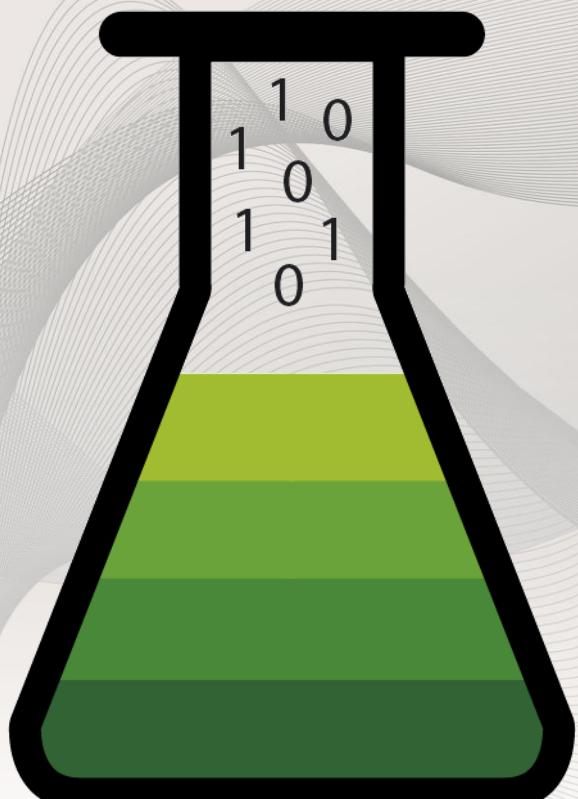


0 1 1
1 0 1 0
0 1 1 0
1 0 1 1



Curso básico sobre testeo de Software

Dr. Julián Alberto García García

Julian.garcia@iwt2.org

Dr. Francisco José Domínguez Mayo

fjdominguez@us.es



Yes,
we **test!**

Índice

- ▶ **1. Conceptos fundamentales del testeo de software**
- **2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- **3. Técnicas de testing: métodos para generar un caso de prueba**
- **4. Herramientas de testing: el soporte para automatizar actividades de testing**
- **5. Métricas de testing: midiendo la calidad del software**
- **6. Procesos de testing: cómo integrar el testeo en la metodología**

INTRODUCCIÓN



- ✓ Sistemas software en la actualidad:
 - Características: mayor tamaño, importante nivel de seguridad y bienestar, y complejidad.
 - Aspectos clave: calidad del producto, precio competitivo, reducción de costes, etc.

Cada día que pasa, el software es pieza fundamental en el funcionamiento de maquinaria, instalaciones, equipamientos, sistemas expertos, financieros y biomédicos, etc.



¿POR QUÉ SON NECESARIAS LAS PRUEBAS?**ARIANE 5**

Vuelo 501 (04/06/1996), primera prueba de vuelo del sistema de lanzamiento del Ariane 5. **No fue un éxito** ya que 37 segundos después del lanzamiento, la **lanzadera explotó** debido a un **mal funcionamiento del software de control**.

MOTIVO: fallo software, el módulo de control no se había probado lo suficiente.

¿POR QUÉ SON NECESARIAS LAS PRUEBAS?

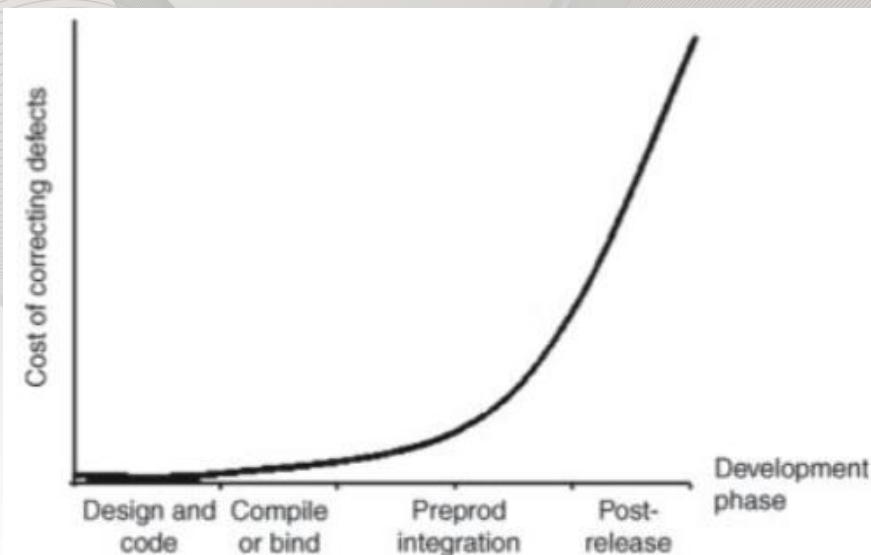
CARACTERÍSTICAS DE LAS PRUEBAS

- ✓ Más importancia y protagonismo día a día.
- ✓ Mejoran la calidad del software.
- ✓ Mejoran la satisfacción de los requisitos.
- ✓ Ahorra tiempo y recursos durante el desarrollo.

- ✓ **Su objetivo:** localizar y subsanar el mayor número de deficiencias lo antes posible.



COSTE DE LOS DEFECTOS



- ✓ Coste de eliminar un defecto se **incrementa con el tiempo** de permanencia de dicho defecto.

- ✓ La detección de errores en **etapas tempranas** permite su corrección a menor coste.

Satisfacción
del cliente y
usuarios



CONCEPTOS BÁSICOS

Test (prueba)

- (1) noun: activity by which a *test item* is evaluated and the outcomes reported.
- (2) verb (testing): set of activities conducted to facilitate discovery and/or evaluation of properties of one or more *test items*.

[ISO/IEC/IEEE 29119](#)
[Software Testing](#)

Test item (objeto de prueba)

system, software item, or work product (e.g. requirements document, design specification) that is an object of testing.

Plan de pruebas

Establece una planificación formal de las pruebas en que se define la secuencia de las pruebas planificadas, quién debe realizar las y cuándo.

Tener en cuenta prioridades.
Disponibilidad de recursos.
Infraestructura de prueba, etc.

¿Funciona el teléfono?

Valores de prueba	Acciones	Resultado esperado
123-45-67-89	1. Descolgar auricular. 2. Marcar número de Pepito. 3. Esperar contestación.	(Pepito): “Digameee”.

¿Funciona la suma?

Valores de prueba	Acciones	Resultado esperado
???	Suma(a, b)	???

```
public int suma(int a, int b)
{
    return a + b;
}
```

Técnicas

Valores de prueba	Acciones	Resultado esperado
0, 0	Suma(a, b)	0
0, b = no 0	Suma(a, b)	b
3, 4	Suma(a, b)	7
-2, -8	Suma(a, b)	-10
-3, 6	Suma(a, b)	3
Integer.MAX_VALUE, 6	Suma(a, b)	-2147483643

CONCEPTOS

¿Qué aspectos debe contemplar una prueba bien definida?

PROPIEDAD	SIGNIFICADO
Identificador	Código único de la prueba
Valores de entrada	Descripción de los datos de entrada de un objeto de prueba
Resultados esperados	Datos de salida que se espera se produzcan
Precondiciones	Situación previa a la ejecución de la prueba o características de un objeto de prueba antes de ejecutar un caso de prueba
Postcondiciones	Características un objeto de prueba tras la ejecución de la prueba
Dependencias	Relación u orden de dependencia entre casos de pruebas
Acciones	Pasos a llevar a cabo para ejecutar la prueba
Requisito vinculado	Relación de requisitos que se pretenden validar con la ejecución de la prueba

Índice

- 1. Conceptos fundamentales del testeo de software**
- 2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- 3. Técnicas de testing: métodos para generar un caso de prueba**
- 4. Herramientas de testing: el soporte para automatizar actividades de testing**
- 5. Métricas de testing: midiendo la calidad del software**
- 6. Procesos de testing: cómo integrar el testeo en la metodología**

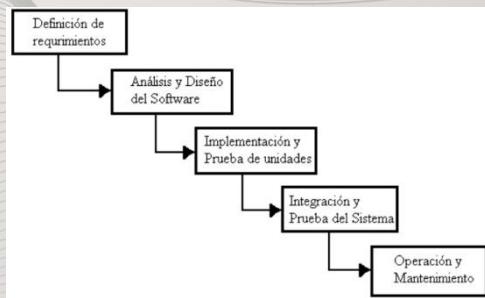
LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE

- **CICLO DE VIDA SOFTWARE:** marco de referencia que define el **enfoque general del desarrollo** software, indicando los **procesos y actividades** a realizar desde la definición del producto software hasta su finalización de uso; así como los **entregables** que se **van a generar y entregar** al cliente (ISO 12207).

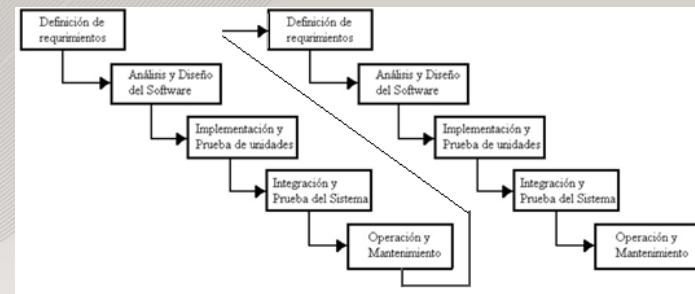
LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE

- **CICLO DE VIDA SOFTWARE:** marco de referencia que define el **enfoque general del desarrollo software**, indicando los **procesos y actividades** a realizar desde la definición del producto software hasta su finalización de uso; así como los **entregables** que se **van a generar y entregar** al cliente (ISO 12207).

Modelo en cascada (clásico)



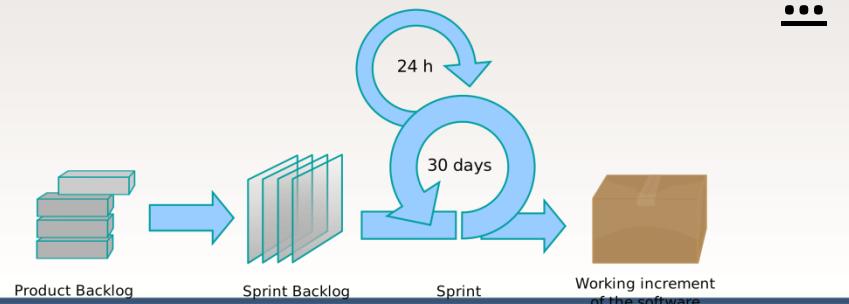
Modelo iterativo (RUP, XP, etc.)



Modelo evolutivo



Modelos ágiles (e.g. SCRUM)



LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE

Pruebas a través del ciclo de vida: Modelo-V

- El modelo-v general es el modelo de desarrollo software **más utilizado**.
 - Desarrollo y pruebas son **dos ramas iguales**. Cada nivel de desarrollo tiene su correspondiente nivel de pruebas.
 - Las pruebas (rama derecha) son diseñadas en paralelo al desarrollo (rama izquierda).
 - Las actividades del proceso de pruebas tiene lugar a través del ciclo de vida software completo
-
- ```

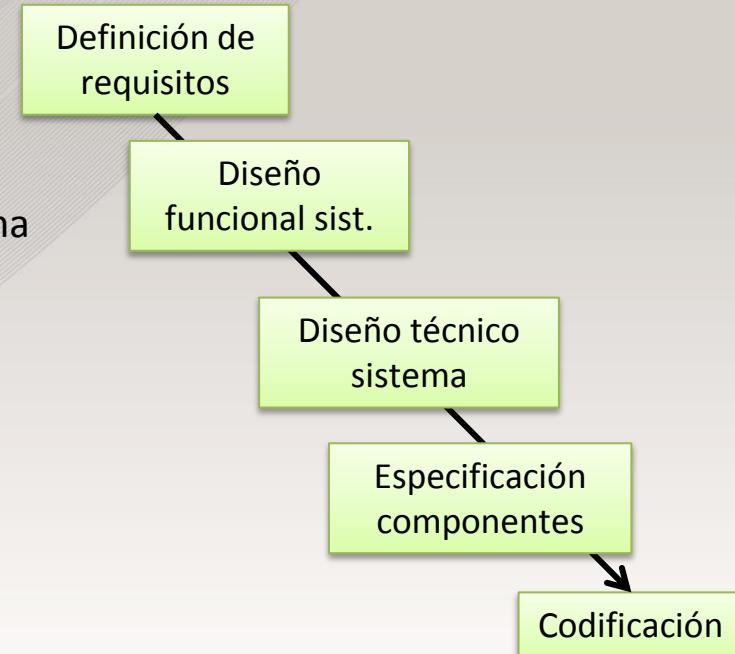
graph TD
 A[Definición de requisitos] --> B[Diseño funcional sist.]
 B --> C[Diseño técnico sistema]
 C --> D[Especificación componentes]
 D --> E[Codificación]
 E --> F[Pruebas de componentes]
 F --> G[Pruebas de integración]
 G --> H[Pruebas de sistema]
 H --> I[Pruebas de aceptación]
 I --> J[Pruebas de aceptación]

```

### Pruebas a través del ciclo de vida: Modelo-V

- Rama de desarrollo software

- Definición de requisitos
  - Documentos de especificación
- Diseño funcional del sistema
  - Diseño del flujo funcional del programa
- Diseño técnico del sistema
  - Definición de arquitectura/interfaces
- Especificación de los componentes
  - Estructura de los componentes
- Programación
  - Creación de código ejecutable



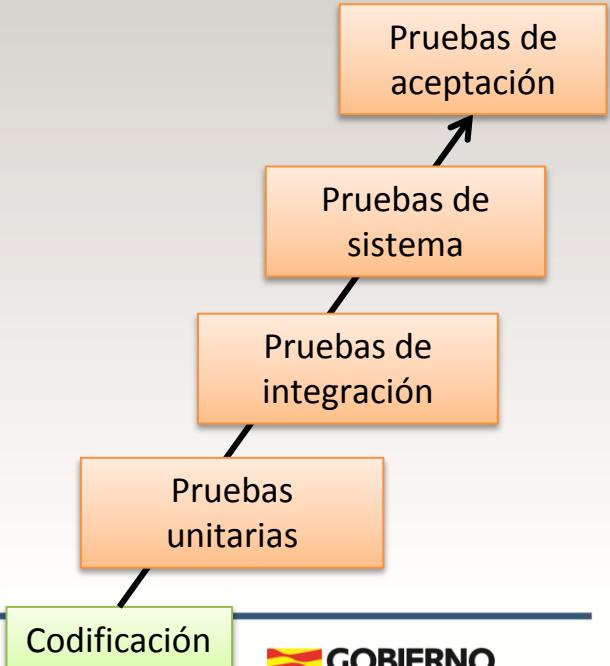
LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE



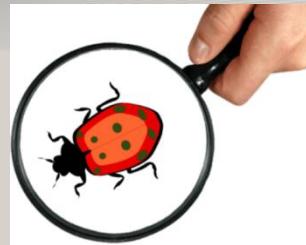
## Pruebas a través del ciclo de vida: Modelo-V

- Rama de pruebas software

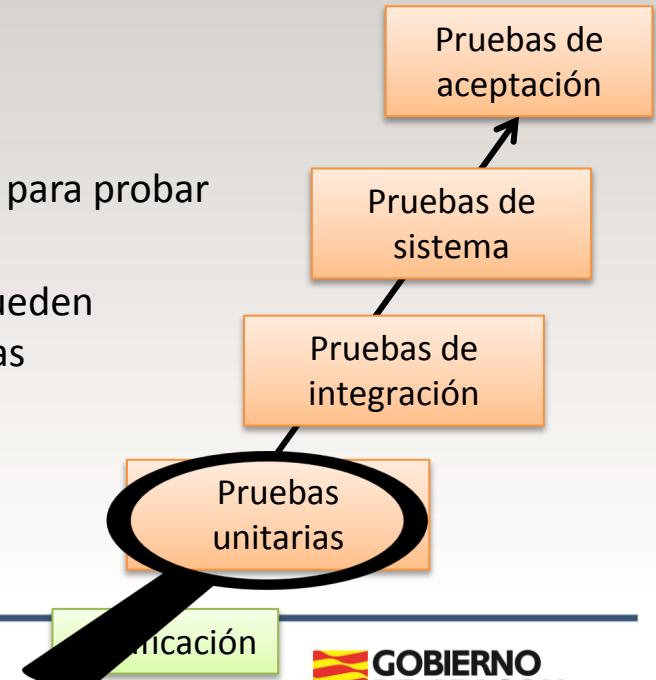
- Pruebas de aceptación
  - Pruebas formales de los requisitos del cliente
- Pruebas del sistema
  - Sistema integrado, especificaciones
- Pruebas de integración
  - Interfaces de componentes
- Pruebas unitarias
  - Funcionalidad del componente (clase, método, módulo, etc.)



## LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE

**PRUEBAS UNITARIAS:** Alcance

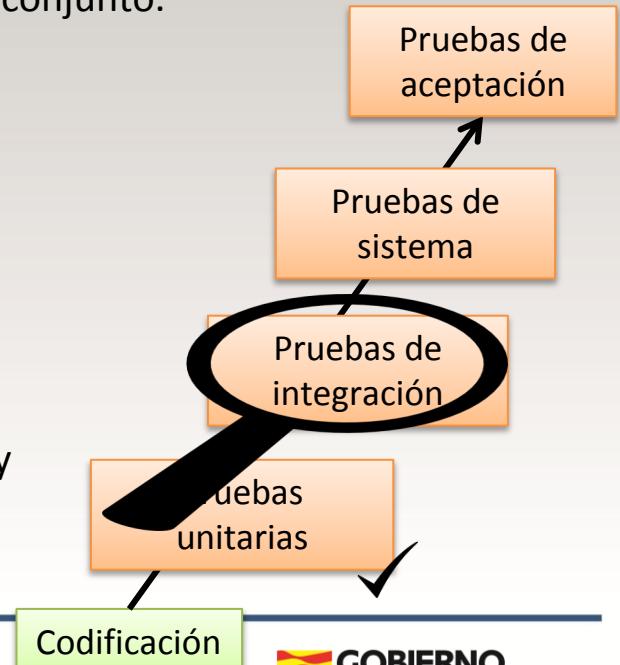
- ✓ Sólo se prueban **componentes individuales** (clases, funciones, módulos, etc.)
- ✓ Cada componente se prueba de **forma independiente**.
  - Descubre errores producidos por defectos internos
- ✓ Los casos de prueba podrán ser obtenidos a partir de:
  - Código fuente, modelo de datos, Diseño software.
- ✓ Se pueden realizar pruebas de **caja blanca** y de **caja negra** para probar los módulos completamente.
  - Las pruebas de caja negra (los casos de prueba) se pueden especificar antes de que programar el modo, no así las pruebas de caja blanca.



