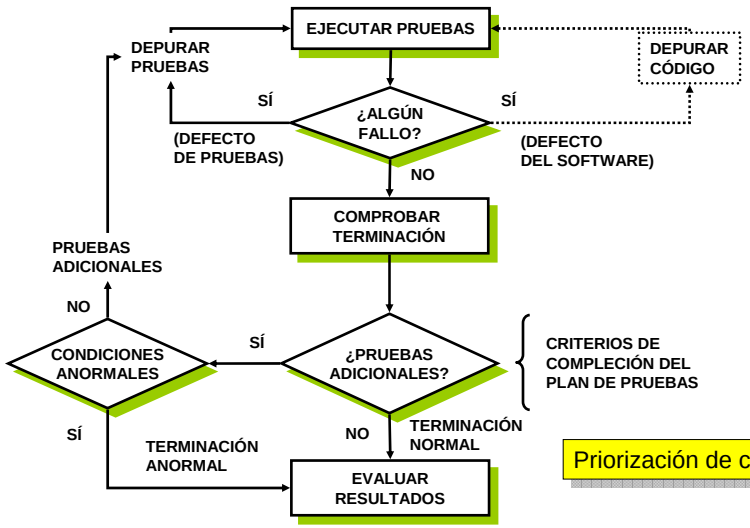
Pruebas en la práctica

- ☹ Pruebas = algo que se hace después de codificar
- ☹ ¿Hay que hacer algo más que ejecutar?
- ☹ Diseño: un poco antes de probar
- ☺ Diseñar lo antes posible
  - ☺ Ayuda a analizar y reflexionar sobre especificaciones
  - ☺ Hay que mantener diseño pero es rentable
  - ☺ Detectar problemas antes de crear sistema
- ☺ Incluso diseñar antes de especificar
  - ☺ Un requisito más que cumplir

Naturaleza de las pruebas

- Ejecución de código:
  - Entorno controlado/observado
    - Reproducibilidad de pruebas
  - Entrada
    - Muestra de la posible entrada (limitaciones)
  - Resultados correctos predichos
    - Comparación con resultados reales
    - Problemas de predicción o especificación
  - Salida analizada
    - Evidente aunque descuido por volumen/repetición

Ejecución



Priorización de casos

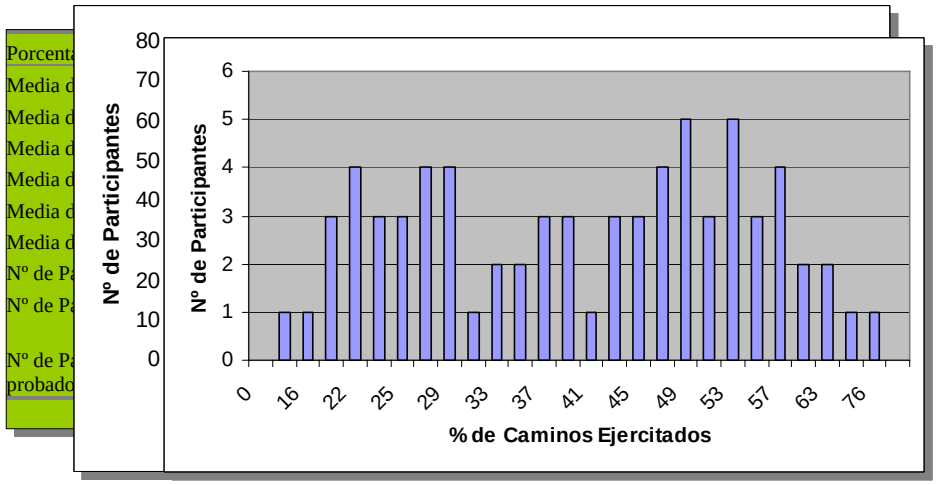
Análisis del diseño de casos

- Programa sencillo de gestión de datos (DVD):
  - http://esp.uem.es/testRecord/pro
  - Especificación y registro de pruebas
  - Comparar con solución correcta
  - 71 participantes con datos fiables
- Análisis de selección sobre 100 expertos:
  - Casos propuestos vs solución
  - Prioridad de casos y eficiencia
  - Opinión

