Pruebas del Software: Descubrir Errores y Más...

Javier Tuya

Universidad de Oviedo, Dpto. de Informática

Grupo de Investigación en Ingeniería del Software

http://www.di.uniovi.es/~tuya/

Universidad de Pablo de Olavide,

21 de Abril de 2008

Red para la Promoción y Mejora de las Pruebas en Ingeniería del Software

in2test.lsi.uniovi.es/repris

giis.uniovi.es

Introducción

- “All in all, coders introduce bugs at the rate of 4.2 defects per hour of programming. If you crack the whip and force people to move more quickly, things get even worse”
  - Watts Humphrey (note), http://www.cs.usask.ca/grads/jpp960/490/BombSquad.html

- 20 por Día / 100 por Semana / 400 por Mes / 5000 por Año
- “5000 Defect Project (not atypical for IBM)”
  - Paul Gibson, Testing Challenges for IBM, UK Test 2005 Keynote, http://www.uktest.org.uk

Introducción. Algunas cifras

- Total de recursos empleados en pruebas:
  - 30% a 90% [Beizer 1990]
  - 50% a 75% [Hailpern & Santhanam, 2002]
  - 30% a 50% [Hartman, 2002]
- Mercado de herramientas: $2,6 billion
  - Coste por infraestructura inadecuada:
    - Transporte y manufactura: $1,840 billion
    - Servicios financiero: $3,342 billion
  - Costes de reparación en función del instante en el ciclo de vida [Baziuk 1995]
    - Requisitos: x 1
    - Pruebas de sistema: x 90
    - Pruebas de instalación: x 90-440
    - Pruebas de aceptación: x 440
    - Operación y mantenimiento: x 880
  - Fuente general: The economic impact of inadequate infrastructure for software testing. NIST Report – May 2002

Introducción. Definiciones

- Definición 1: La prueba (testing) es el proceso de ejecutar un programa con la intención de encontrar fallos [Glenford J. Myers]
  - Un buen caso de prueba es el que tiene alta probabilidad de detectar un nuevo error
  - Un caso de prueba con éxito es el que detecta un error nuevo
- Definición 2 [Cem Kaner]:
  - Una investigación técnica del producto bajo prueba
  - ...para proporcionar a los interesados (stakeholders)
  - ...información relacionada con la calidad

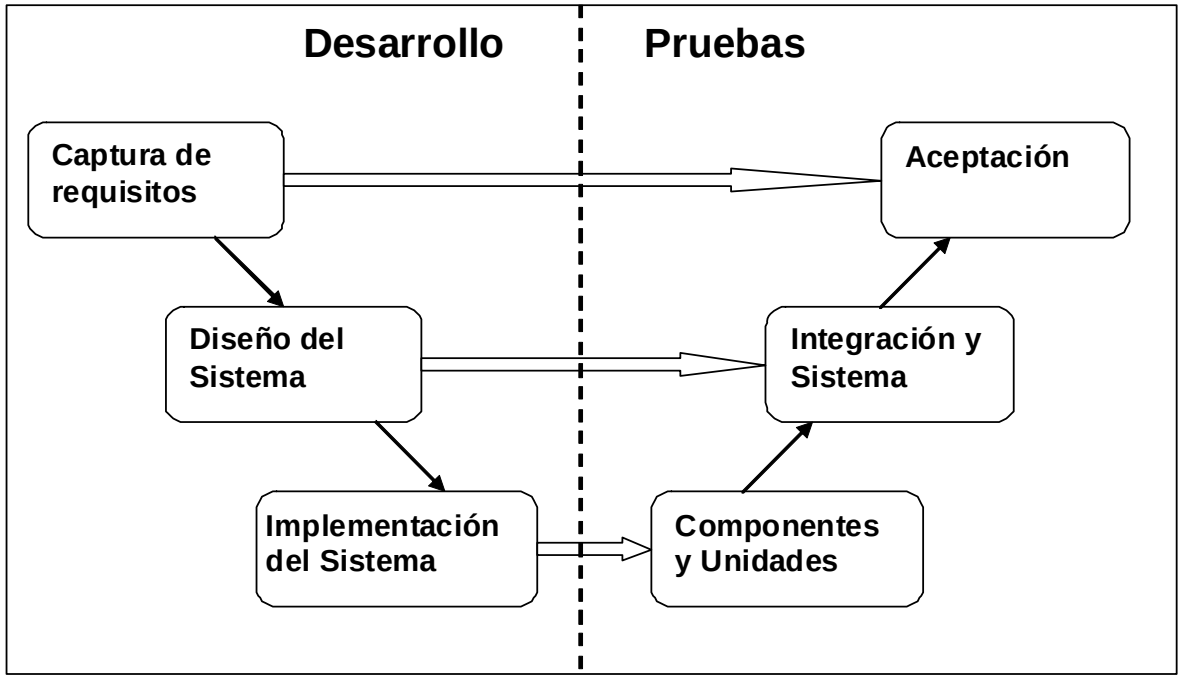
Indice

- El proceso (Modelo en V)
- Técnicas. Caja Blanca y Caja Negra
- Pruebas de Sistema
- Herramientas y Automatización
- Planes de pruebas, personal y costes
- Pruebas sobre bases de datos