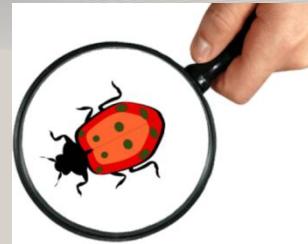
### LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE

#### PRUEBAS DE INTEGRACIÓN: Alcance

- ✓ Las pruebas de integración comprueban la interacción mutua entre componentes (subsistemas) software entre sí. Se asumen que los componentes ya han sido aprobados.
- ✓ Tendencia a intentar una integración no incremental:
  - Combinar todos los módulos y probar el sistema en su conjunto.
  - Resultado previsible: CAOS !!!
- ✓ Recomendación: aplicar integración incremental:
  - El software se prueba en pequeñas porciones.
  - En la detección y resolución de errores es más: Fácil, controlable y gestionable.
- ✓ Los casos de prueba podrán ser obtenidos a partir de:
  - Especificación de interfaces, diseño de la arquitectura y modelo datos

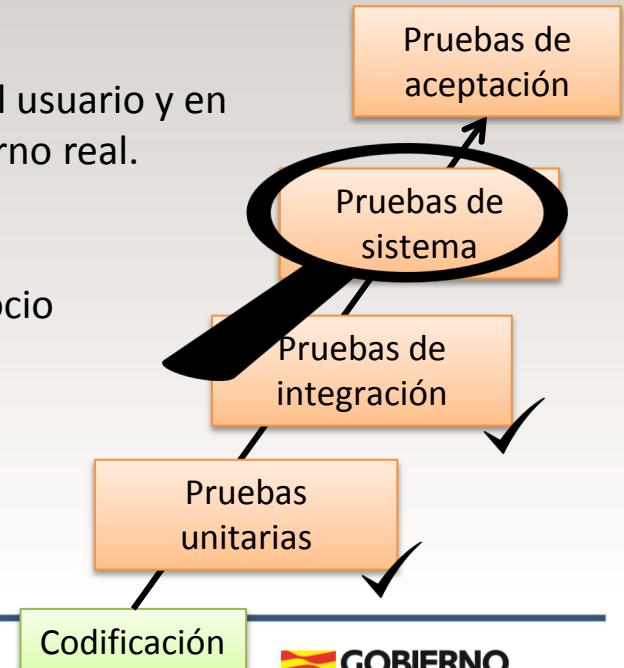


### LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE



#### PRUEBAS DE SISTEMA: Alcance

- ✓ Consiste en probar (lo más exhaustivamente posible) el software completo para verificar que:
  - Se cumplen los requisitos funcionales establecidos
  - Se cumplen aspectos no funcionales de calidad: usabilidad, eficiencia, portabilidad, seguridad, etc.
- ✓ La calidad software es observada desde el punto de vista del usuario y en un entorno de pruebas coincidente (en lo posible) con entorno real.
- ✓ Los casos de prueba podrán ser obtenidos a partir de:
  - Especificaciones funcionales, casos uso, procesos negocio
- ✓ Para la generación de casos de prueba se utilizan técnicas de caja negra

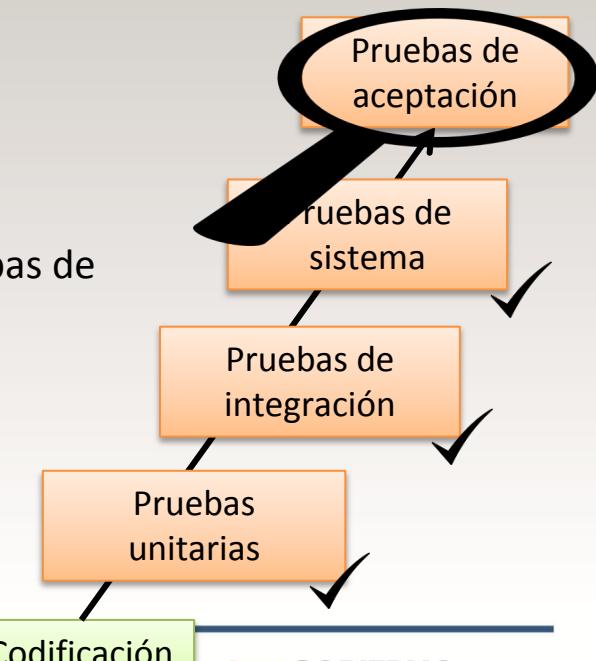


## LA CALIDAD EN CADA ETAPA DEL CICLO DE DESARROLLO SOFTWARE



### PRUEBAS DE ACEPTACIÓN: Alcance

- ✓ Son pruebas para aceptar formalmente el software. Son las pruebas de sistema del cliente/usuario.
- ✓ Es conveniente haber definido los criterios de aceptación verificables de manera previa y consensuada.
- ✓ Están enfocadas a demostrar que no se cumplen los requisitos, criterios de aceptación o el contrato.
- ✓ El usuario selecciona casos de prueba concretos para sus pruebas de aceptación, según las prioridades de su negocio.
- ✓ Las pruebas se realizarán en el entorno del cliente (real) y se utiliza técnicas de caja negra.



# Índice

- 1. Conceptos fundamentales del testeo de software**
- 2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- 3. Técnicas de testing: métodos para generar un caso de prueba**
- 4. Herramientas de testing: el soporte para automatizar actividades de testing**
- 5. Métricas de testing: midiendo la calidad del software**
- 6. Procesos de testing: cómo integrar el testeo en la metodología**



Realizadas sin ejecutar el código de la aplicación y su objetivo son realizar documentación y código fuente.  
Incluye revisiones y análisis estático.

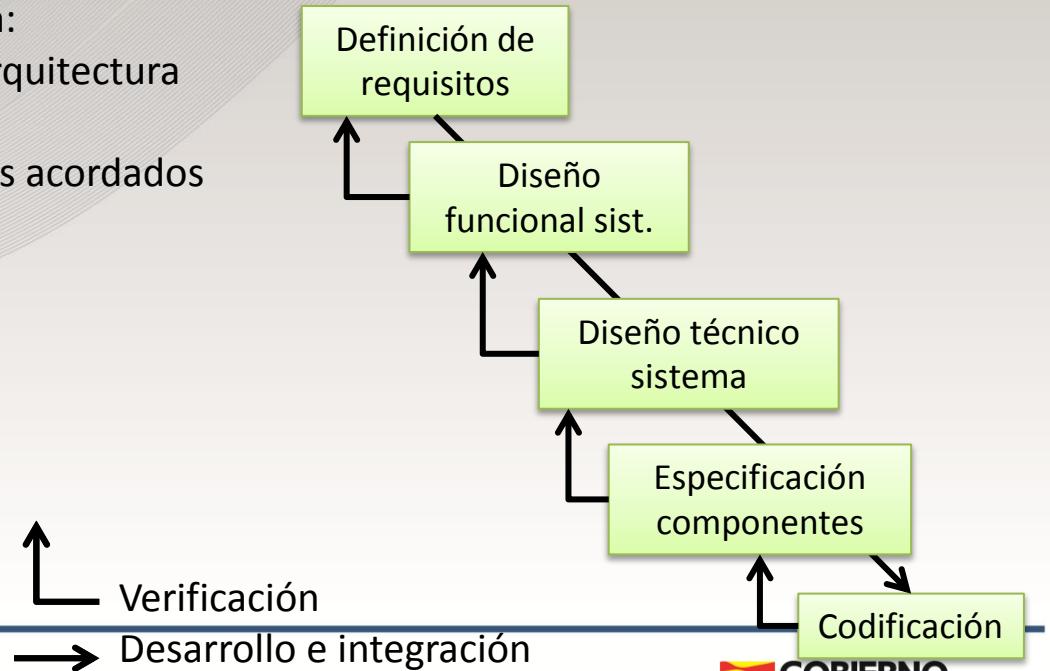


Requieren la ejecución del software por lo que es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.



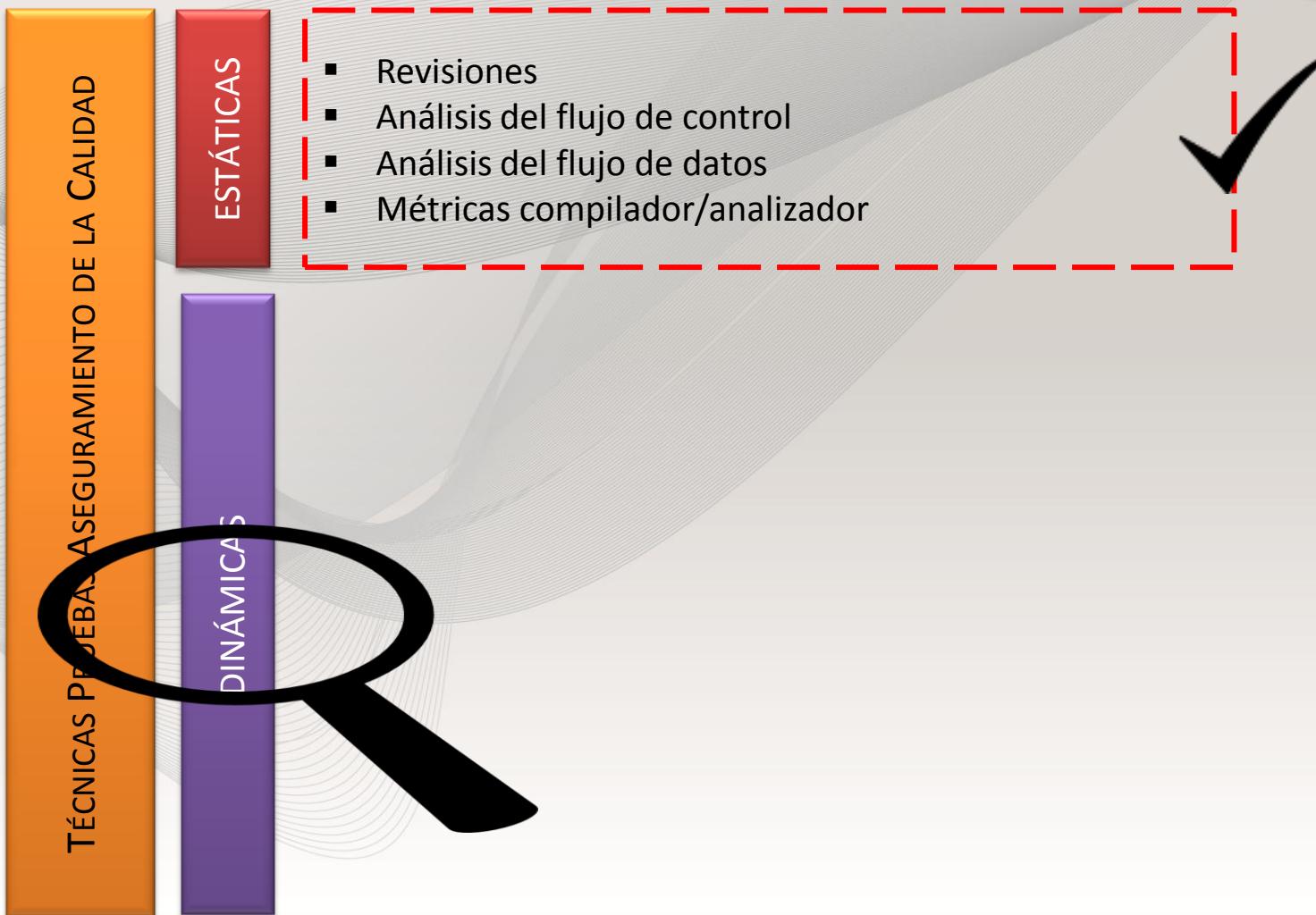
**TÉCNICAS ESTÁTICAS DE TESTING :-: REVISIONES**

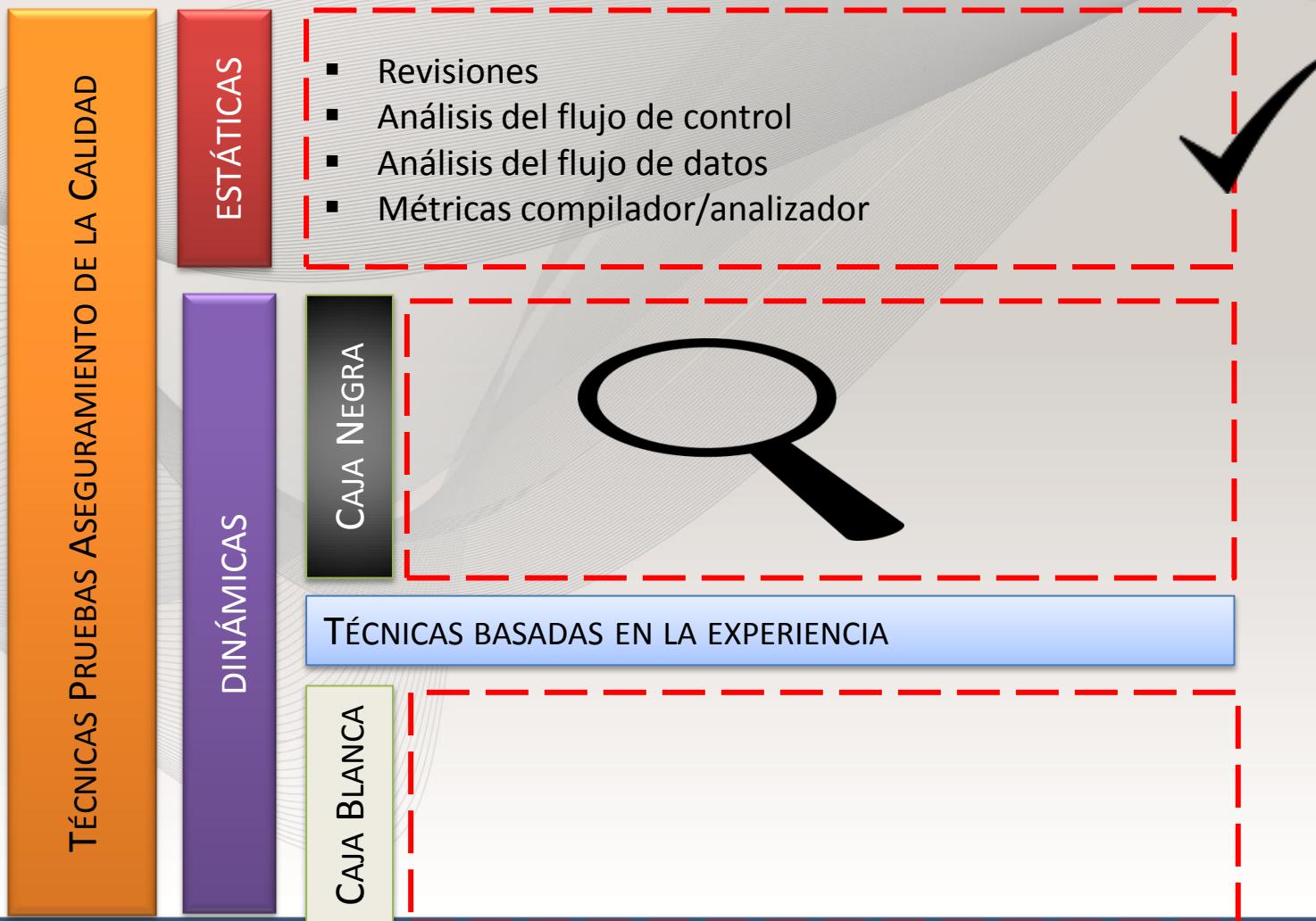
- ✓ OBJETIVO: mejorar la calidad del producto y reducir propagación de errores entre fases (modelo-v)
- ✓ La detección temprana de errores ahorra costes a posteriori.
- ✓ Defectos potencialmente detectables en:
  - Documentos de especificación y arquitectura
  - Especificaciones de interfaces
  - Desviaciones respecto a estándares acordados (e.g. Guías de programación)
- ✓ Tarea eminentemente manual.



**TÉCNICAS ESTÁTICAS DE TESTING :-: ANÁLISIS ESTÁTICO**

- ✓ OBJETIVO: Buscar errores en la especificación de objetos de prueba (e.g. Código fuente, script, etc.) sin ejecutar el objeto de prueba.
- ✓ Aspectos a detectar:
  - Reglas y estándares de programación
  - Diseño de un programa (análisis del flujo de control)
  - Uso de datos (análisis flujo de datos)
  - Complejidad de la estructura de un programa (métricas, e.g., nº ciclomático)
- ✓ Existen herramientas → compiladores, analizadores
  - Detectar lógica errónea (bucles potencialmente infinitos)
  - Detectar estructuras lógicas complejas, inconsistencias entre interfaces, etc.
  - Supone menor esfuerzo que una inspección





## CAJA NEGRA

## TÉCNICAS DINÁMICAS DE TESTING

- ✓ El tester observa el objeto de prueba como una caja negra.
  - La estructura interna del objeto de prueba es irrelevante o desconocida.
- ✓ Los casos de prueba se obtienen a partir del análisis de la especificación (funcional o no funcional) de un componente o sistema
  - Prueba de comportamiento entrada/salida
- ✓ ¡La utilidad es el foco de atención!
  - La técnica de caja negra también se denomina prueba funcional o prueba orientada a la especificación



## CAJA NEGRA

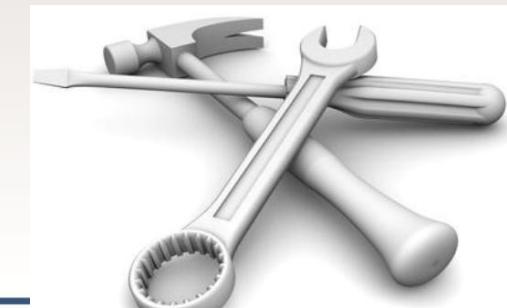
## TÉCNICAS DINÁMICAS DE TESTING

✓ **Métodos de caja negra:**

- Partición de equivalencias o clases de equivalencia.
- Análisis de valores límite.
- Pruebas de casos de uso.
- Tablas de decisión, de transición de estado, ...

## ✓ De manera general, la ejecución de casos de prueba debería ser ejecutados con una baja redundancia, pero con carácter completo.