¿Cómo se diseñan? (I)

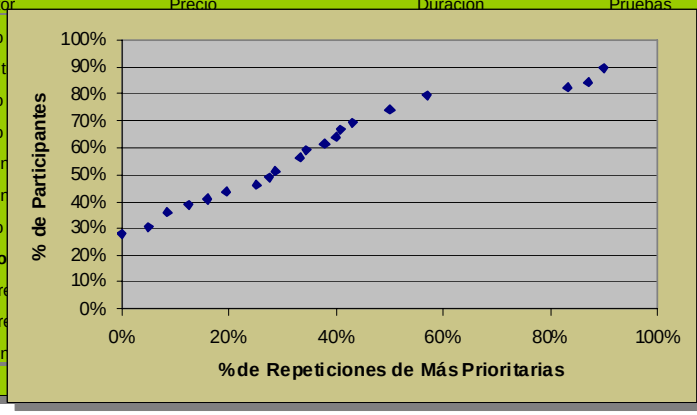
- Promedio: 42,36% de opciones
- Casos repetidos (total):
  - Inserción: 135% adicional
  - Consulta: 15,4%
  - Borrado: 13,8%
- Resultados diferenciados:
  - Por grupos de participantes
  - Por resultados de diseño y prioridad de casos

¿Cómo se diseñan? (II)



¿Cómo se diseñan? (III)

Operación	Director	Precio	Duración	Nº Pruebas	Prioridad Media
Inserción	Valido				1.3
Inserción	De ent				1.3
Inserción	Valido				1.3
Inserción	Valido				1.3
Consulta	Ningur				1.4
Borrado	Ningur				1.4
Inserción	Valido				1.3
Inserción	**Valido**				**1.6**
Inserción	Indifere				1.2
Inserción	Indifere				1.4
Consulta	Ningur				1



Conclusiones de diseño

- De 71 *testers*:
  - 1 probó > 75% de caminos totales
  - 56% alcanzan <50%
- 50% de pruebas probaban caminos ya probados previamente por ese *tester*
- Prioridades:
  - entre 10 caminos más probados sólo aparece uno de los 10 más prioritarios
  - entre los diez caminos menos probados aparecen tres de los 10 caminos valorados como más prioritarios

Conclusiones

- En general, las pruebas concitan olvido y poco entusiasmo
  - Poca formación específica
  - Son el mal necesario y poco estimulante
  - Se busca la panacea rápida
- Mayor concienciación de sistemática de diseño de casos
- Sin embargo, es elemento fundamental
  - Muy dependiente del resto del desarrollo
  - Con mucho sentido común a aplicar

Gracias por su atención

¿preguntas?

Contacto: lufesa@computer.org

Recomiendo:
http://www.ati.es/gtcalidadsoft
http://www.ati.es/reicis
http://esp.uem.es/Testrecord/pro
http://in2test.lsi.uniovi.es/repris/
http://esp.uem.es/rentic

Resultados (IV)

¿Utiliza casos de uso para completar la especificación del sistema?	Sí (53.13%)	No (28.13%)	Depende (18.75%)		
¿Utiliza algún diagrama UML	Sí (46.88%)	No (34.38%)	Depende (18.75%)		
Utilidad del uso del"Algoritmo Automático" para el diseño y ejecución de pruebas	Nada útil (3.33%)	Poco Útil (10%)	Útil (30%)	Muy Útil (46.67%)	Imprescindible (10%)
¿Sería rentable en su organización?	Sí (76%)	No (24%)			
En la simulación, los casos generados automáticamente suponen una cobertura del código de	Menos del 20% (16%)	Entre 20 y 50% (4%)	Entre 50 y 80% (52%)	Mas del 80% (28%)	
En el experimento, los casos generados automáticamente, sobre el total posible, cubren	Menos del 20% (8%)	Entre 20 y 50% (8%)	Entre 50 y 80% (44%)	Mas del 80% (40%)	