Ejercicio introductorio

- Ref: “The Art of Software Testing”, G.J. Myers, 1979,2004
- Especificación:
  - Un programa lee tres valores enteros desde la pantalla.
  - Los tres valores son interpretados de forma que representan las longitudes de los tres lados de un triángulo.
  - El programa muestra un mensaje que indica si el triángulo es escaleno (todos los lados distintos), isósceles (dos iguales) o equilátero (todos iguales)
- Problema:
  - Escribir el conjunto de casos de prueba que se consideren adecuados para probar este programa.

El proceso: Modelo en V (simplificado)



Otros conceptos:

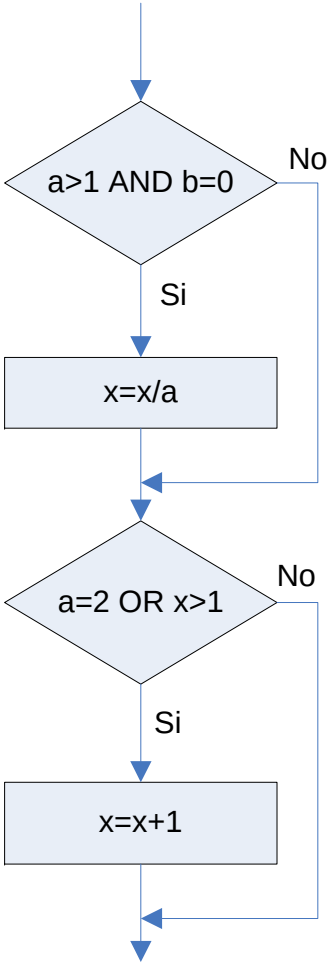
- Pruebas Unitarias (CSI)
- Estrategias de Integración
  - Ascendente. Conductores (test drivers, test harness)
  - Descendente. Resguardos (Stubs, mocks)
- Pruebas de regresión (regression testing)
- Pruebas de humo (smoke testing)

- Mapeo a Métrica V3
  - Componentes: Proceso CSI (Construcción del sistema)
  - Integración y Sistema: Proceso CSI (Construcción del Sistema)
  - Aceptación: Proceso IAS (Implantación y aceptación del Sistema)

Técnicas de prueba (clasificación clásica)

- Caja Blanca:
  - Cuando se diseña la prueba a partir del conocimiento de la estructura interna del código
  - Para ello se utiliza algún criterio de cobertura
  - Problemas: No se prueba la especificación y no se detectan ausencias
- Caja negra:
  - Cuando se diseña la prueba a partir del conocimiento de la especificación, ignorando la estructura interna.
  - Problemas: Infinitas posibilidades para las entradas

Técnicas.
Criterios de Cobertura (caja blanca)



- Cobertura de sentencias (cada línea ejecutada al menos una vez)
  - Caso: a=2, b=0, x=3
  - Qué pasaría si en vez de AND es OR
- Cobertura de decisiones (ramas)
  - Dos casos: a=3, b=0, x=3 ; a=2, b=1, x=1
  - Cubre el anterior
  - Qué pasaría si en la segunda decisión en vez de x>1 debiera ser x<1?
- Otros más completos. Cobertura de:
  - Condiciones
  - Decisión condición, Múltiple condición
  - Decisión/condición modificada (MCDC)
- Cobertura de bucles, Camino básico (complejidad ciclomática)

Técnicas.
Clases de equivalencia (caja negra)

- Procedimiento (simplificado):
  - Identificar condiciones de entrada/salida que influyen en diferentes comportamientos del programa
  - Identificar en cada condición de entrada/salida las categorías que causan comportamientos homogéneos (clases de equivalencia)
  - Crear casos de prueba para cada una de las clases
- Ejemplo: Obtener los recibos devueltos de un socio
  - Condición 1: Tipo del recibo
    - El que puede ser devuelto (domiciliado/reliquidado)
    - El que no (contado)
  - Condición 2: Estado del cobro
    - Que hay que tratar como impagado (devuelto/reclamado)
    - Que no (resto)
  - Condición para las salidas: Número de recibos obtenidos para un socio
    - Que no tenga impagados
    - Que los tenga