- Probar lo **menos** posible, pero
- Probar **tanto** como sea necesario



✓ **Método: Clases de equivalencia (CE)**

- Consiste en dividir los valores de entrada en clases de datos para derivar casos de prueba.
- Se asume que el resultado de una prueba con un valor representativo de cada CE equivale a realizar la misma prueba con cualquier otro valor de la CE.
- Pasos para diseñar casos de prueba:
  1. Identificar clases de equivalencia.
  2. Identificar los casos de prueba. Minimizando nº casos de prueba, considerar tantas condiciones como sea posible. Pasos:
    - Asignar a cada CE un representante único.
    - Definir casos de prueba que cubran tantas CE válidas como sea posible. Repetir hasta que todas las CE estén cubiertas.
    - Definir un caso de prueba para cubrir una única CE no válida. Repetir hasta que todas estén cubiertas.

CAJA NEGRA

TÉCNICAS DINÁMICAS DE TESTING

**Ejemplo:** un programa requiere un número entero [0...100]. Posibles clases de equivalencia:

1. Identificar clases de equivalencia. Definir clases de datos válidos y no válidos.

- CE válida:  $0 \leq X \leq 100$
- 1ra CE no válida:  $X > 100$
- 2da CE no válida:  $X < 0$
- 3ra CE no válida:  $X = \text{decimal}$
- 4ta CE no válida:  $X = \text{alfanumérico}$

**Ejemplo:** un programa requiere un número entero [0...100]. Posibles clases de equivalencia:

1. Identificar clases de equivalencia. Definir clases de datos válidos y no válidos.

- CE válida:  $0 \leq X \leq 100$
- 1ra CE no válida:  $X > 100$
- 2da CE no válida:  $X < 0$
- 3ra CE no válida:  $X = \text{decimal}$
- 4ta CE no válida:  $X = \text{alfanumérico}$

2. Identificar casos de prueba (5 casos de prueba).

CAJA NEGRA

TÉCNICAS DINÁMICAS DE TESTING

✓ **Método: Análisis de Valores Límite**

- La experiencia demuestra que casos de prueba sobre condiciones límite infieren mejores resultado.
- Condiciones límite = márgenes de las clases de equivalencia (CE).
- Esta técnica complementada técnica de CE:
  1. Identificar las condiciones límite para los datos de entrada
  2. Generar tantos casos de prueba como sean necesarios para ejercitar las condiciones límites.

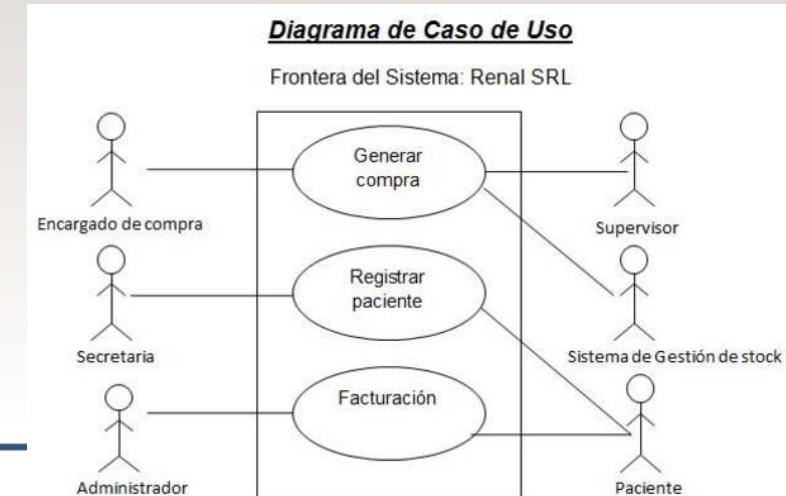
✓ **Método: Análisis de Valores Límite.** Ejemplo

- Construcción de una batería de pruebas para detectar posibles errores en la construcción de los identificadores de un hipotético lenguaje de programación. La regla que determina sus construcción sintáctica es:
  - *No debe tener mas de 15 ni menos de 5 caracteres*

| Condición               | Descripción de los casos de prueba                                                                                                                                                                                                                                        |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Entre 5 y 15 caracteres | <ul style="list-style-type: none"><li>• 1 caso con nº de caracteres identificador = 15</li><li>• 1 caso con nº de caracteres identificador = 5</li><li>• 1 caso con nº de caracteres identificador = 16</li><li>• 1 caso con nº de caracteres identificador = 4</li></ul> |

✓ Método: Pruebas de Casos de Uso

- Los casos de prueba son obtenidos desde los casos de uso.
  - Cada caso de uso puede ser utilizado como fuente para un caso de prueba, pero **cada paso alternativo** del caso de uso, se traduce en **una prueba distinta**.
- Cada caso de uso describe una cierta interacción usuario-sistema. Elementos: precondiciones, pasos del comportamiento del sistema, post condiciones.



✓ Método: Pruebas de Casos de Uso

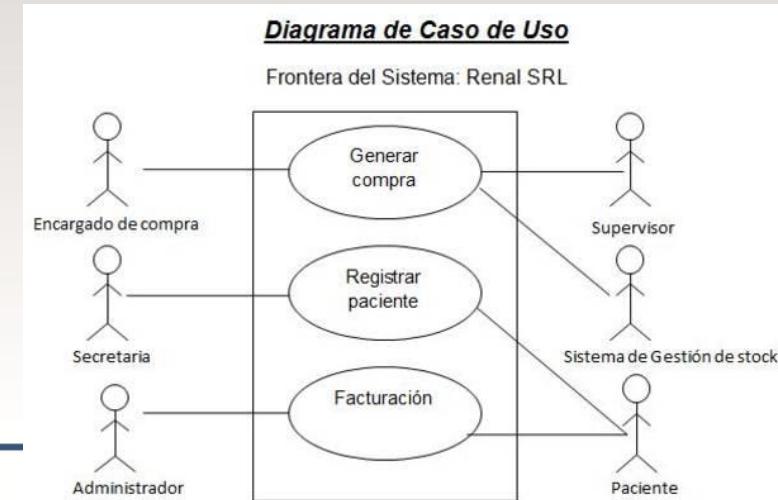
- Los casos de prueba son obtenidos desde los casos de uso.
  - Cada caso de uso puede ser utilizado como fuente para un caso de prueba, pero **cada paso alternativo** del caso de uso, se traduce en **una prueba distinta**.
- Cada caso de uso describe una cierta interacción usuario-sistema. Elementos: precondiciones, pasos del comportamiento del sistema, post condiciones.

✓ **Beneficios**

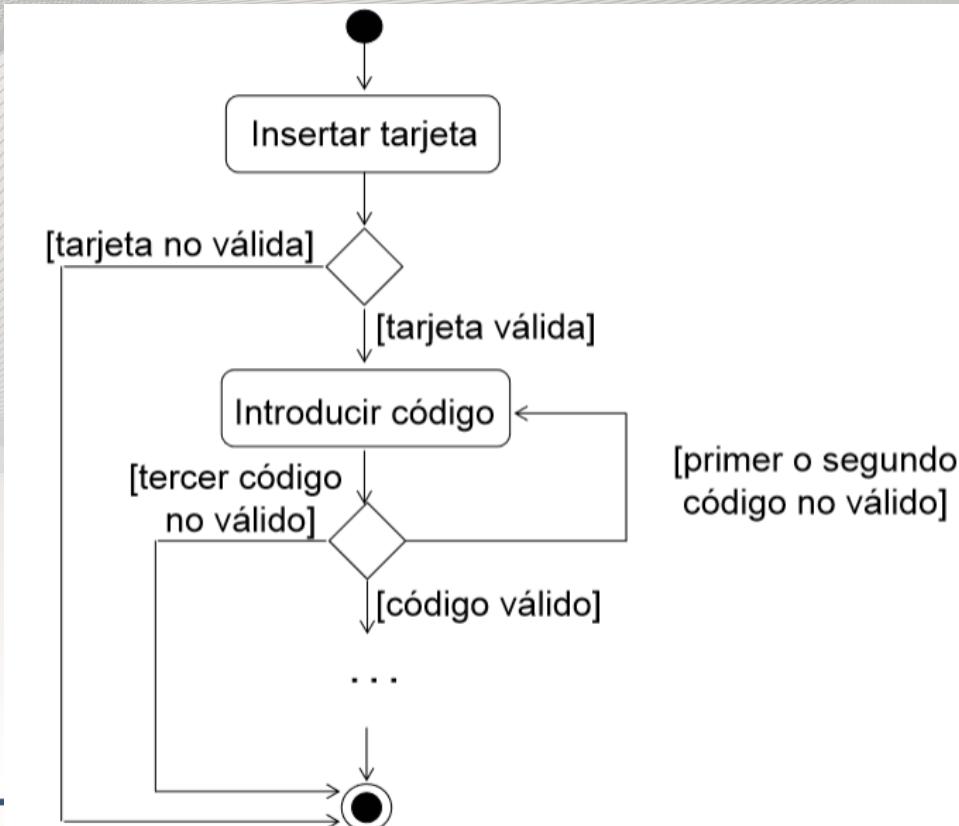
- Pruebas apropiadas para pruebas de aceptación y de sistema.
- Útil para diseñar pruebas con el cliente/usuario
- Puede ser combinadas con otras técnicas basadas en la especificación

✓ **Desventajas**

- No es posible obtener casos de prueba más allá de la información de los casos de uso.



- ✓ **Método: Pruebas de Casos de Uso.** Ejemplo del cajero automático



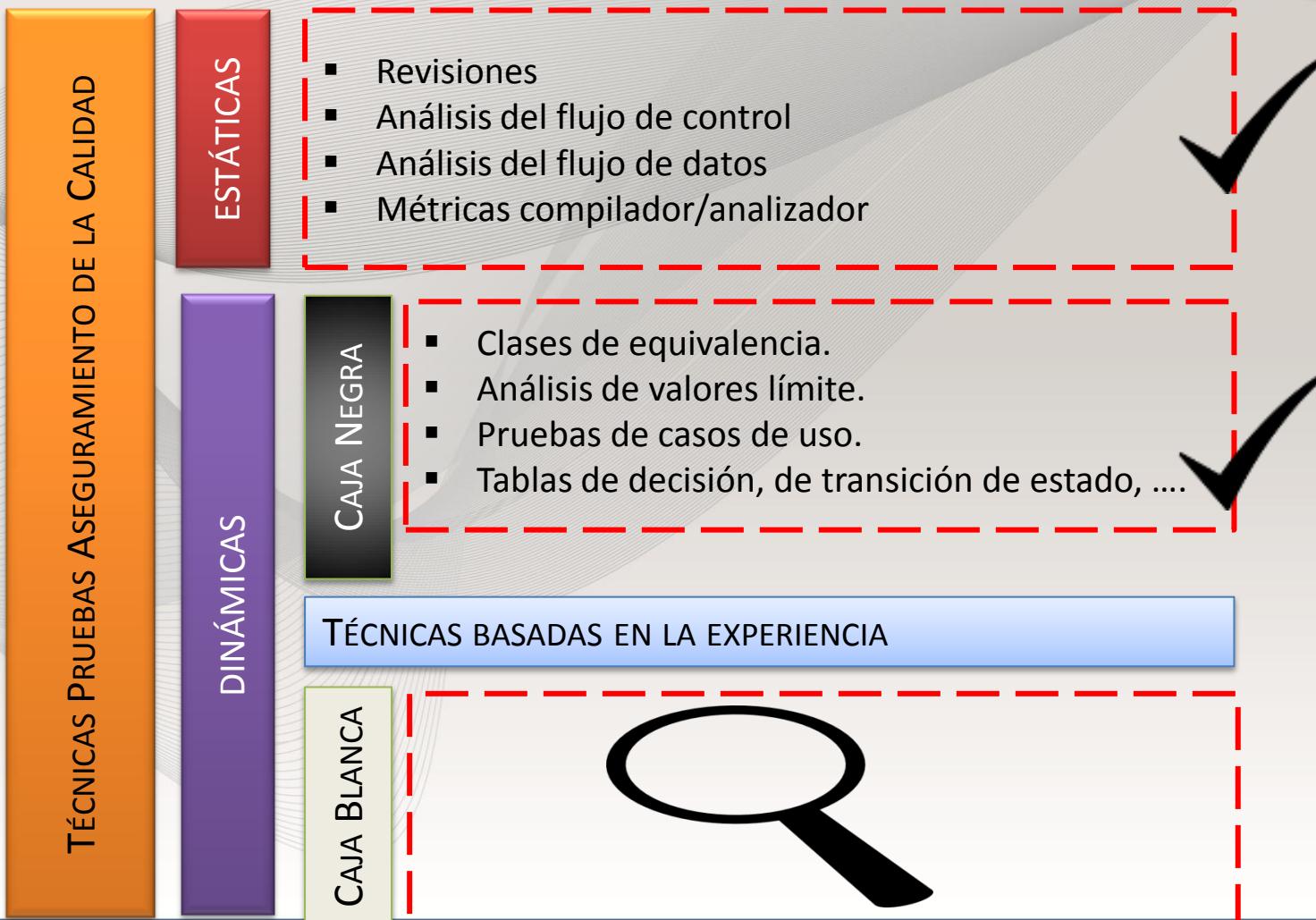
**Prueba 1.** Insertar tarjeta (no válida); fin

**Prueba 2.** Insertar tarjeta (válida); introduce código (no válido); fin

**Prueba 3.** Insertar tarjeta (válida); introduce código (válido);...; fin

**Prueba 4.** Insertar tarjeta (válida); introduce código (no válido); introduce código (no válido); fin

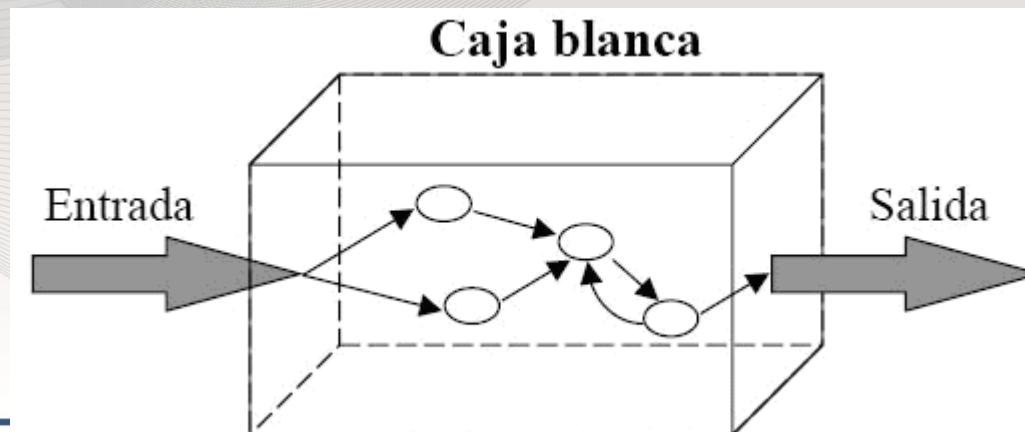
**Prueba 5. ...**



## CAJA BLANCA

## TÉCNICAS DINÁMICAS DE TESTING

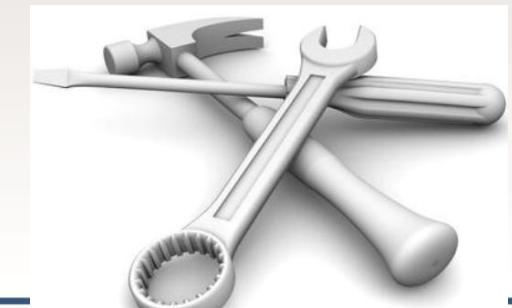
- ✓ El tester conoce la estructura interna del código, i.e., la jerarquía de componentes, flujo de control y datos, etc.
- ✓ Los casos de prueba son seleccionados en base a la estructura del código.
- ✓ ¡La estructura del trabajo es el foco de atención!
  - La técnica de caja blanca también es conocida como prueba basada en la estructura o prueba basada en el flujo de control.



## CAJA BLANCA

## TÉCNICAS DINÁMICAS DE TESTING

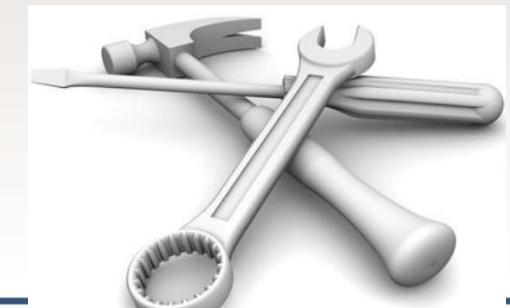
- ✓ Los métodos de caja blanca **requieren el apoyo de herramientas**, lo que asegura la **calidad** de las pruebas e incrementa su **eficiencia**.
- ✓ Dada la complejidad de las mediciones necesarias para las pruebas de caja blanca, la ejecución manual implica: consumo de tiempo y recursos, dificultad en la implementación y propensión a errores.



## CAJA BLANCA

## TÉCNICAS DINÁMICAS DE TESTING

- ✓ Los métodos de caja blanca **requieren el apoyo de herramientas**, lo que asegura la **calidad** de las pruebas e incrementa su **eficiencia**.
- ✓ Dada la complejidad de las mediciones necesarias para las pruebas de caja blanca, la ejecución manual implica: consumo de tiempo y recursos, dificultad en la implementación y propensión a errores.
- ✓ **Métodos de caja blanca (basados en la cobertura)**
  - Cobertura de sentencias.
  - Cobertura de decisión.
  - Cobertura de condición (simple y múltiple).
  - Cobertura de caminos.
  - ...



CAJA BLANCA

TÉCNICAS DINÁMICAS DE TESTING

✓ **Método: Cobertura de Sentencias**

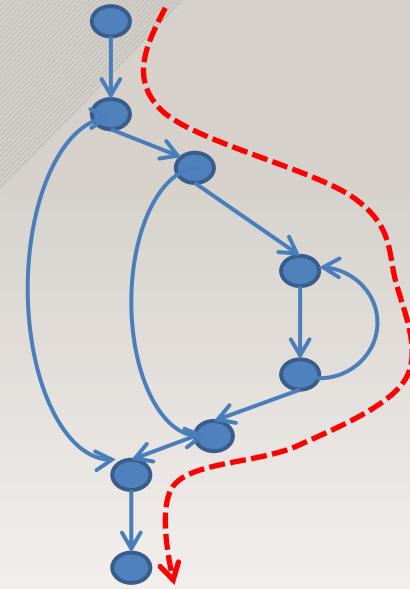
- Técnica basada en el análisis del gráfico del flujo de control (sentencias = nodos; flujo de control = aristas). El foco de atención es la sentencia del código !!
- **Objetivo:** lograr la cobertura de un porcentaje específico ( $C_0$ ) de todas las sentencias. Cada sentencia se debe ejecutar, al menos, una vez.
  - $C_0 = 100\% * (\text{nº sentencias ejecutadas} / \text{nº total sentencias})$

✓ **Método: Cobertura de Sentencias.** Ejemplo

```
if (i > 0) {
 j = f (i);
 if (j > 10) {
 for (k=i; k > 10; k--) {
 ...
 }
 }
}
```

Todas las sentencias pueden ser alcanzadas haciendo uso de un único camino → 100% de cobertura de sentencia.

Un único caso prueba.



## CAJA BLANCA

## TÉCNICAS DINÁMICAS DE TESTING

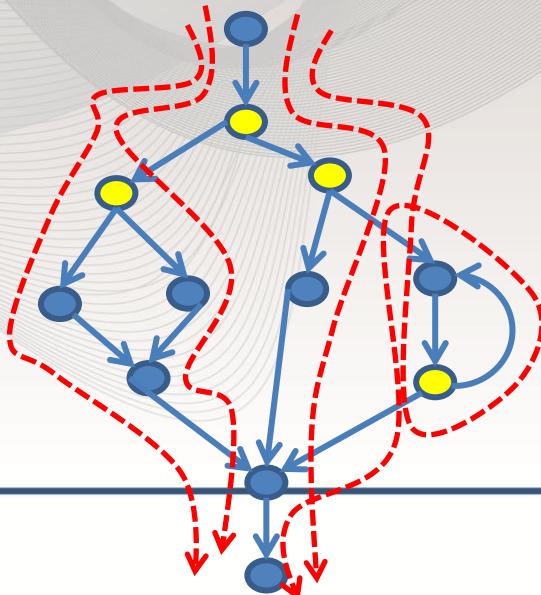
✓ **Método: Cobertura de Decisión**

- Se centra en el flujo de control (aristas del diagrama de flujo)
- **Objetivo:** Todas las aristas del diagrama de flujo tiene que ser cubiertas al menos una vez.  
Lograr un porcentaje de cobertura de todas las decisiones ( $C_1$ )
  - $C_1 = 100\% * (\text{nº decisiones ejecutadas} / \text{nº total decisiones})$

## ✓ Método: Cobertura de Decisión

- Se centra en el flujo de control (aristas del diagrama de flujo)
  - **Objetivo:** Todas las aristas del diagrama de flujo tiene que ser cubiertas al menos una vez.  
Lograr un porcentaje de cobertura de todas las decisiones ( $C_1$ )
    - $C_1 = 100\% * (\text{nº decisiones ejecutadas} / \text{nº total decisiones})$

## ✓ Ejemplo:



- ¿Cuántos caminos consigue una cobertura de precisión del 100%?
  - Una cobertura de decisión del 100% requiere, al menos, los mismos casos de prueba que una cobertura sentencia.

✓ **Método: Cobertura de Condición**

- **Objetivo:** detectar defectos en la implantación de condiciones.
- En este criterio es necesario presentar un número suficiente de casos de prueba de modo que cada condición en una decisión tenga, al menos una vez, todos los resultados posibles.
- **Tipos:**
  - Cobertura de condición simple.
  - Cobertura de condición múltiple.

✓ **Método: Cobertura de Condición SIMPLE**

- Cada subcondición atómica de una sentencia condicional tiene que tomar, al menos una vez, los valores verdadero ("true") y falso ("false").
- ✓ **Ejemplo:** considerar la condición  $A>2 \text{ OR } B<6$ . En los casos de prueba para la cobertura de condición simple podrían ser (por ejemplo):

|               |               |                   |
|---------------|---------------|-------------------|
| A = 6 (true)  | B = 9 (false) | A>2 OR B<6 (true) |
| A = 1 (false) | B = 2 (true)  | A>2 OR B<6 (true) |

- Con sólo dos casos de prueba se puede lograr una cobertura de condición simple.
  - Cada subcondición ha tomado los valores verdadero y falso.
- Sin embargo, el resultado combinado es verdadero en ambos casos.

✓ **Método: Cobertura de Condición MÚLTIPLE**

- Todas las combinaciones que pueden ser creadas utilizando permutaciones de las subcondiciones atómicas deben formar parte de las pruebas.

✓ **Ejemplo:** considerar la condición  $A>2 \text{ OR } B<6$ . En los casos de prueba para la cobertura de condición múltiple podrían ser (por ejemplo):

| A = 6 (true)  |               | B = 9 (false)     | A>2 OR B<6 (true) |
|---------------|---------------|-------------------|-------------------|
| A = 6 (true)  | B = 2 (true)  | A>2 OR B<6 (true) |                   |
| A = 1 (false) | B = 2 (true)  | A>2 OR B<6 (true) |                   |
| A = 1 (false) | B = 9 (false) | A>2 OR B<6        | (false)           |

- Con sólo 4 casos de prueba se puede lograr una cobertura de condición múltiple.
  - Se han creado todas las combinaciones verdadero/falso.
  - Se han logrado todos los posibles resultados de la condición.
- nº casos de prueba exponencial:  $2^n$ , donde n = nº condiciones atómicas

✓ **Método: Cobertura de Camino**

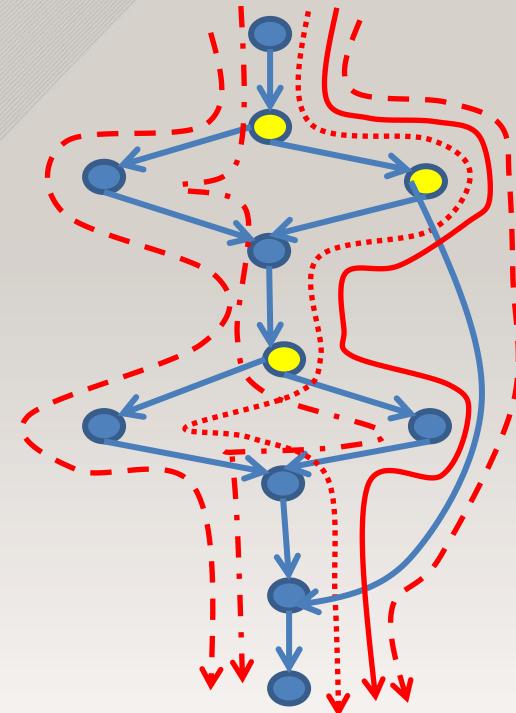
- Consiste en ejecutar todos los posibles cambios a través de un programa. Esto puede conducir a un nº muy alto de casos de prueba.
  - Camino: secuencia de instrucciones en el flujo de control (nodos y aristas en el diagrama de control).
  - Cada camino va desde el nodo inicial al final del diagrama de flujo de control.
- Para una cobertura de decisión, un solo camino a través de un bucle es suficiente. Sin embargo, en la cobertura por caminos hay más casos de prueba:
  - Un caso prueba no entrante en el bucle
  - Un caso prueba adicional para cada ejecución del bucle
- **Objetivo:** alcanzar un porcentaje definido de cobertura de camino (CC):
  - $CC = 100\% * (\text{nº caminos cubiertos} / \text{nº total caminos})$

CAJA BLANCA

TÉCNICAS DINÁMICAS DE TESTING

✓ Ejemplo:

- ¿Cuántos casos de prueba para una cobertura de camino del 100%? → 5 casos de prueba
- ¿Cuántos casos de prueba para una cobertura de sentencia del 100%? → 2 casos de prueba
- ¿Cuántos casos de prueba para una cobertura de decisión del 100%? → 3 casos de prueba

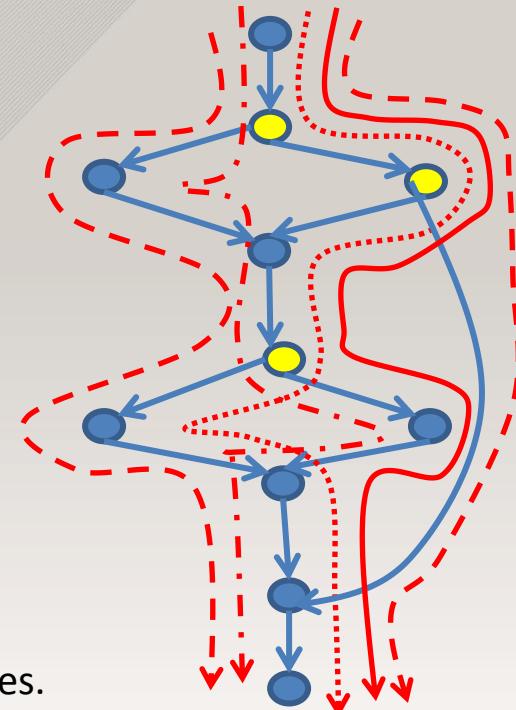


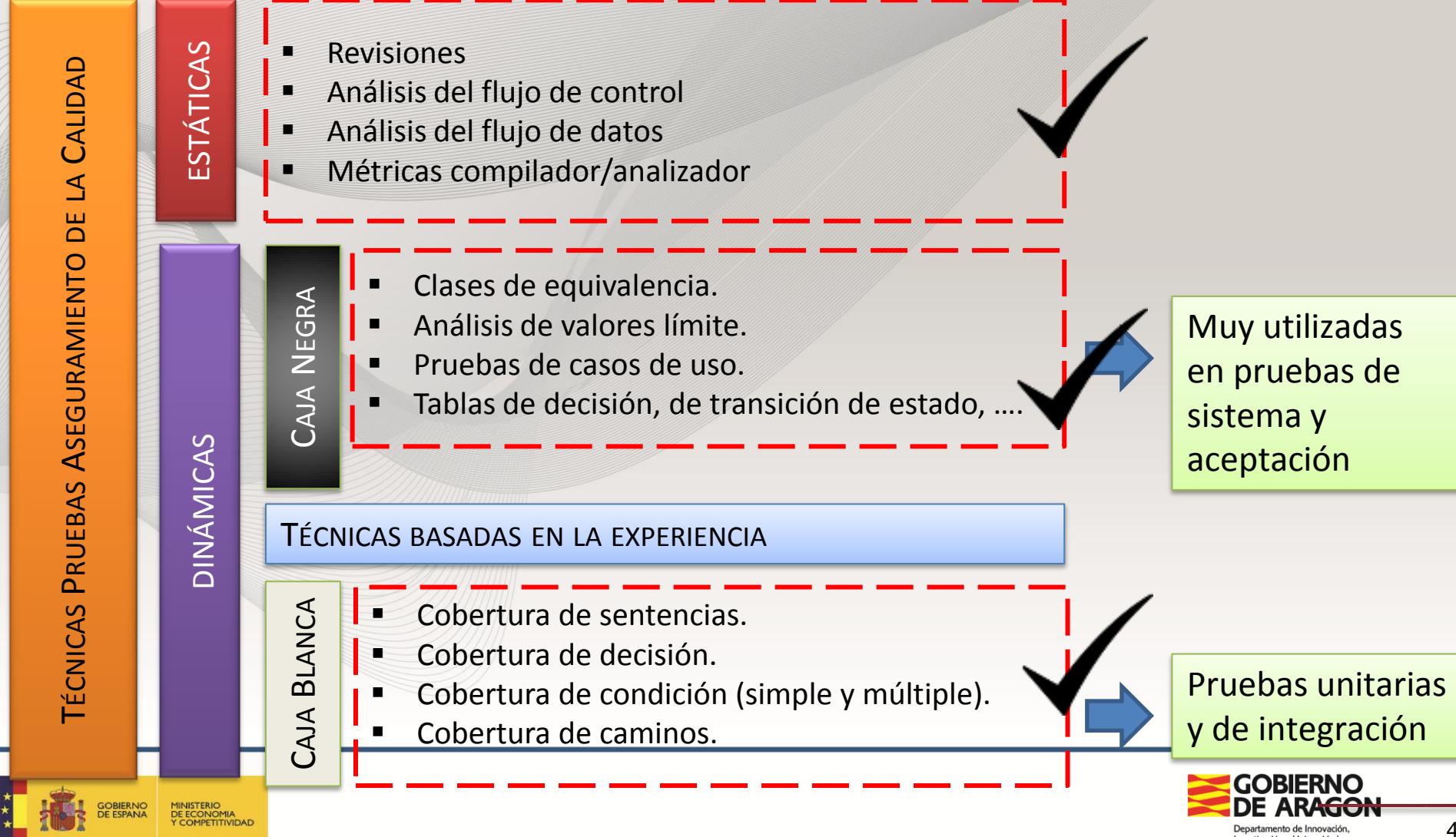
✓ Ejemplo:

- ¿Cuántos casos de prueba para una cobertura de camino del 100%? → 5 casos de prueba
- ¿Cuántos casos de prueba para una cobertura de sentencia del 100%? → 2 casos de prueba
- ¿Cuántos casos de prueba para una cobertura de decisión del 100%? → 3 casos de prueba

✓ Aspectos a tener en cuenta:

- El 100% de cobertura de caminos sólo en programas muy simples.
- La cobertura de camino es más exhaustiva que la cobertura de sentencia y de decisión.
- El 100% de cobertura de camino incluye el 100% de cobertura de decisión que a su vez incluye el 100% de cobertura de sentencia.





# Índice

- 1. Conceptos fundamentales del testeo de software**
- 2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- 3. Técnicas de testing: métodos para generar un caso de prueba**
- 4. Herramientas de testing: el soporte para automatizar actividades de testing**
- 5. Métricas de testing: midiendo la calidad del software**
- 6. Procesos de testing: cómo integrar el testeo en la metodología**

- Actualmente el número de herramientas para pruebas de software disponibles, tanto en el mercado como de manera gratuita (herramientas de código abierto), se pueden organizar de la siguiente manera:
  - Herramientas de gestión de pruebas
  - Herramientas para pruebas funcionales
  - Herramientas para pruebas de carga y rendimiento.
  - Herramientas de calidad del producto software

## Herramientas de testing: el soporte para automatizar actividades de testing

- Herramientas de gestión de pruebas
  - **Bugzilla Testopia**
  - **FitNesse**
  - **qaManager**
  - **qaBook**
  - **RTH (open source)**
  - **Salome-tmf**
  - **Squash TM**
  - **Test Environment Toolkit**
  - **TestLink**
  - **Testitool**
  - **XQual Studio**
  - **Radi-testdir**
  - **Data Generator**
- Herramientas para pruebas funcionales
  - **Selenium**
  - **Soapui**
  - **Watir (Pruebas de aplicaciones web en Ruby)**
  - **WatiN (Pruebas de aplicaciones web en .Net)**
  - **Capedit**
  - **Canoo WebTest**
  - **Solex**
  - **Imprimatur**
  - **SAMIE**
  - **ITP**
  - **WET**
  - **WebInject**

### Herramientas Open Source



- Herramientas para pruebas de carga y rendimiento
  - **FunkLoad**
  - **FWPTT load testing**
  - **loadUI**
  - **jmeter**
- Herramientas de calidad del Producto Software
  - **PMD**
  - **Check Style**
  - **SONAR**
  - **Simian**

## Herramientas de testing: el soporte para automatizar actividades de testing

- **Herramientas de gestión de pruebas**
  - **HP Quality Center/ALM**
  - **QA Complete**
  - **qaBook**
  - **T-Plan Professional**
  - **SMARTS**
  - **QAS.Test Case Studio**
  - **PractiTest**
  - **SpiraTest**
  - **TestLog**
  - **ApTest Manager**
  - **Zephyr**
- **Herramientas para pruebas funcionales**
  - **QuickTest Pro**
  - **Rational Robot**
  - **Sahi**
  - **SoapTest**
  - **Test Complete**
  - **QA Wizard**
  - **Squish**
  - **vTest**
  - **Internet Macros**

### Herramientas Comerciales



- **Herramientas para pruebas de carga y rendimiento**
  - **HP LoadRunner**
  - **LoadStorm**
  - **NeoLoad**
  - **WebLOAD Professional**
  - **Forecast**
  - **ANTS – Advanced .NET Testing System**
  - **Webserver Stress Tool**
  - **Load Impact**
- **Herramientas de calidad del Producto Software**
  - **CheckKing QA**
  - **Kiuwan**
  - **Google CodePro Analytix**
  - **Simian**

## Herramientas de testing: el soporte para automatizar actividades de testing

- **Herramientas Todo en Uno**  
–**Test Studio**– Una herramienta para pruebas de rendimiento, carga, pruebas automáticas, gestión de pruebas y test exploratorio.
- **Herramientas para pruebas sobre teléfonos móviles**  
–**Testdroid**– Herramienta para pruebas automatizadas para Android.

### Herramientas Comerciales



The screenshot shows the TestLink 1.9.13 [DEV] application running in a browser window. The URL in the address bar is [demo.testlink.org/latest/index.php](http://demo.testlink.org/latest/index.php). The top navigation bar includes the user name "Francisco José" and standard window controls. The main content area is divided into several sections:

- Gestión de Proyectos de Pruebas:** Asignar Roles a Usuarios, Gestión de Keywords, Gestión de Plataformas, Inventario.
- Requisitos:** Documento de Especificación de Requisitos, Resumen de Requisitos, Buscar Requisitos, Buscar Especificación de Requisitos, Asignar Requisitos, Imprimir Especificación de Requisitos.
- Especificación de Pruebas:** Editar Caso(s) de Prueba, Buscar Caso(s) de Prueba, Casos de Prueba creados por Usuario.
- Test Plan Section:** Shows "Test Link 1.9.13 [DEV] (Stormbringer) 20141226" and "Proyecto de Pruebas 00:This Project is only for test". It displays the "Plan de Pruebas Actual: Test plan 1".
- Gestión de Planes de Pruebas:** Gestión de Planes de Pruebas, Gestión de Builds.
- Ejecución de Pruebas:** Ejecutar Casos de Prueba, Informes y Métricas de Pruebas, Resumen de Métricas.
- Contenidos del Plan de Pruebas:** Agregar / Quitar Casos de Prueba, Asignar la Ejecución de Casos de Prueba.

At the bottom center of the page is the URL <http://demo.testlink.org>.

**SeleniumHQ**  
Browser Automation[edit this page](#)

search selenium:

[Projects](#)[Download](#)[Documentation](#)[Support](#)[About](#)

## Selenium Projects

Selenium has many projects that combine to make a versatile testing system.

### Selenium WebDriver



[Selenium WebDriver](#) can drive a browser natively either locally or on remote machines.

### Selenium Grid



[Selenium Grid](#) takes Selenium Remote Control to another level by running tests on many servers at the same time, cutting down on the time it takes to test multiple browsers or operating systems.

### Selenium IDE



[Selenium IDE](#) is a Firefox add-on that makes it easy to record and playback tests in Firefox 2+. You can even use it to generate code to run the tests with Selenium Remote Control.

### Selenium Remote Control



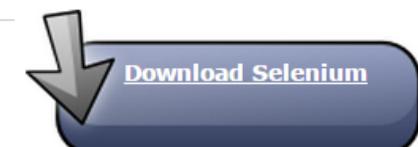
[Selenium Remote Control](#) is a client/server system that allows you to control web browsers locally or on other computers, using almost any programming language and testing framework.