Técnicas.
Análisis Valores límite (caja negra)

- Es muy común que los errores se produzcan en valores puntuales de los datos.
- Ejemplo:
  - Especificación: Mostrar solamente recibos devueltos con importe superior 100€ (condición en el programa: importe>100)
    - Condición de entrada: Importe
      - Clase 1: Menor que 100€
      - Clase 2: Mayor que 100€
    - Dos casos, uno por clase: p. e. uno con 90€ y otro con 110€
  - Fallo no detectado:
    - Cuando el importe del recibo sea 100€ el programa no lo muestra pero debería mostrarlo (aunque la especificación es un poco ambigua)
- Se debe probar en los límites de cada clase de equivalencia. Aquí se probaría con 100 y se detectaría el fallo

Pruebas de sistema: diferentes objetivos

- Funcionales: se realizan las funciones especificadas
- Pruebas relacionadas con el rendimiento del sistema:
  - Rendimiento (tiempos de respuesta adecuados)
  - Volumen (funcionamiento con grandes volúmenes de datos)
  - Sobrecarga (funcionamiento en el umbral límite de los recursos)
- Disponibilidad de datos (cuando se produce una recuperación ante fallos)
- Facilidad de uso (usabilidad y accesibilidad)
- Operación e instalación (operaciones de rearranque, actualización de software)
- Entorno (interacciones con otros sistemas) y comunicaciones
- Seguridad (control de acceso e intrusiones, ej. Inyección código SQL, XSS). Aunque esté al final de la lista no es el menos importante!!!

Herramientas

- JUnit (http://www.junit.org/)
  - Las pruebas se escriben en Java
  - Clases derivadas de TestCase
  - Comparación salidas (assertXXX)
  - Automatización prueba de regresión
  - Pruebas autodocumentadas (en el código)
  - Independiente e Integrado en Eclipse
- NUnit (entorno .NET)
  - Equivalente a Junit, diversos lenguajes
  - Menor nivel de integración