**Selenium is a suite of tools** to automate web browsers across many platforms.

Selenium...

- runs in [many browsers](#) and [operating systems](#)
- can be controlled by [many programming languages](#) and [testing frameworks](#).



# Herramientas de testing: el soporte para automatizar actividades de testing

<http://www.seleniumhq.org/>

**java** **csharp** **python** **ruby** **php** **perl** **javascript**

```
package org.openqa.selenium.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Selenium2Example {
 public static void main(String[] args) {
 // Create a new instance of the Firefox driver
 // Notice that the remainder of the code relies on the interface,
 // not the implementation.
 WebDriver driver = new FirefoxDriver();

 // And now use this to visit Google
 driver.get("http://www.google.com");
 // Alternatively the same thing can be done like this
 // driver.navigate().to("http://www.google.com");

 // Find the text input element by its name
 WebElement element = driver.findElement(By.name("q"));

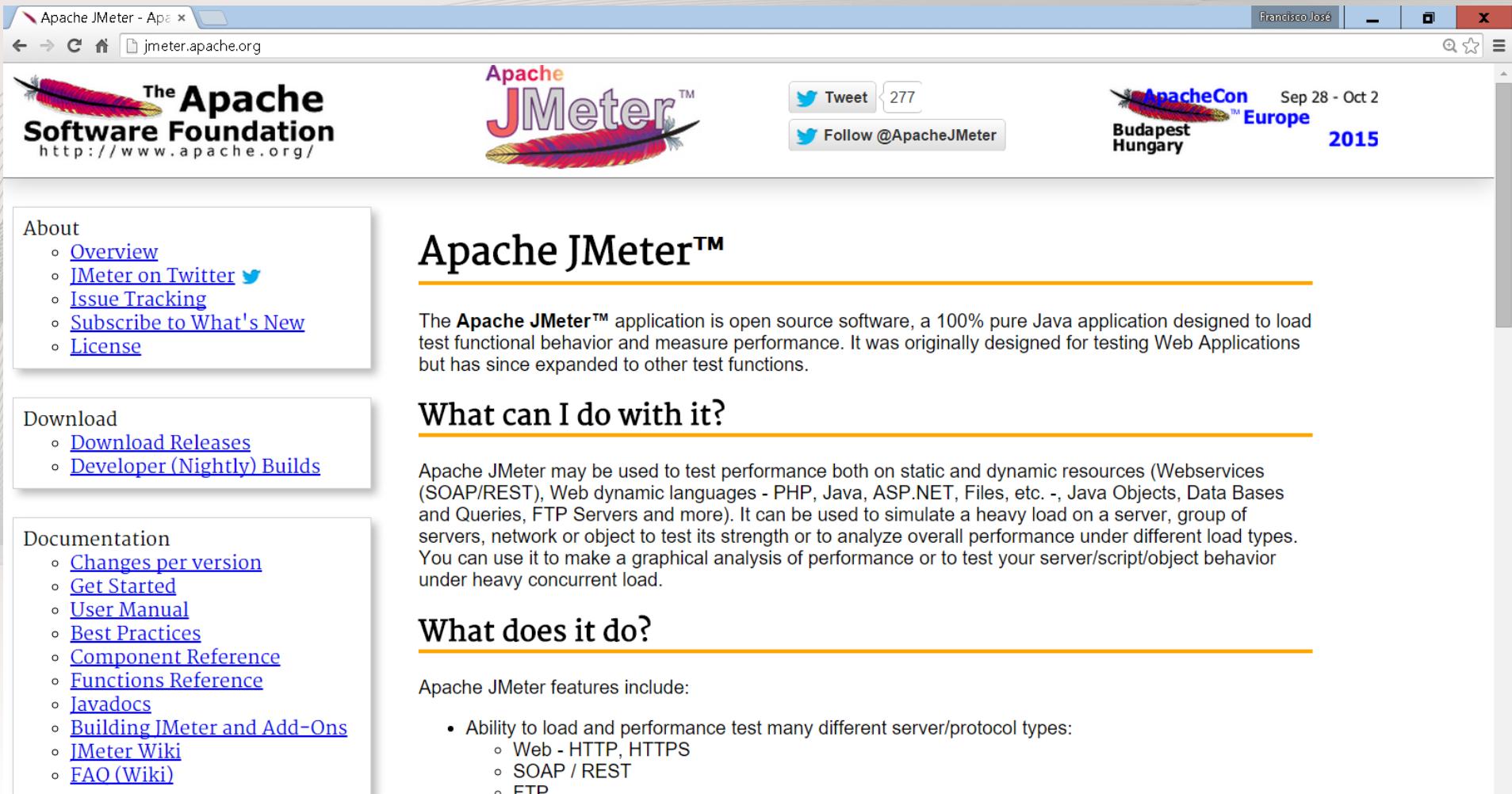
 // Enter something to search for
 element.sendKeys("Cheese!");

 // Now submit the form. WebDriver will find the form for us from the element
 element.submit();

 // Check the title of the page
 System.out.println("Page title is: " + driver.getTitle());

 // Google's search is rendered dynamically with JavaScript.
 // Wait for the page to load, timeout after 10 seconds
 (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
 public Boolean apply(WebDriver d) {
 return d.getTitle().toLowerCase().startsWith("cheese!");
 }
 });
 // Should see: "cheese! - Google Search"
 System.out.println("Page title is: " + driver.getTitle());

 //Close the browser
 driver.quit();
 }
}
```

<http://jmeter.apache.org>


The screenshot shows the Apache JMeter website at <http://jmeter.apache.org>. The page features the Apache Software Foundation logo and the Apache JMeter logo with a feather. It includes social media links for Twitter and a sidebar with navigation links for About, Download, and Documentation.

**About**

- Overview
- JMeter on Twitter
- Issue Tracking
- Subscribe to What's New
- License

**Download**

- Download Releases
- Developer (Nightly) Builds

**Documentation**

- Changes per version
- Get Started
- User Manual
- Best Practices
- Component Reference
- Functions Reference
- Iavadocs
- Building JMeter and Add-Ons
- JMeter Wiki
- FAQ (Wiki)

**Apache JMeter™**

The Apache JMeter™ application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

**What can I do with it?**

Apache JMeter may be used to test performance both on static and dynamic resources (Webservices (SOAP/REST), Web dynamic languages - PHP, Java, ASP.NET, Files, etc. -, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behavior under heavy concurrent load.

**What does it do?**

Apache JMeter features include:

- Ability to load and performance test many different server/protocol types:
  - Web - HTTP, HTTPS
  - SOAP / REST
  - FTP

The screenshot shows the Apache JMeter application window. On the left, the **Test Plan** tree is displayed, showing a hierarchy of samplers and controllers. A blue box highlights the **Menú principal** (Main menu) in the top navigation bar. Another blue box highlights the **Threads activos y total de threads** (Active threads and total threads) status bar at the top right. A third blue box highlights the **Panel de control** (Control panel) in the bottom center. A fourth blue box highlights the **testplan en forma de árbol** (Test plan as a tree) label pointing to the tree view on the left. The main panel displays the **Response Assertion** configuration, including fields for Name, Comments, Apply to, Response Field to Test, Pattern Matching Rules, and Patterns to Test, along with several regular expression patterns listed.

<http://nemo.sonarqube.org/>

**sonarqube** Cuadros de mando ▾ Evidencias Medidas Reglas Perfiles Umbrales Más ▾

**ALL PROJECTS**

Calificación SQALE A Ratio De Deuda Técnica 5.3%

Deuda Técnica Líneas De Código  
42,435d ↗ 12,934K ↗

**ALL PROJECTS**

Debt 42,435d ↗ Evidencias 1,928,037 ↗

| Estado     | Cantidad  |
|------------|-----------|
| Bloqueante | 19,046 ↗  |
| Critica    | 58,225 ↗  |
| Mayor      | 848,265 ↗ |
| Menor      | 914,858 ↗ |
| Info       | 87,643 ↗  |

Global Security Issue Tags

| Categoría         | Cantidad |
|-------------------|----------|
| error-handling    | 52692    |
| owasp-a6          | 956      |
| owasp-a1          | 802      |
| injection         | 14       |
| multi-threading   | 5874     |
| sans-top25-risky  | 820      |
| owasp-top10       | 316      |
| sans-top25        | 1044     |
| owasp-a2          | 815      |
| denial-of-service | 299      |

**FORGES**

| Plataforma  | Cantidad |
|-------------|----------|
| Apache      | ~45%     |
| Others      | ~35%     |
| Sourceforge | ~10%     |
| Codehaus    | ~5%      |
| OW2         | ~2%      |
| OPS4J       | ~1%      |
| GoogleCode  | ~1%      |

**ALL PROJECTS**

Size: Líneas de código Color: Calificación SQALE

Solo se muestran los primeros 100 componentes

**ALL PROJECTS**

1 de marzo del 2015 Líneas de código: 6,830,796 Líneas duplicadas: 548,523 Tests unitarios: 83,837

## Herramientas de testing: el soporte para automatizar actividades de testing

Dashboards Projects Measures Issues Quality Profiles Log in Search

Forges > SpringSource >

**Dashboard**

SQALE  
Issues  
Hotspots  
Time Machine  
By Developers

TOOLS  
Components  
Issues Drilldown  
Clouds  
Compare

**sonarqube**

Oct 04 2013 06:10 Time changes...

**Lines of code**  
**51,192 ↗**  
113,876 lines ↗  
18,350 statements ↗  
1,034 files  
2 projects

**Classes**  
**1,140**  
157 packages  
5,309 methods ↗  
997 accessors

**Issues**  
**1,627 ↗**  
Rules compliance **92.1%**

**Blocker** 0  
**Critical** 0  
**Major** 1,341 ↗  
**Minor** 6 ↗  
**Info** 280 ↗

**Alerts** : 1 project in warning.

**Unit Tests Coverage**  
**77.2%**  
79.1% line coverage  
71.9% branch coverage

**Unit test success**  
**99.3%**  
0 failures  
28 errors ↗  
3,854 tests  
2 skipped  
3:54 min ↗

**Complexity**  
**2.1 /method**  
**9.6 /class**  
**10.6 /file**  
Total: 10,960 ↗

**Methods**  **Files**

**Documentation**  
**62.5% docu. API**  
4,411 public API ↗  
1,653 undocu. API

**Comments**  
**27.3%**  
19,216 lines ↗

**Package tangle index**  
**0.0%**  
> 0 cycles

**Dependencies to cut**  
0 between packages  
0 between files

**Duplications**  
**2.2%**  
2,539 lines ↗  
82 blocks  
44 files

**Size: Lines of code** **Color: Coverage 0.0%**

**06:10**

**● Lines of code: 51,192**  
**● Coverage: 77.2%**  
**● Duplicated lines: 2,539**

## Herramientas de testing: el soporte para automatizar actividades de testing

<https://sonarqubehispano.org>



The screenshot shows the homepage of the SonarQube Hispano community. At the top, there is a navigation bar with links for 'SonarQube Hispano', 'Documentación', 'Noticias', 'Preguntas', a search icon, 'Iniciar sesión', and 'Registrarse'. The main title 'sonarQube' is displayed with a stylized 'Q' icon. Below it, the subtitle 'Comunidad Hispana de usuarios de SonarQube' is shown. A text block explains the purpose of the community: 'Aquí podrás encontrar toda la información y ayuda en español de SonarQube, y puedes resolver tus dudas, conocer a los mejores expertos e incluso convertirte en uno de ellos.' It also highlights the size of the community: 'SonarQube es el estándar de facto para la inspección continua de código, con más de 250.000 descargas en todo el mundo y más de 1.500 miembros activos en la comunidad.' A call-to-action button 'Hacer pregunta' is visible. At the bottom, there is a footer with links to 'Comunidad hispana de SonarQube', 'Le gusta Confluence?', 'Con tecnología Confluence de Atlassian 5.7, el software de colaboración en equipo', 'Petición de revisión de Bug/Característica', 'Noticias de Atlassian', and logos for 'GOBIERNO DE ARAGÓN' and 'Departamento de Innovación, Investigación y Universidad'.

# Índice

- 1. Conceptos fundamentales del testeo de software**
- 2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- 3. Técnicas de testing: métodos para generar un caso de prueba**
- 4. Herramientas de testing: el soporte para automatizar actividades de testing**
- 5. Métricas de testing: midiendo la calidad del software**
- 6. Procesos de testing: cómo integrar el testeo en la metodología**

## CALIDAD DEL SOFTWARE

- El **software es un producto** como cualquier otro, y por tanto podemos hablar de
  - software de buena calidad y
  - software de mala calidad.
- La **calidad del software** comprende distintos aspectos como estética (que sea agradable a la vista), funcionalidad (que sea fácil de usar), eficiencia (que ejecute con rapidez y precisión los procesos), etc.
- Lo que **distingue al software de otros productos industriales** es que
  - no es de naturaleza material, no se puede tocar.
  - Por tanto no resulta viable hacer una valoración del mismo en base a una impresión rápida o análisis del aspecto ni en base al coste de materiales componentes.
- La **calidad del producto, junto con la calidad del proceso, es uno de los aspectos más importantes actualmente en el desarrollo de Software.** Relacionada con la calidad del producto,

## Introducción a la medición

### Aportaciones de la medición a la resolución de problemas

**La medición contribuye a superar algunos problemas habituales en el desarrollo del software**

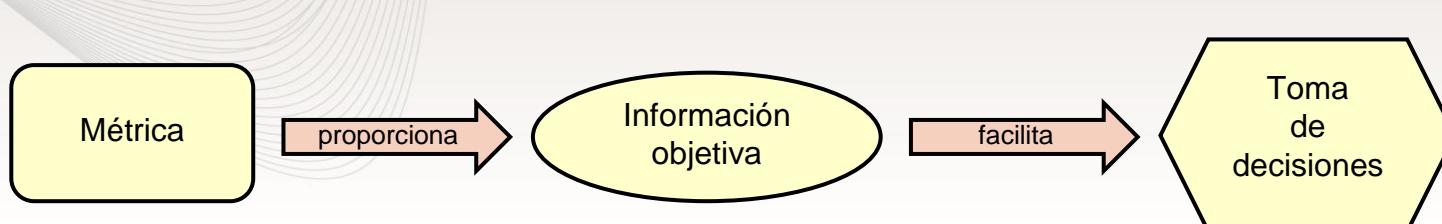
| Problema                     | Medir ayuda a                                                                          |
|------------------------------|----------------------------------------------------------------------------------------|
| Requisitos incorrectos       | Desarrollar requisitos verificables expresados en términos medibles.                   |
| Toma de decisiones           | Proporciona evidencia cuantificable para apoyar las decisiones.                        |
| Falta de control             | Hacer más visible el desarrollo e identificar problemas anticipadamente.               |
| Exceso de gasto              | Realizar predicciones de coste y plazo justificables.                                  |
| Costes de mantenimiento      | Recomendar determinadas estrategias de prueba e identificar los módulos problemáticos. |
| Evaluación de nuevos métodos | Valorar los efectos en la productividad y la calidad .                                 |

### Necesidad de medir

- *No se puede controlar lo que no se puede medir. (DeMarco)*
- *No se puede predecir lo que no se puede medir. (Fenton y Pfleeger)*
- *Ingeniería del software: aplicación de una aproximación sistemática, disciplinada y **cuantificable** al desarrollo del software. (Glosario IEEE)*
- La medición posibilita la mejora de la calidad.

### Objetivos de la medición

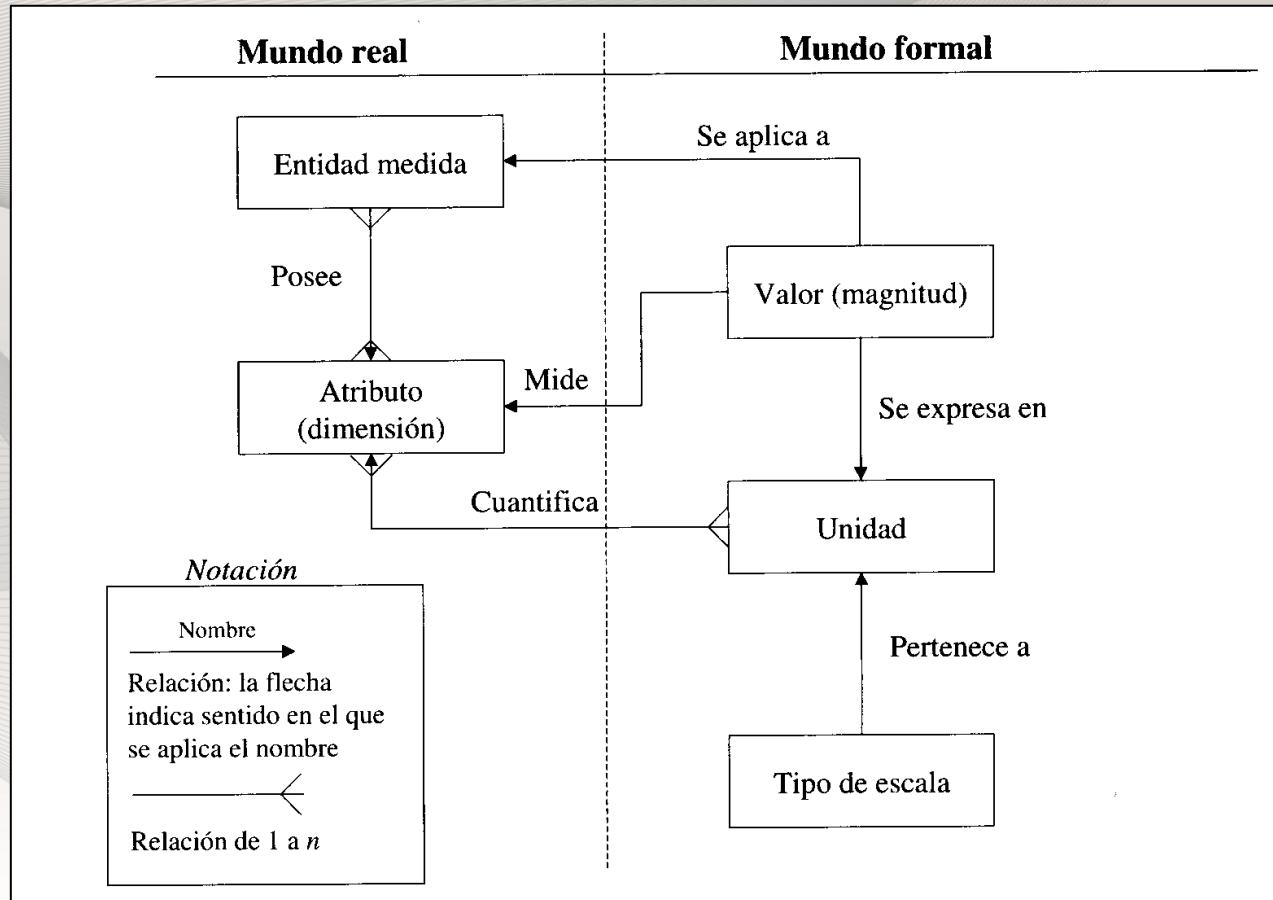
- *Entender lo que ocurre.*
- *Controlar lo que ocurre en los proyectos.*
- *Mejorar los productos y los procesos.*