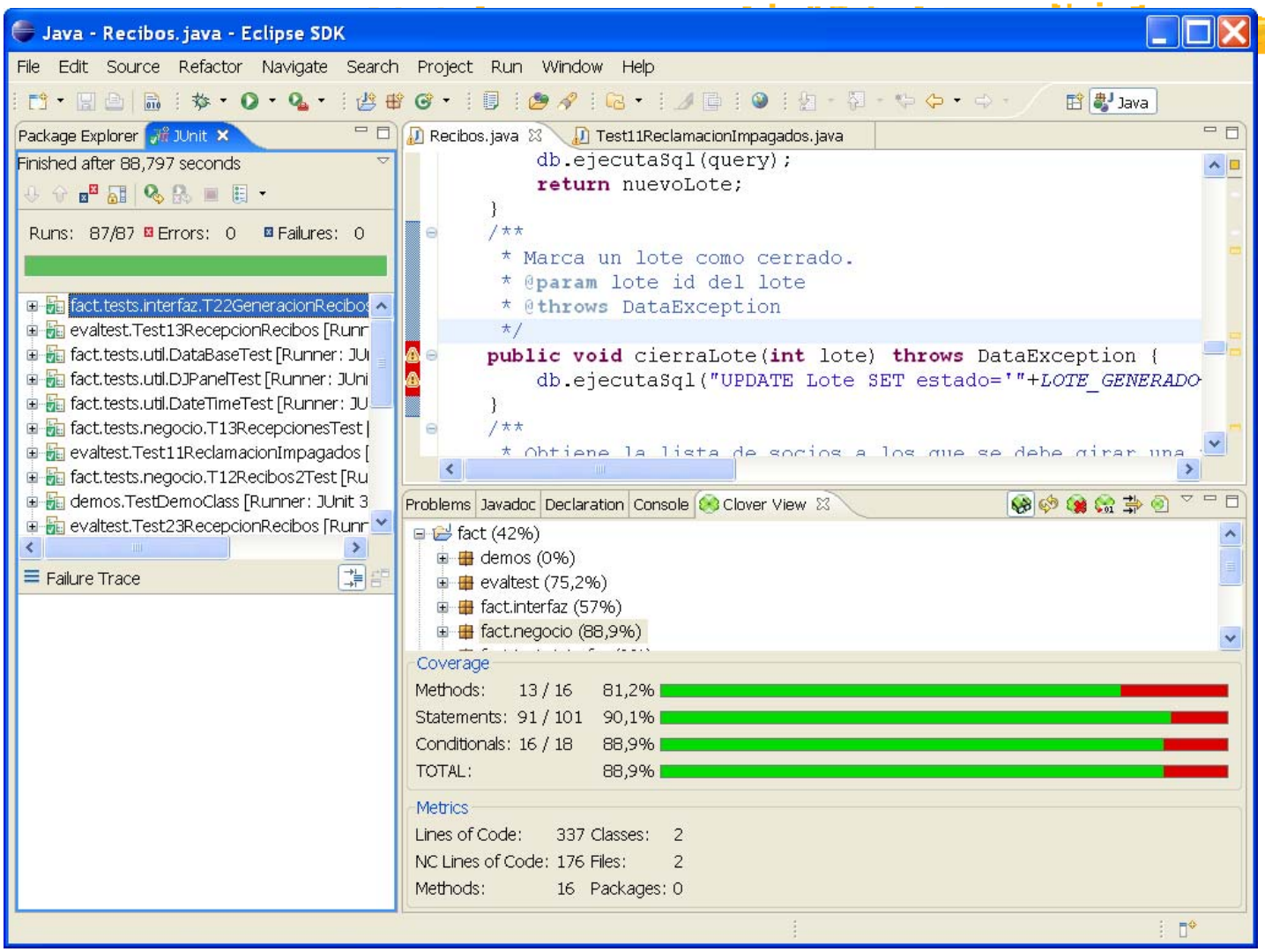
Herramientas

- Extensiones Junit (Xunit)
  - Interfaz de usuario swing: JFCUnit (http://jfcunit.sourceforge.net/)
    - Identificar e interactuar con objetos gráficos
  - Interfaz Web: HttpUnit (http://httpunit.sourceforge.net/) y JWebUnit (http://jwebunit.sourceforge.net/)
    - Identificar e interactuar con objetos de la página
    - Manejo de request/response
  - Datos: DBUnit (http://dbunit.sourceforge.net/)
    - Conexión con la base de datos
    - Diversos formatos para cargar datos (XML, base de datos, CSV, etc.)
    - Métodos assertXXX para comparar las estructuras de tablas
  - Cargas masivas de datos: DBMonster (http://dbmonster.kernelpanic.pl/)

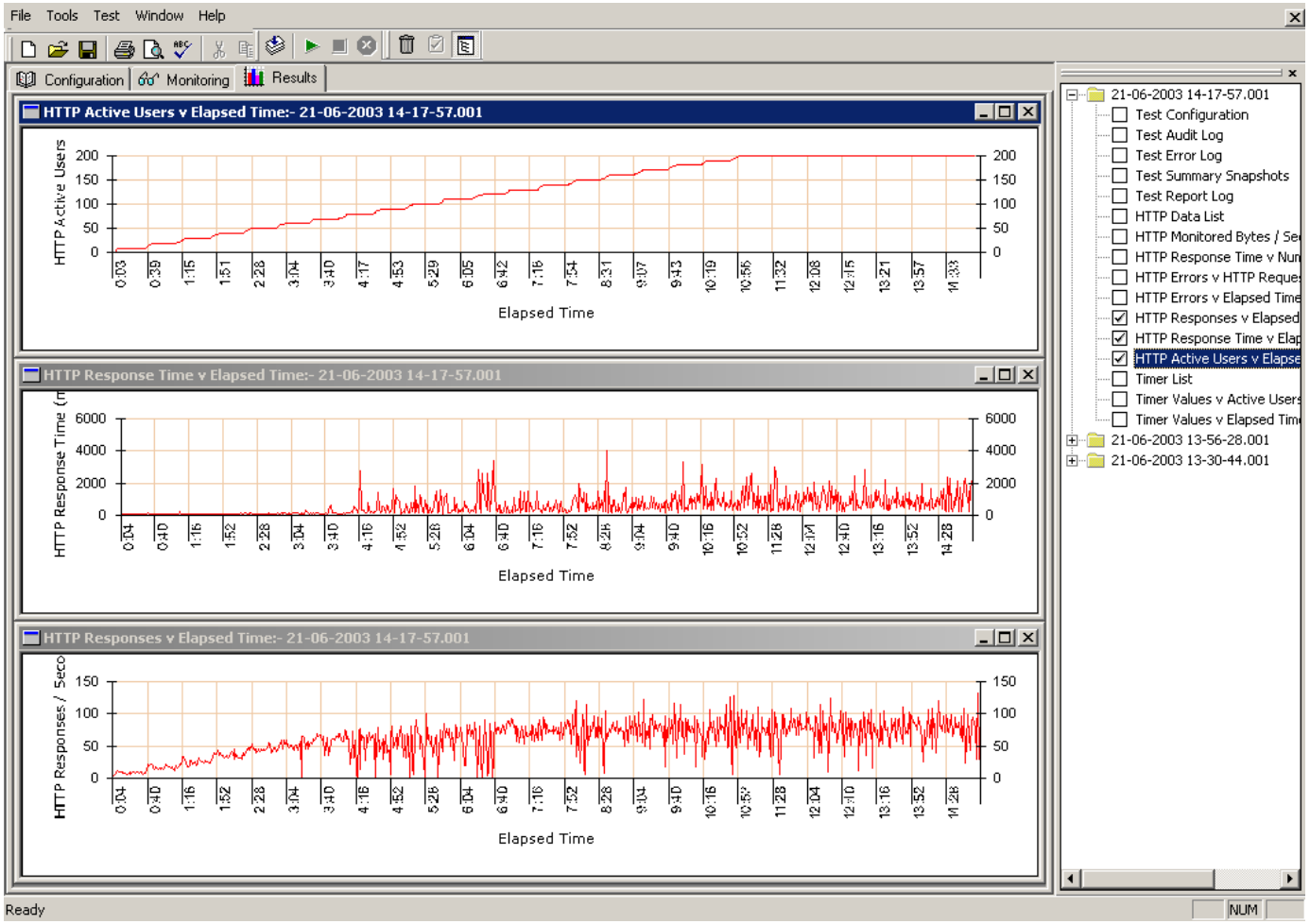
Herramientas

- Análisis de cobertura de código: Clover (http://www.cenqua.com)
  - Integración con Eclipse
  - Cobertura de líneas y decisiones. Estadísticas
- Carga y Stress: OpenSTA (http://www.opensta.org/).
  - Registro de una sesión interactiva
  - Lenguaje de programación, modificar script
  - Ejecutar, simulando usuarios y controlando la carga
  - Estadísticas
- Seguimiento de defectos: Bugzilla (http://www.bugzilla.org/).
- Muchas otras comerciales.
- Links de interés:
  - “Software testing tools FAQ” (http://www.testingfaqs.org/)
  - “Software QA Testing and Test Tool Resources” (http://www.aptest.com/resources.html).

Herramientas.
Ejemplo Cobertura Clover



Herramientas.
Ejemplo salida OpenSTA



Herramientas. Automatizar o no automatizar?

- Las herramientas no son la panacea, pero ayudan
- Siempre se requiere un buen diseño
- La automatización es complementaria, facilita la repetibilidad
- Algunos problemas, p.e. herramientas capture/playback, difícil mantenimiento
- Casos típicos: pruebas de humo y regresión, pruebas de configuraciones, aplicaciones web, extreme programming
- Prever el coste y recursos del proyecto de automatización
- Subconjunto mínimo: Junit/DBUnit, carga/stress, seguimiento defectos

El Plan de Pruebas

- IEEE Standard for Software Test Documentation. IEEE Std 829-1983.
  - especificación del diseño de las pruebas,
  - casos de prueba,
  - procedimientos de prueba
  - informes
  - registros de prueba.
- Problema: Complejo, mucha documentación

a) Test plan identifier;
b) Introduction;
c) Test items;
d) Features to be tested;
e) Features not to be tested;
f) Approach;
g) Item pass/fail criteria;
h) Suspension criteria and resumption requirements;
i) Test deliverables;
j) Testing tasks;
k) Environmental needs;
l) Responsibilities;
m) Staffing and training needs;
n) Schedule;
o) Risks and contingencies;
p) Approvals.

El Plan de Pruebas. Enfoque minimalista

- Hoja de cálculo para seguimiento de pruebas
  - Estructura jerárquica hasta llegar a los casos de prueba individuales
  - Identificación, descripción
  - estado (indicando si se ha superado la prueba o si se ha encontrado un fallo)
  - información adicional (fecha, quién la realizó y comentarios)
- La misma u otra con similar estructura como especificación de los casos:
  - Entradas
  - salidas esperadas.