# Medición en el proceso software

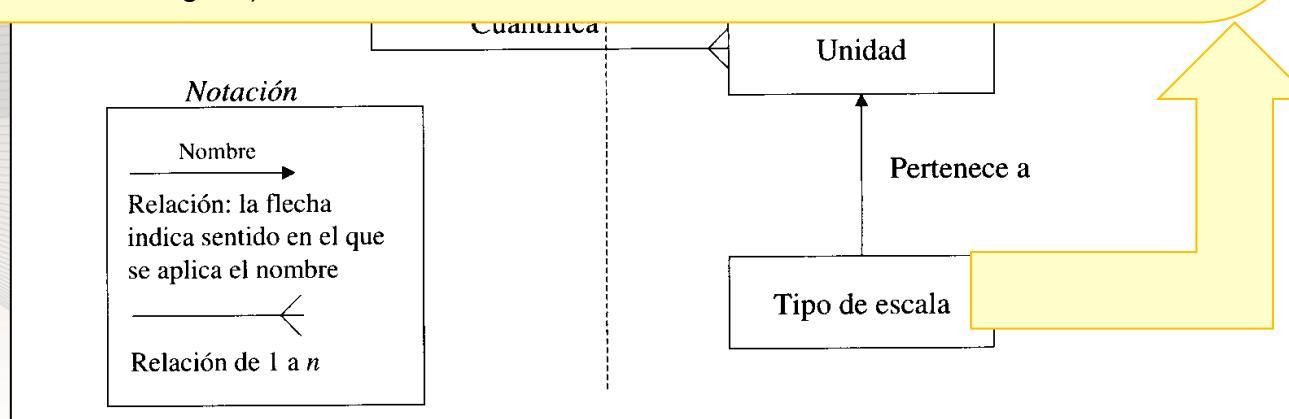
## Introducción a la medición

### Modelo estructural



## Tipos de escala

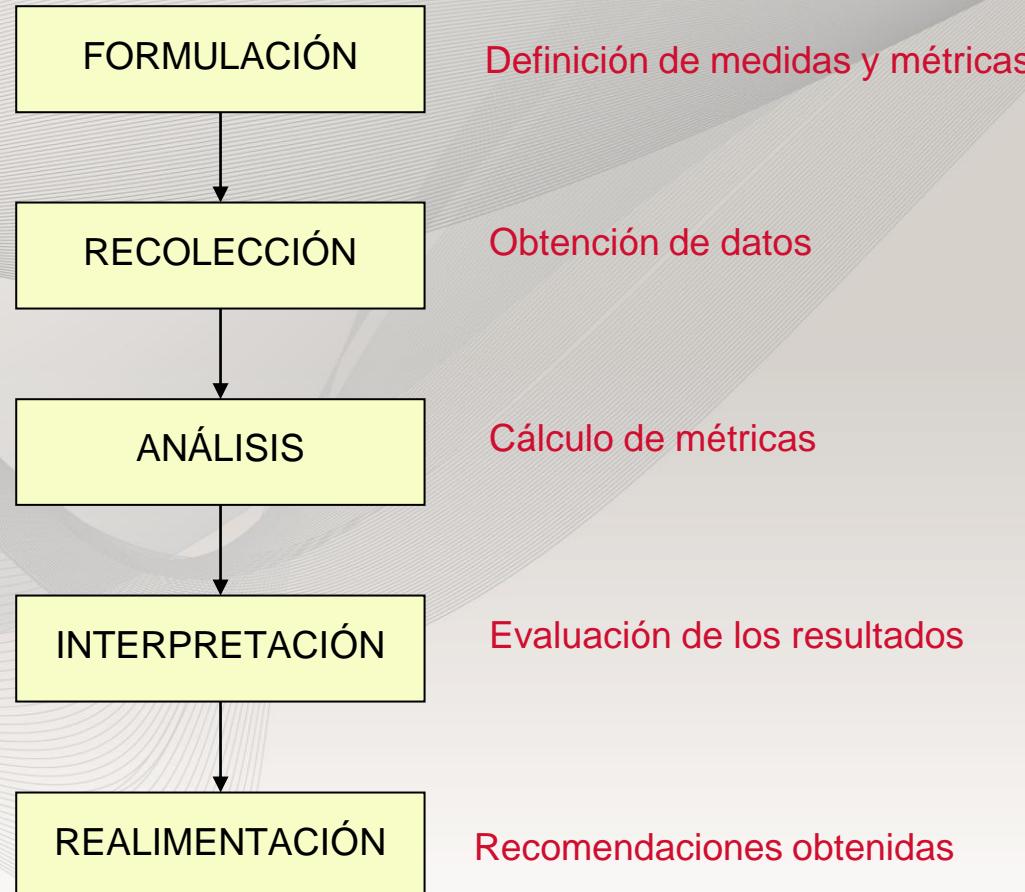
- **Nominal:** lista de las diferentes posiciones que puede adoptar la variable, pero sin que ello signifique ningún tipo de orden o de relación. (Cereal cultivado: trigo, maíz, centeno, etc.)
- **Ordinal:** distinguen los diferentes valores de la variable jerarquizándolos de acuerdo a un rango. (Grado de escolaridad de una persona: 2 años, 3 años, 7 años, ...). No se puede determinar la equivalencia entre las distancias que separan a un valor de otro..
- **Intervalo:** además de poseer las características de la ordinal, tiene la característica de que la distancia entre sus intervalos está claramente determinada y que éstos son iguales entre sí. (Escala termométrica: entre 23 y 24 grados centígrados existe la misma diferencia que entre 45 y 46 grados).
- **Ratio (cociente o razón):** se conservan las propiedades de todos los anteriores pero además se añade la existencia de un cero real, con lo que se hacen posible ciertas operaciones matemáticas, como la obtención de proporciones o cocientes. (Longitud: 20 metros es el doble de 10 metros. Existe el valor real de cero de longitud).



Modelo estructural de Kitchham et al., 1995

- **Entidad:** objeto que va a ser caracterizado mediante una medición de sus atributos. (Ej: persona). (ISO/IEC 15939)
- **Atributo:** característica medible de una entidad. (Ej: altura).
- **Medición:** proceso objetivo y empírico por el que se asignan números o símbolos a atributos de entidades del mundo real con objeto de describirlas (Fenton y Kitchenham, 1991).
- **Métrica:** evaluación del grado en el cual un producto o proceso posee un atributo determinado (extensión, cantidad, dimensiones, capacidad o tamaño) (IEEE, 1993).
- **Escala:** conjunto de valores que permite establecer valores entre medidas. Con frecuencia dicho conjunto es continuo, está ordenado y viene delimitado por un punto inicial y otro final.
- **Métrica directa e indirecta:**
  - Una **métrica** es **directa** si se puede medir directamente del atributo y su valor no depende de la medida de otros atributos (longitud, longitud del código fuente, duración del proceso de prueba, número de defectos...).
  - Una **métrica** es **indirecta** si comprende la medición de varios atributos, es decir, si deriva de otros atributos (volumen, productividad, estabilidad de requisitos, densidad de defectos en un módulo, etc.) (Wohlin et al. 2000).
- **Métrica objetiva y subjetiva:** una métrica es objetiva si su valor no depende del observador y es subjetiva en caso contrario.
- **Indicador:** métrica o combinación de métricas que proporcionan comprensión acerca del proceso, proyecto o producto. (Pressman) (Número de alumnos que aprueban en primera convocatoria)

## Proceso de medición



### Proceso de medición de Roche

## Alcance de las métricas

### Justificación del uso de métricas

1. ¿En cuánto podría ser mejorada la productividad si no tuviese que gastar tiempo en mantenimiento?
2. ¿Cuánto tiempo le costó el último año adaptar su presupuesto en trabajar con nuevas versiones de compiladores, bases de datos o sistemas operativos?
3. ¿Cuáles de las aplicaciones que desarrolla su empresa demanda el mayor tiempo de soporte al usuario?
4. ¿Cuánto tiempo se gasta realmente en testing?
5. ¿Cree que sus desarrolladores dedican suficiente tiempo a actividades de diseño?
6. ¿Su proceso de desarrollo ha madurado en los últimos años?
7. ¿El esfuerzo dedicado a mejorar la calidad del software está reduciendo el tiempo que se dedica a corregir errores ?
8. ¿Con qué precisión es usted capaz de estimar proyectos futuros?
9. ¿En cuántos proyectos han trabajado cada uno de sus desarrolladores en el último año?
10. ¿Cuál es el número medio de horas por semana que sus desarrolladores dedican a un proyecto?

## Alcance de las métricas

### Entidades y atributos

**El primer paso de la medición es identificar las entidades y los atributos a medir.**

#### Entidades:

- **Productos:** componentes, entregas o documentos resultantes de una actividad de proceso. (Ej. **código**)
- **Procesos:** atributos de actividades relacionadas con el software. (Ej. **etapa de análisis**)
- **Recursos:** entidades requeridas por una actividad del proceso. (Ej. **recursos humanos**)

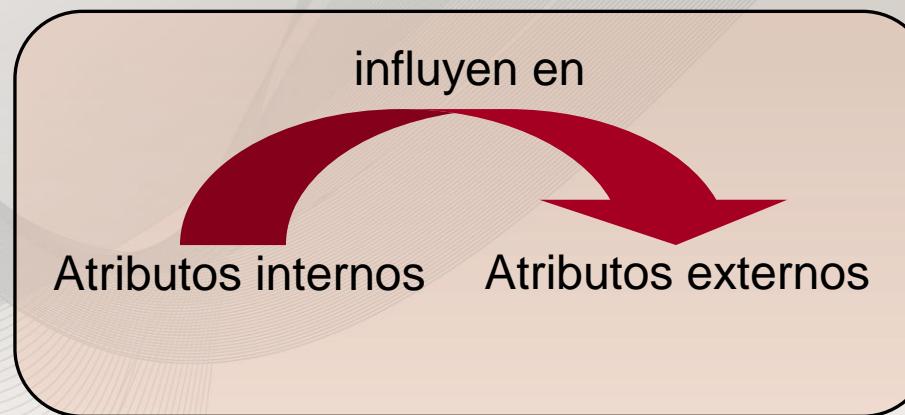
#### Dentro de cada entidad se puede distinguir:

- **Atributos internos:** son aquellos que pueden ser medidos examinando el proceso, producto o recurso mismo.
- **Atributos externos:** se miden con respecto a como el proceso, producto o recurso se relaciona con su entorno.

## Alcance de las métricas

### Entidades y atributos

Generalmente, el interés de conocer el valor de un atributo interno es que pueda *dar información* sobre algún atributo externo de interés para el observador.



- *Ejemplo:* el **número de requisitos** de una especificación de requisitos de sistema puede ayudarnos a predecir el **esfuerzo** asociado al proyecto.

## Alcance de las métricas

### Entidades y atributos

| Producto         | Atributos internos                                | Atributos externos                      |
|------------------|---------------------------------------------------|-----------------------------------------|
| Especificaciones | Tamaño, reutilización, etc.                       | Calidad, legibilidad                    |
| Diseño           | Tamaño, acoplamiento, cohesión, complejidad, etc. | Calidad, complejidad                    |
| Código           | Tamaño, complejidad, etc.                         | Facilidad de mantenimiento, fiabilidad, |
| Casos de prueba  | Nº de casos, % cobertura                          | Calidad                                 |
| Proceso          | Atributos internos                                | Atributos externos                      |
| Requisitos       | Tiempo, esfuerzo, nº de requisitos                | Calidad, coste, estabilidad             |
| Diseño           | Tiempo, esfuerzo, nº de errores                   | Calidad, coste, estabilidad             |
| Pruebas          | Tiempo, esfuerzo, nº de errores                   |                                         |
| Recurso          | Atributos internos                                | Atributos externos                      |
| Personal         | Edad, salario                                     | Productividad, experiencia              |
| Equipos          | Nº personas, estructura del equipo                | Productividad, experiencia              |
| Software         | Coste, nº de licencias                            | Usabilidad, fiabilidad                  |
| Hardware         | Marca, coste, especificaciones técnicas           | Usabilidad, fiabilidad                  |

Ejemplos de métricas de, adoptado de Fenton y Pfleeger (1998)  
Ref: Ingeniería del Software. Un enfoque desde la guía SWEBOK, 2011.



## Medición de atributos internos del producto

### Caracterización de los atributos internos

**Los atributos internos describen los productos de software de forma que dependen únicamente del producto mismo.**

**El producto puede ser descrito en función de su tamaño.** Se pueden definir un conjunto de atributos para medir el tamaño del software:

- **Longitud:** tamaño físico del producto.
- **Funcionalidad:** funciones que proporciona el producto al usuario.
- **Complejidad** (de tiempo o espacio): recursos necesarios (de tiempo o memoria de ordenador) para implementar una solución particular.

**Las propiedades estructurales del software son atributos internos relacionados con la calidad del producto.** Los tipos de medidas estructurales son:

- **Flujo de control:** secuencia en que se ejecutan las instrucciones.
- **Flujo de datos:** seguimiento de cómo los datos se crean y se manejan por un programa.
- **Estructura de los datos:** organización de los datos independiente del programa.

**Los principales productos que resulta útil medir son la especificación, el diseño y el código.**

# ISO/IEC 25000

## Calidad del Producto Software

- Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones.

**ISO/IEC 2501n:**  
División para el modelo de calidad

**ISO/IEC 2502n:**  
División para la medición de calidad

**ISO/IEC 2500n:**  
División para gestión de la calidad

**ISO/IEC 2503n:**  
División para los requisitos de calidad

**ISO/IEC 2504n:**  
División para la evaluación de calidad

[iso25000.com](http://iso25000.com)

- La **ISO/IEC 25000** conocida como **SQuaRE** (System and Software Quality Requirements and Evaluation), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software.
- La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de las normas **ISO/IEC 9126**, que describe las particularidades de un modelo de calidad del producto software, e **ISO/IEC 14598**, que abordaba el proceso de evaluación de productos software.
  - **Un modelo de calidad**, con las características y subcaracterísticas de calidad que se pueden evaluar de un producto software.
    - Métricas y requisitos de calidad**, que se pueden aplicar al producto software y,
    - Un proceso de evaluación** a seguir para determinar la calidad de los productos software.

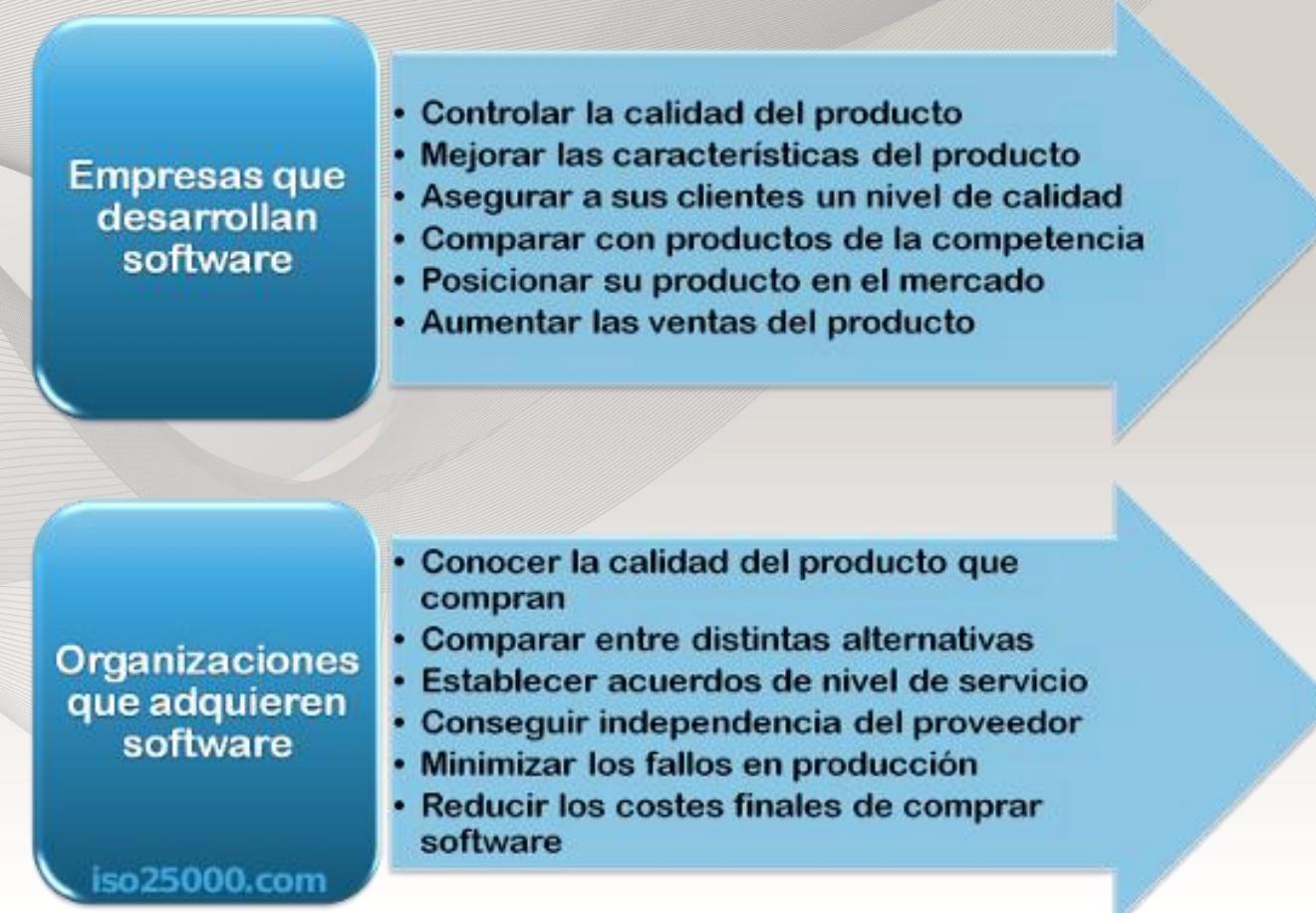
## Evaluación de productos

- Cada día son más las organizaciones que muestran interés en **asegurar o controlar la calidad del producto software**, y aunque cada una de ellas tiene características que las diferencian del resto, de manera global se pueden clasificar en alguna de las siguientes categorías:
  - Organismos de las Administraciones Públicas**, que tanto a nivel estatal como autonómico o local,
    - cada día externalizan más el desarrollo de software a otras empresas o factorías de software, y que
      - necesitan disponer de un control de calidad que les permita verificar que el software que reciben cumple los requisitos mínimos de calidad exigidos
      - y además poder de esta manera gestionar de forma adecuada los acuerdos de nivel de servicio pactados con los proveedores.
  - Empresas de software que externalizan**, ya sea bajo el método del **nearshoring** o bajo el método del **offshoring**, parte de sus procesos de desarrollo de software, y que deben controlar también de forma continua la calidad del software que reciben.
  - Factorías y empresas desarrolladoras de software** que están interesadas en disponer de un mecanismo que les permita asegurar la calidad del software que fabrican.
  - Factorías y empresas desarrolladoras de software** que están interesadas en asegurar a sus clientes, mediante una verificación y validación independientes, la calidad de los productos que les están entregando.

## ISO/IEC 25000 y los motivos para la evaluación

- Son muchos los motivos por los que una organización puede estar interesada en implantar un sistema de control de la calidad del producto bajo la familia de normas ISO/IEC 25000. Entre los más destacados se pueden incluir:
  - Diferenciarse de los competidores**, asegurando tiempos de entrega y reducción de fallos en el producto tras su implantación en producción.
  - Poder establecer acuerdos de nivel de servicio**, definiéndose determinados parámetros de calidad que el producto debe cumplir antes de ser entregado.
  - Detectar los defectos en el producto software** y proceder a su eliminación antes de la entrega, lo que supone un ahorro de costes en la fase de mantenimiento posterior.
  - Evaluar y controlar el rendimiento del producto software desarrollado**, asegurando que podrá generar los resultados teniendo en cuenta las restricciones de tiempo y recursos establecidas.
  - Asegurar que el producto software desarrollado respeta los niveles necesarios para las características de seguridad** (confidencialidad, integridad, autenticidad, no-repudio, etc.).
  - Comprobar que el producto desarrollado podrá ser puesto en producción sin poner en compromiso el resto de sistemas y manteniendo la compatibilidad con las interfaces necesarias**.

Resaltando los beneficios de evaluar el producto software en función de la tipología de organización, podríamos destacar dos: las empresas que desarrollan software y las organizaciones que adquieren software. En la siguiente figura se enumeran estos beneficios:



### ISO/IEC 2500n - División de Gestión de Calidad

- Las normas que forman este apartado definen todos los modelos, términos y definiciones comunes referenciados por todas las otras normas de la familia 25000. Actualmente esta división se encuentra formada por:

**-ISO/IEC 25000 - Guide to SQuaRE:** contiene el modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.

**-ISO/IEC 25001 - Planning and Management:** establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.



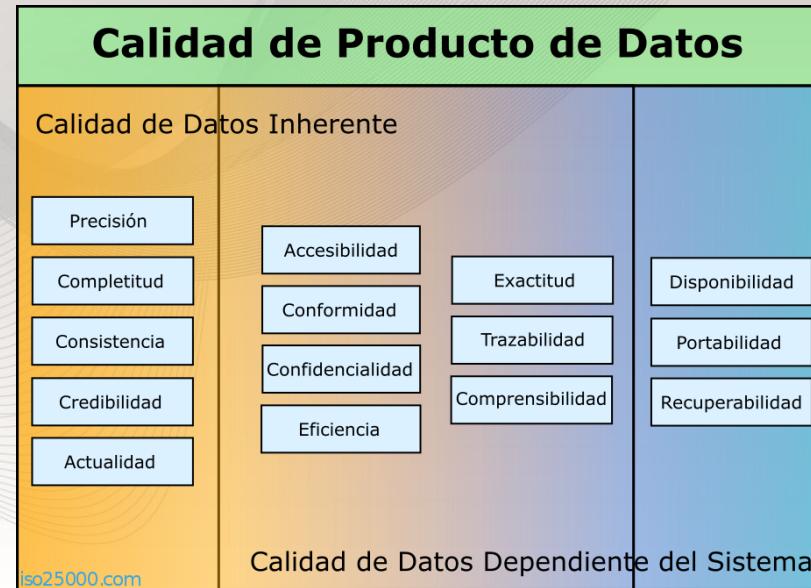
### ISO/IEC 2501n - División de Modelo de Calidad

- Las normas de este apartado presentan modelos de calidad detallados incluyendo características para calidad interna, externa y en uso del producto software. Actualmente esta división se encuentra formada por:

**-ISO/IEC 25010 - System and software quality models:** describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.

**-ISO/IEC 25012 - Data Quality model:** define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran almacenados de manera estructurada y forman parte de un Sistema de Información.




[iso25000.com](http://iso25000.com)


## ISO/IEC 2502n - División de Medición de Calidad

- Estas normas incluyen un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad (interna, externa y en uso) y guías prácticas para su aplicación. Actualmente esta división se encuentra formada por:

- ISO/IEC 25020 - Measurement reference model and guide:** presenta una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad. También proporciona una guía para que los usuarios seleccionen o desarrollen y apliquen medidas propuestas por normas ISO.
- ISO/IEC 25021 - Quality measure elements:** define y especifica un conjunto recomendado de métricas base y derivadas que puedan ser usadas a lo largo de todo el ciclo de vida del desarrollo software.
- ISO/IEC 25022 - Measurement of quality in use:** define específicamente las métricas para realizar la medición de la calidad en uso del producto.
- ISO/IEC 25023 - Measurement of system and software product quality:** define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.
- ISO/IEC 25024 - Measurement of data quality:** define específicamente las métricas para realizar la medición de la calidad de datos.



### ISO/IEC 2503n - División de Requisitos de Calidad

- Las normas que forman este apartado ayudan a especificar requisitos de calidad que pueden ser utilizados en el proceso de elicitation de requisitos de calidad del producto software a desarrollar o como entrada del proceso de evaluación. Para ello, este apartado se compone de:
  - ISO/IEC 25030 - Quality requirements:** provee de un conjunto de recomendaciones para realizar la especificación de los requisitos de calidad del producto software.



## ISO/IEC 2504n - División de Evaluación de Calidad

- Este apartado incluye normas que proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación del producto software. Esta división se encuentra formada por:

- ISO/IEC 25040 - Evaluation reference model and guide:** propone un modelo de referencia general para la evaluación, que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas.
- ISO/IEC 25041 - Evaluation guide for developers, acquirers and independent evaluators:** describe los requisitos y recomendaciones para la implementación práctica de la evaluación del producto software desde el punto de vista de los desarrolladores, de los adquirentes y de los evaluadores independientes.
- ISO/IEC 25042 - Evaluation modules:** define lo que la Norma considera un módulo de evaluación y la documentación, estructura y contenido que se debe utilizar a la hora de definir uno de estos módulos.
- ISO/IEC 25045 - Evaluation module for recoverability:** define un módulo para la evaluación de la subcaracterística Recuperabilidad (Recoverability).



## Establecer los requisitos de la evaluación

Definir el propósito de la evaluación, los requisitos de calidad que se deben considerar, las partes involucradas y el rigor de la evaluación.



## Especificar la evaluación

Determinar las métricas, técnicas y herramientas que se utilizarán para llevar a cabo la evaluación.



## Diseñar la evaluación

Definir el plan con las actividades de evaluación que se deben llevar a cabo



## Ejecutar la evaluación

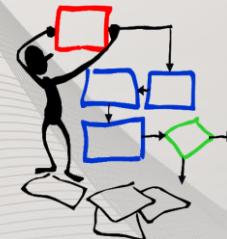
Obtener las mediciones y aplicar los criterios de evaluación determinados en las actividades anteriores



## Concluir la evaluación

Analizar los resultados y elaborar un informe descriptivo para que la organización evaluada conozca la calidad de su producto software

# Métricas de testing: midiendo la calidad del software



## Actividad 1: Establecer los requisitos de la evaluación

**El primer paso del proceso de evaluación consiste en establecer los requisitos de la evaluación.**

Tarea 1.1: Establecer el propósito de la evaluación

Tarea 1.2: Obtener los requisitos de calidad del producto

Tarea 1.3: Identificar las partes del producto que se deben evaluar

Tarea 1.4: Definir el rigor de la evaluación

## Actividad 2: Especificar la evaluación

**En esta actividad se especifican los módulos de evaluación (compuestos por las métricas, herramientas y técnicas de medición) y los criterios de decisión que se aplicarán en la evaluación.**

Tarea 2.1: Seleccionar los módulos de evaluación

Tarea 2.2: Definir los criterios de decisión para las métricas

Tarea 2.3: Definir los criterios de decisión de la evaluación

## Actividad 3: Diseñar la evaluación

**En esta actividad se define el plan con las actividades de evaluación que se deben realizar.**

Tarea 3.1: Planificar las actividades de la evaluación

## Actividad 4: Ejecutar la evaluación

**En esta actividad se ejecutan las actividades de evaluación obteniendo las métricas de calidad y aplicando los criterios de evaluación.**

Tarea 4.1: Realizar las mediciones

Tarea 4.2: Aplicar los criterios de decisión para las métricas

Tarea 4.3: Aplicar los criterios de decisión de la evaluación

## Actividad 5: Concluir la evaluación

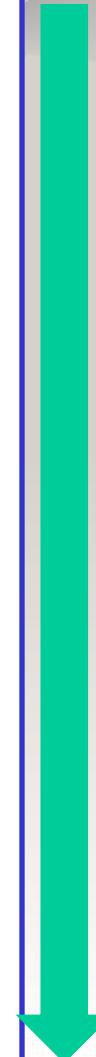
**En esta actividad se concluye la evaluación de la calidad del producto software, realizando el informe de resultados que se entregará al cliente y revisando con éste los resultados obtenidos.**

Tarea 5.1: Revisar los resultados de la evaluación

Tarea 5.2: Crear el informe de evaluación

Tarea 5.3: Revisar la calidad de la evaluación y obtener *feedback*

Tarea 5.4: Tratar los datos de la evaluación



# Índice

- 1. Conceptos fundamentales del testeo de software**
- 2. Niveles de testing: la calidad en cada etapa del ciclo de vida**
- 3. Técnicas de testing: métodos para generar un caso de prueba**
- 4. Herramientas de testing: el soporte para automatizar actividades de testing**
- 5. Métricas de testing: midiendo la calidad del software**
- 6. Procesos de testing: cómo integrar el testeo en la metodología**

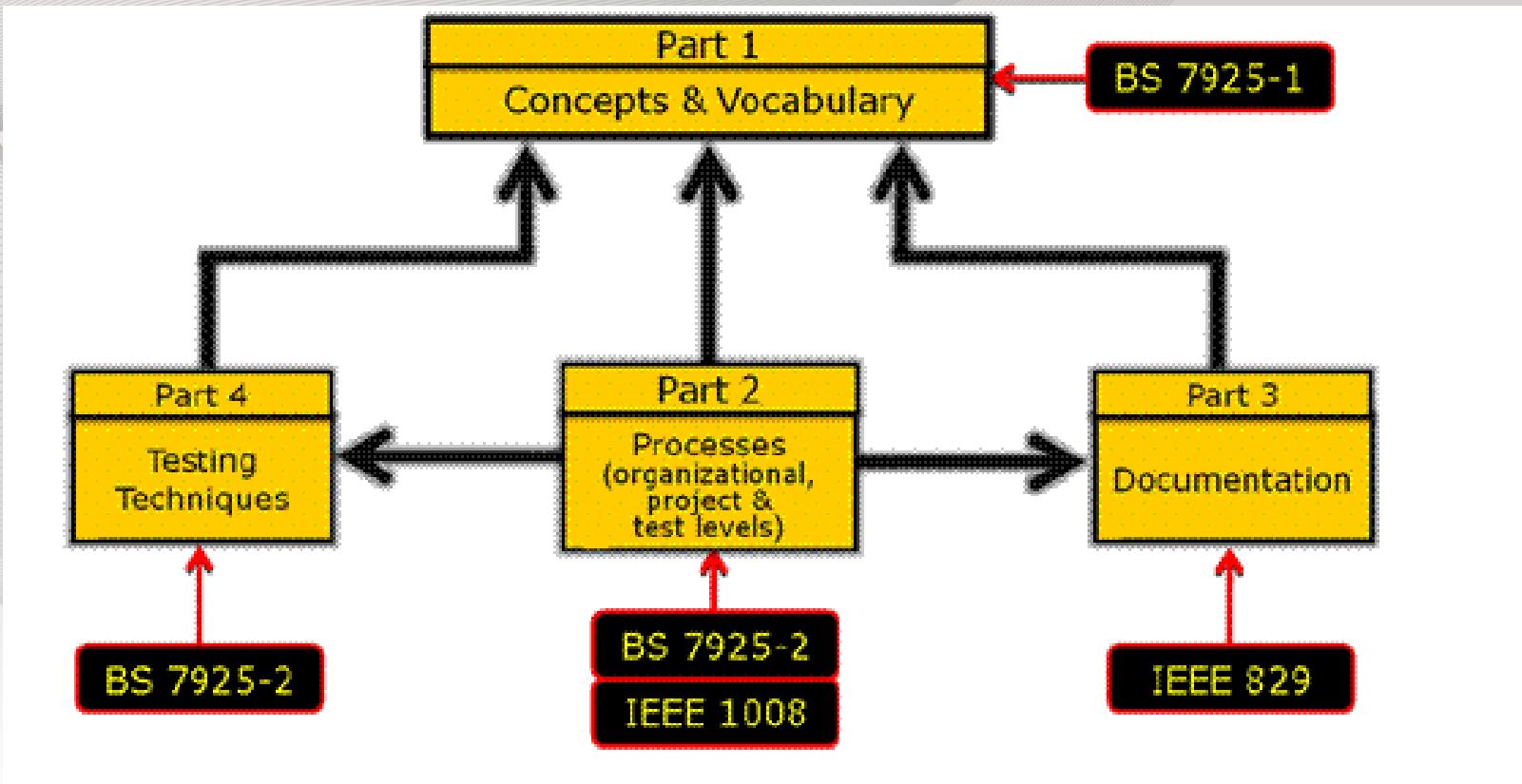
## ISO/IEC/IEEE 29119 Software Testing

- Cobertura de una laguna en el estado actual de los estándares (**Áreas no cubiertas por estándares de pruebas previos**)
  - Aspectos organizativos
  - Proceso y gestión de las pruebas
  - Pocas técnicas funcionales y no funcionales
  - Pruebas basadas en riesgos
- Proveer a los profesionales de una **guía sobre pruebas** cubriendo todos los aspectos del ciclo de vida (**Conceptos, Vocabulario, Proceso, Documentación y Técnicas**).
- **Objetivos**
  - Unificar estándares anteriores en uno solo
  - Cubrir el ciclo de vida completo
  - Aplicable a todo tipo de sistemas software
  - Consistente con otros estándares ISO

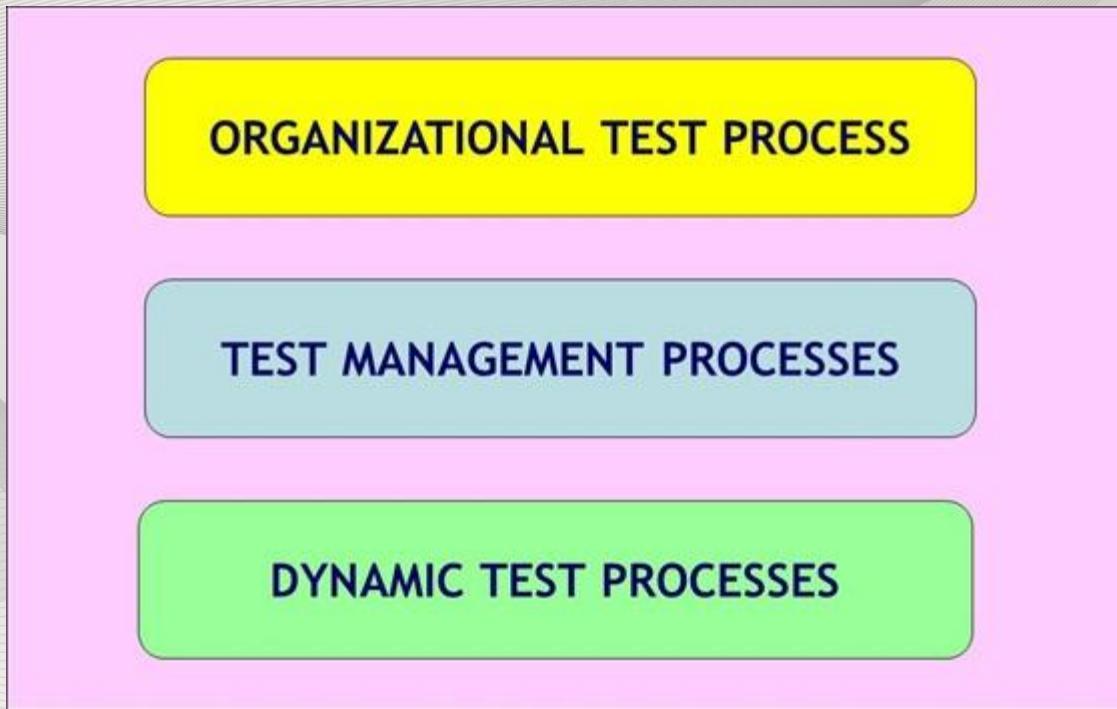
## ISO/IEC/IEEE 29119 Software Testing

- Parte 1: Conceptos y vocabulario
  - Conceptos generales (Ciclo de vida de las pruebas, objetivos, tipos de pruebas, etc.)
  - Conformidad
  - Implicaciones en diferentes ciclos de vida (secuencial, evolutivo, ágil)
  - Roles y Responsabilidades
  - Vocabulario
- Parte 2: Procesos (Versión revisada de Septiembre 2009)
  - Proceso genérico para políticas y estrategias (Organizational Test Process)
  - Proceso genérico para gestión (Test Management Processes)
  - Diseño y Ejecución (Fundamental Test Processes)
- Parte 3: Documentación
  - Contenido + Plantillas
- Parte 4: Técnicas
  - Descripción + Ejemplos
  - Estáticas: revisiones, inspecciones, etc.
  - Dinámicas:
    - Especificación: PCE, AVL, Sintácticas, Casos Uso, Combinatorias...
    - Estructura: Condiciones... MC/DC, Flujo Datos...
    - Experiencia: Búsqueda Errores, Prueba Exploratoria