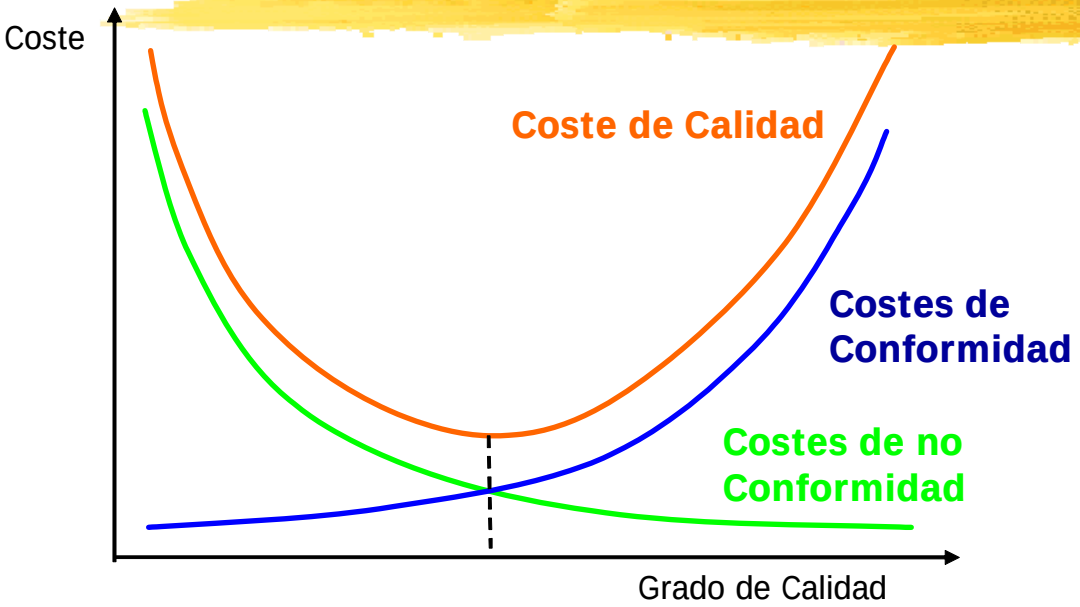
Personal

- Habilidades específicas del ingeniero de pruebas
  - Asumir el reto de encontrar fallos, más allá de “comprobar que funciona”
  - Concentrar la atención en los detalles
  - Curiosidad e intuición: explorar situaciones donde se pueden encontrar errores.
  - Capacidad de actuar bajo presión
    - Actuar cuando la presión del plazo es mayor
    - Conflicto de intereses con el equipo de desarrollo
  - Habilidades comunicativas
    - Comunicar de forma clara y precisa los defectos encontrados
    - Discutirlos con los equipos de desarrollo
  - Conocimiento funcional (depende de la tarea)
- Organización de equipos de pruebas. Máxima independencia
  - Asignación temporal o rotación entre equipo de desarrollo y prueba
  - Formación de equipos especializados
    - Internos (mejor conocimiento funcional)
    - Externos (más especializados, menor conocimiento funcional)
  - Mixtas

Planificación

- Principio básico:
  - Asignar recursos suficientes tanto a la preparación de pruebas como a su ejecución.
- Relación de tamaños equipos de desarrollo respecto de pruebas
  - Cifras típicas: desde 4 a 1 hasta 3 a 2
- Ventana temporal para las pruebas
  - Siempre considerar varios ciclos
  - Mínimo dos
  - Mejor más (el segundo ciclo puede incluir más pruebas para detectar más errores)

Costes

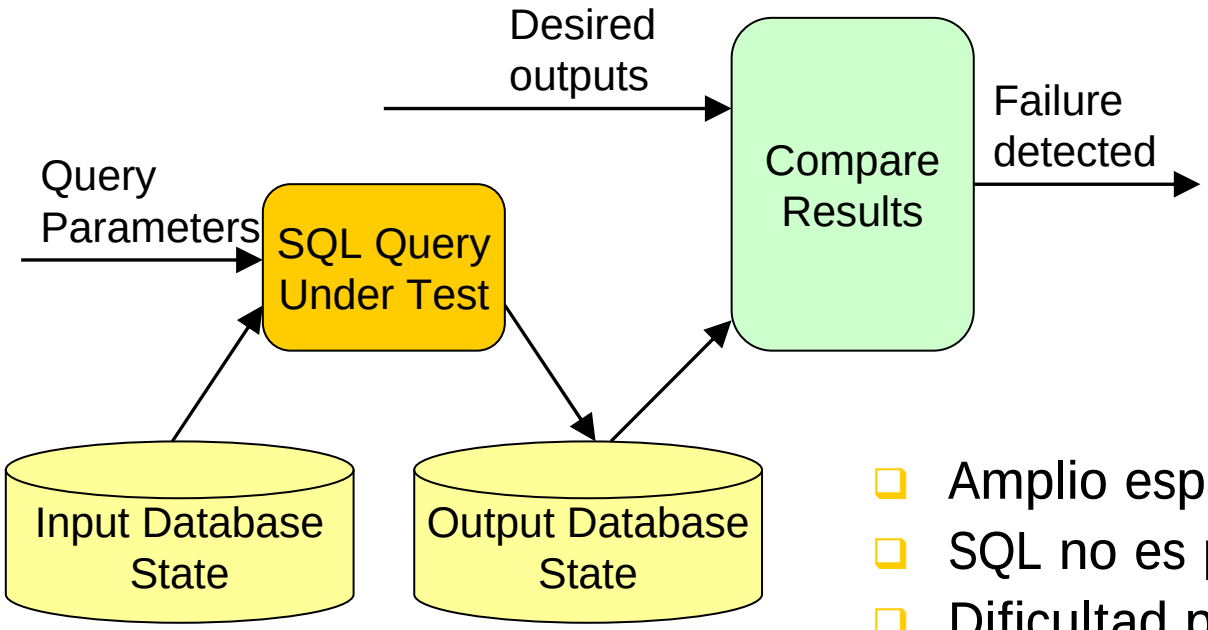


- Invertir en pruebas (y en general en calidad) es rentable
  - (Krasner)
    - En manufactura: 5-25%
    - En software: 10-70%
  - En Empresas de Automoción:
    - 4% Excelencia
    - 4-8% Buenos
    - >10% No deseables

- $C = C_{conformidad} + C_{noconformidad}$
  - $C_{conformidad} = C_{prevención} + C_{evaluación}$
  - $C_{noconformidad} = C_{internos} + C_{externos}$

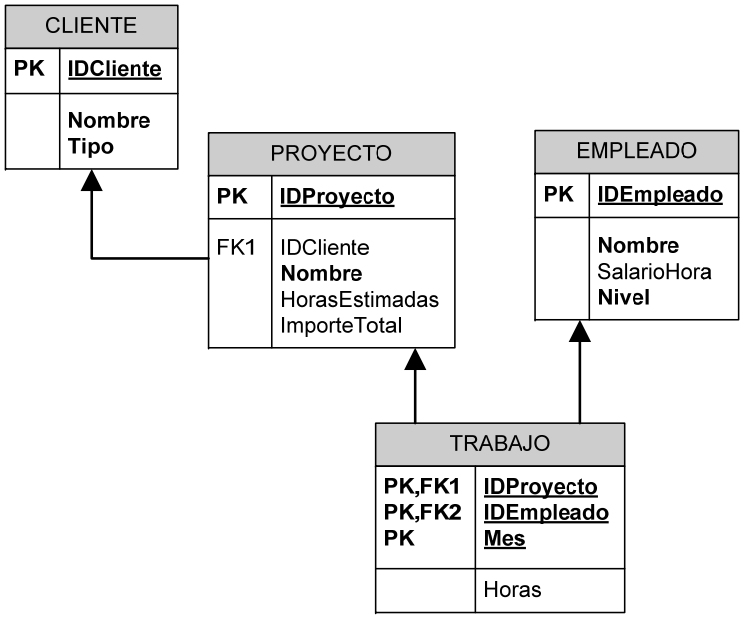
- Costes de inactividad en cliente (N. Donfrio, 2002):
  - Cadena Suministro: 300K$/hora
  - ERP/e-comm: 480K$/hora

Pruebas sobre bases de datos



- Amplio espacio de entrada y salida
- SQL no es procedural
- Dificultad para determinar salida deseada
- Qué es un caso de prueba?
- Qué criterios utilizamos para diseñar los casos?

Pruebas sobre bases de datos

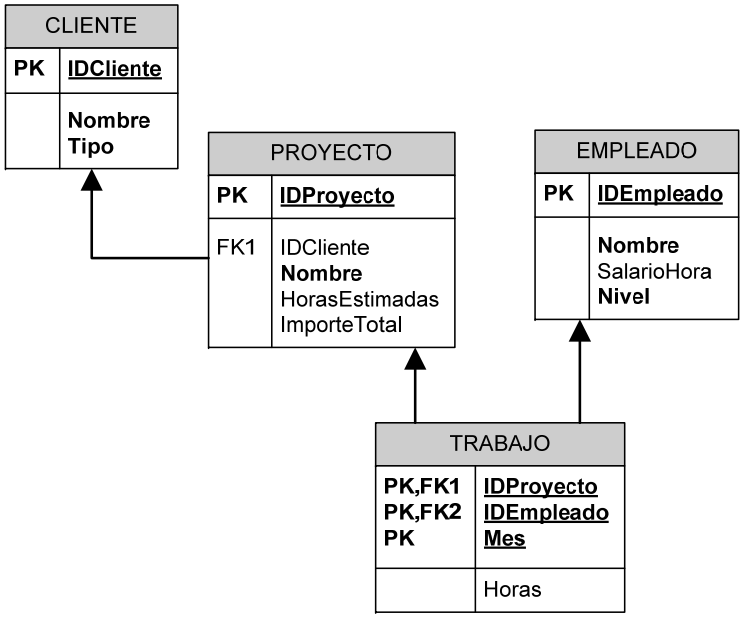


Listar todos los nombres de proyectos, mes y de empleados que han trabajado en el mismo, en los que el empleado ha trabajado más de 80 horas en el mes, o con un salario no mayor que 20 o nivel distinto de B ordenado por nombre de proyecto, de empleado y mes

```
SELECT P.nombre, T.mes, E.nombre
    FROM proyecto P
    INNER JOIN trabajo T ON P.idproyecto = T.idproyecto
    INNER JOIN empleado E ON T.idempleado = e.idempleado
    WHERE  T.horas>80 OR E.salariohora<=20 OR E.nivel<>'B'
    ORDER BY p.nombre,E.nombre,T.mes
```

- Joins (maestro con/sin detalle)
- Condiciones del where
- Ordenación (filas con mismo valor)

Pruebas sobre bases de datos



- Valores null
- Criterios de agrupación
- Condiciones del having
- Funciones agregado

Listar código de proyecto, de empleado, importe total de proyecto y suma de horas invertidas por cada empleado para aquellos proyectos cuyo importe total no llega a 2000, en los que el empleado haya invertido menos de 20 horas, ordenado por proyecto y empleado

```
SELECT P.idproyecto, E.idempleado, P.importetotal,
    SUM(T.horas)
     FROM proyecto P
     INNER JOIN trabajo T ON P.idproyecto = T.idproyecto
     INNER JOIN empleado E ON t.idempleado = E.idempleado
     GROUP BY P.idproyecto , E.idempleado, P.importetotal
     HAVING SUM(T.horas)<20  AND (P.importetotal<2000 OR
    P.importetotal IS NULL)
     ORDER BY P.idproyecto, E.idempleado
```


RePRIS - Red para la promocion y mejora de las Pruebas en Ingenieria del Software - Windows Internet Explorer
http://in2test.lsi.uniovi.es/repris/
RePRIS - Red para la promocion y mejora de las Prue...
RePRIS
Red para la promoción y mejora de las Pruebas en Ingeniería del Software
Objetivos | Participantes | Actividades | Enlaces | Taller PRIS 2008
Objetivos de la Red
La Red para la promoción y mejora de las Pruebas en Ingeniería del Software (RePRIS) tiene como objetivo coordinar los esfuerzos y conocimientos en pruebas del software entre grupos investigadores y el sector empresarial para difundir el conocimiento y promover la mejora de la práctica de la prueba del software en la industria, de las líneas de investigación y de la formación tanto a nivel universitario como no universitario.
Dar a conocer y promover la investigación en pruebas de software.
Promover el uso de buenas prácticas de pruebas y de las herramientas adecuadas en la industria del software.
Analizar y promover la docencia en las pruebas del software.
Incrementar el nivel competitivo de la investigación.
Contribuir a mejorar la calidad del software desarrollado.
Acción Especal (TIN2005-24792-E, TIN2007-30391-E) financiada por el Plan Nacional de I+D+I, Ministerio de Educación y Ciencia, cofinanciado con Fondos FEDER.
Intranet local
100%

Bibliografía Básica

- Metodología de planificación y desarrollo de sistemas de información MÉTRICA, Versión 3 – Interfaces, Técnicas y Prácticas; 280-285 (estrategias de prueba)
- G.J. Myers, The Art of Software Testing, 1979, 2004
- C. Kaner, J. Falk, H.Q. Nguyen, Testing Computer Software, 1999
- R.S. Pressman, Ingeniería del software, Un enfoque práctico; Cap. 17 (técnicas), 18 (estrategias), 23 (orientación a objetos)
- J. Tuya, I. Ramos, J. Dolado, Técnicas cuantitativas para la gestión en la ingeniería del software, 2007; Cap. 3: Técnicas y prácticas en las pruebas del software
- M. Piattini, J. Garzas Eds., Fábricas de Software: experiencias, tecnologías y organización, 2007; Cap. 7: Las pruebas del Software (J. Tuya)
- G. Tassey, The economic impact of inadequate infrastructure for software testing, NIST Report – May 2002