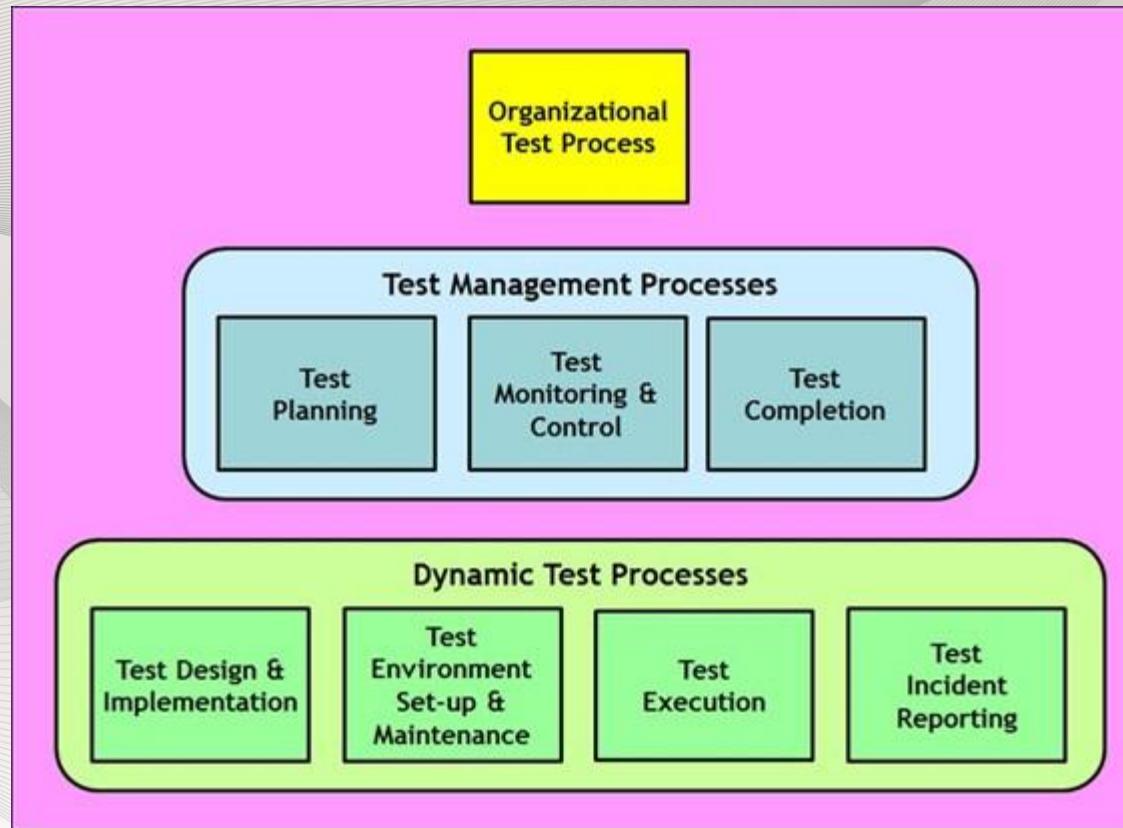
## ISO/IEC/IEEE 29119 Software Testing



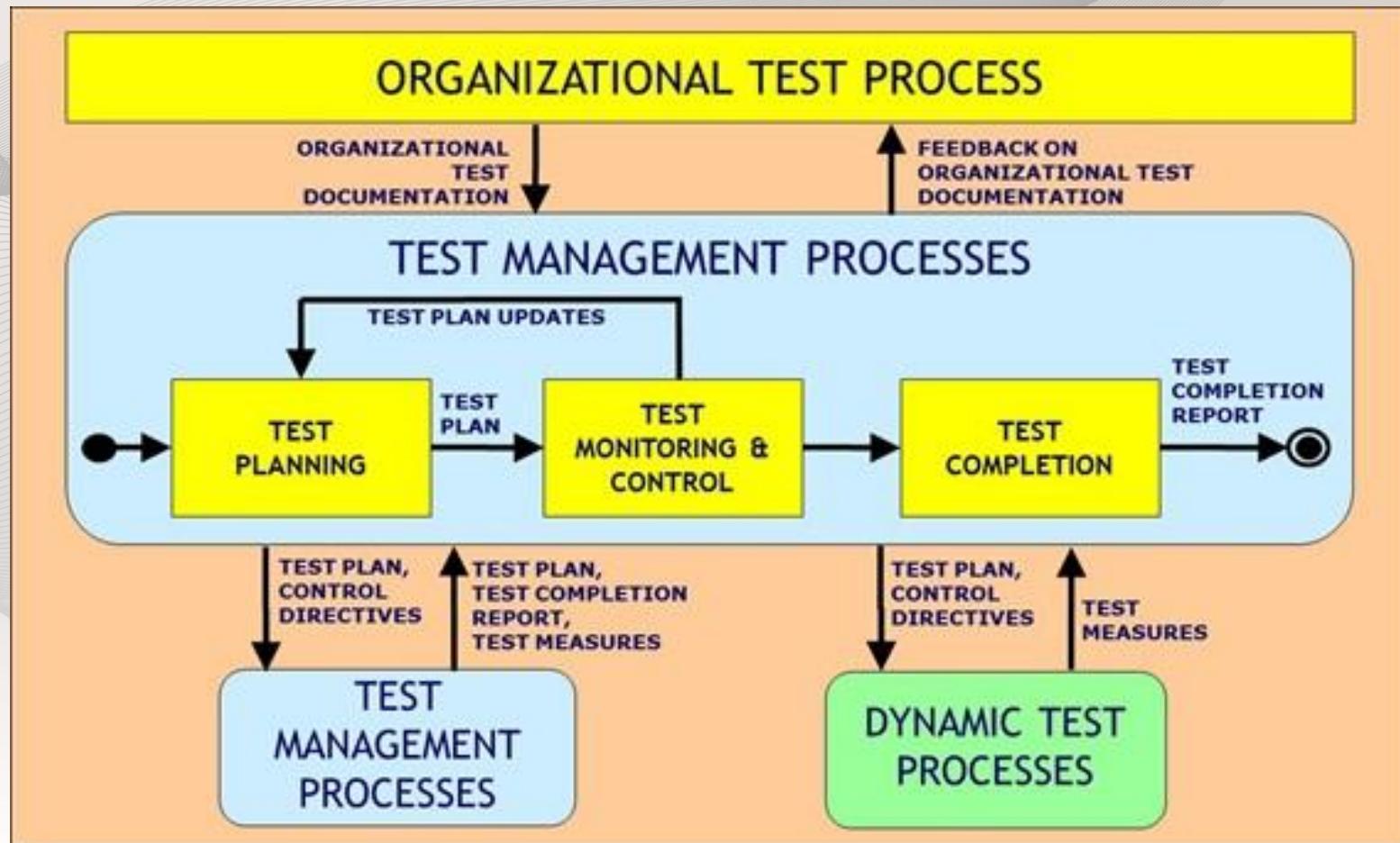
## The ISO/IEC/IEEE 29119-2 Test Process Model



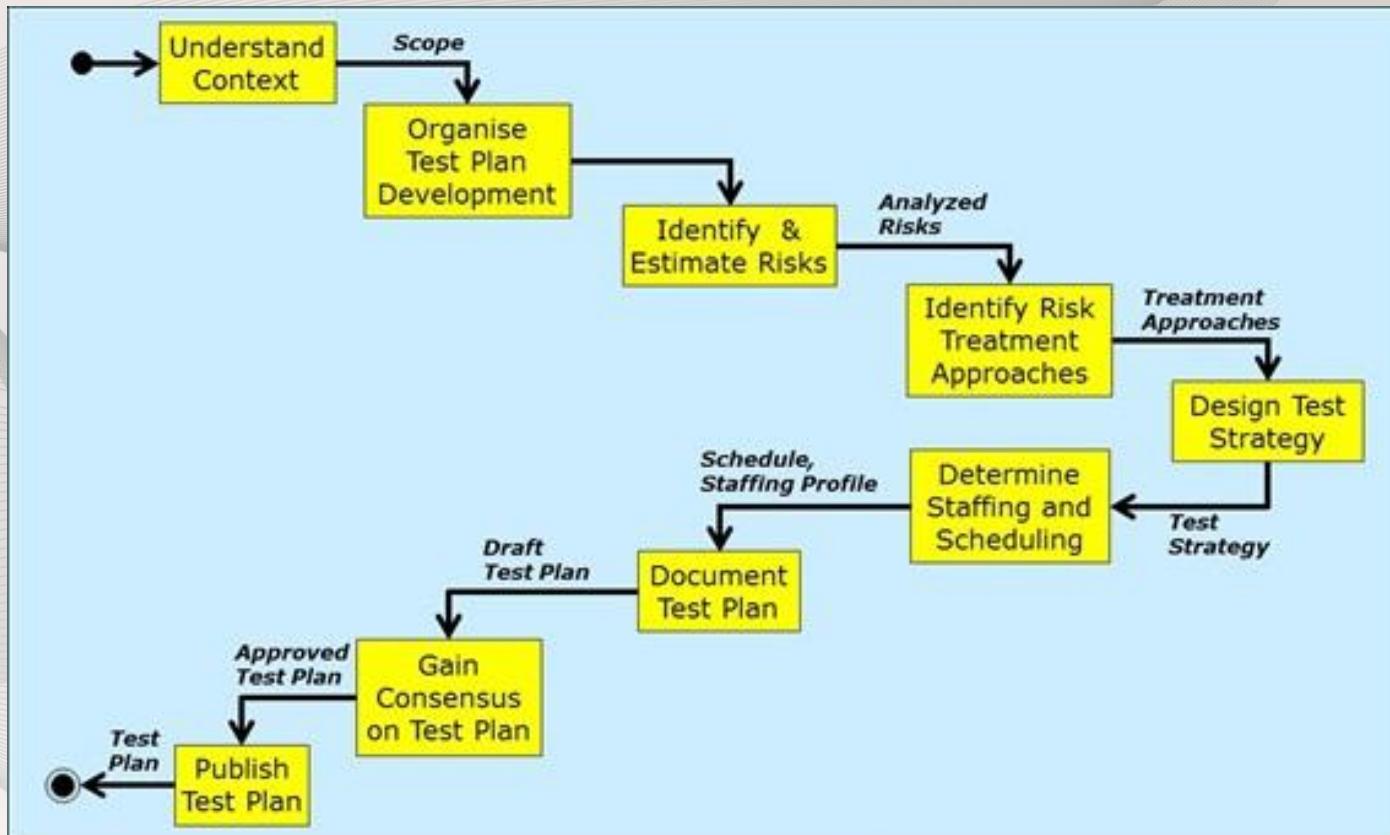
## Detailed view of the Test Process Model



## Test Management Processes

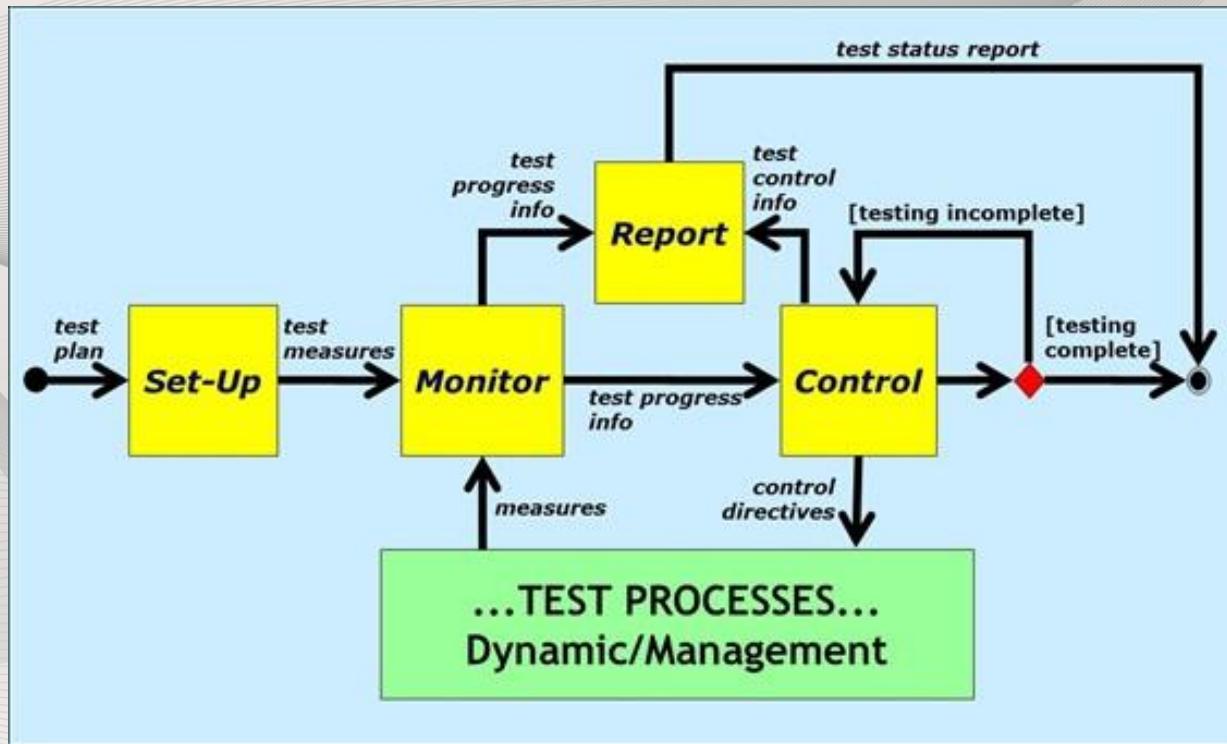


Test Planning Processes



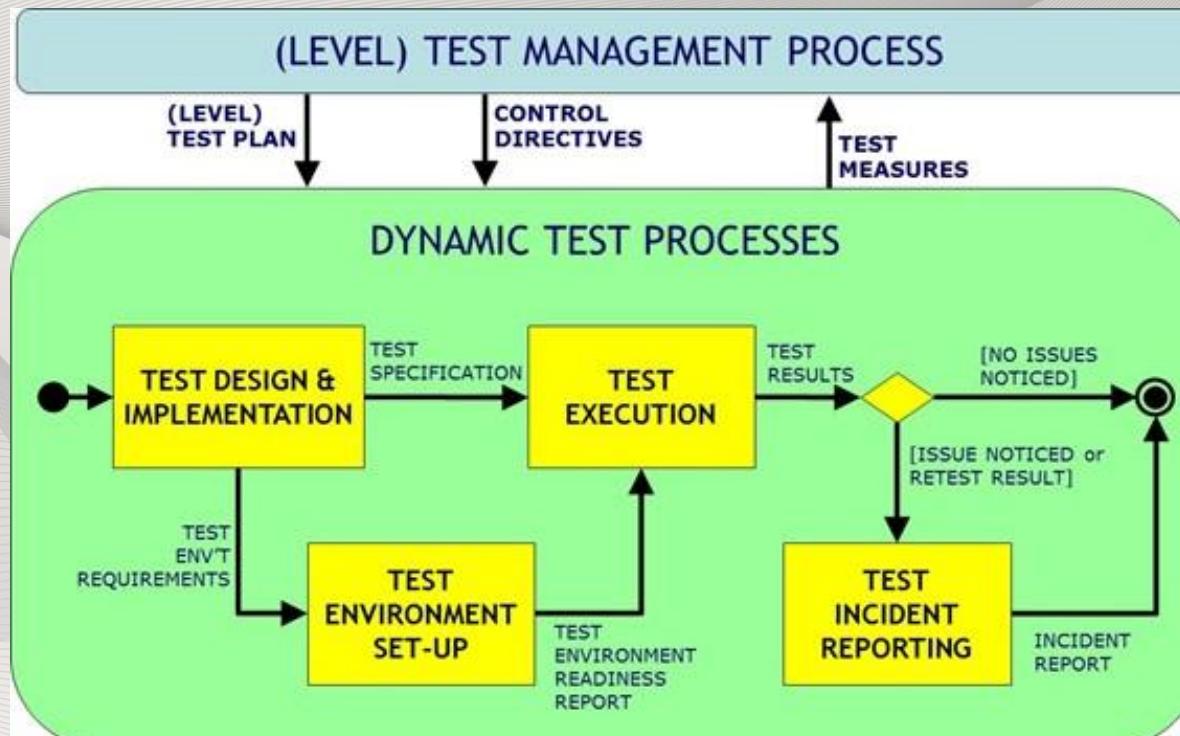
*The process is shown as purely sequential, but in practice it may be carried out iteratively, with some activities being revisited.  
See text for details.*

### Test Monitoring & Control Processes



*The process is shown as purely sequential, but in practice it may be carried out iteratively, with some activities being revisited.  
See text for details.*

## Dynamic Test Processes

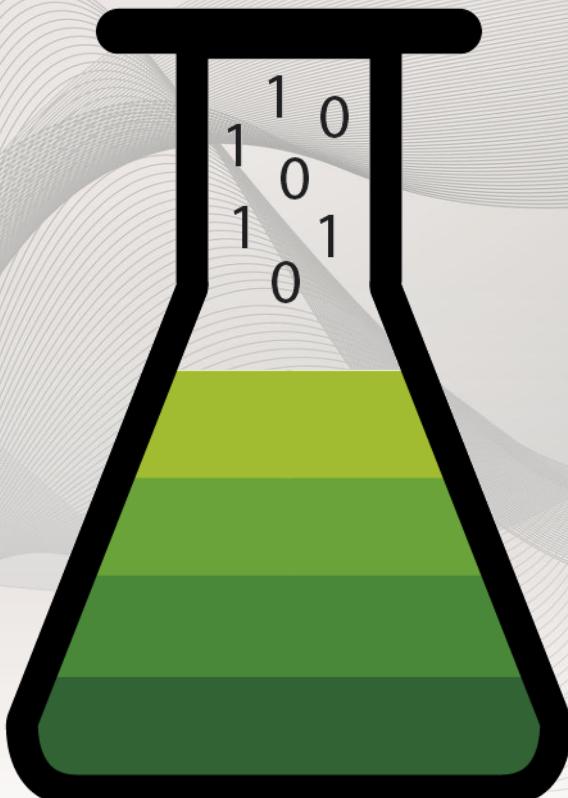


*The process is shown as purely sequential, but in practice it may be carried out iteratively, with some activities being revisited.  
See text for details.*

## Referencias

1. Binder, R., **Testing Object-Oriented Systems**, American Programmer, 7(4), 22-29, 1994.
2. Clemons RK, Project Estimation with Use Case Points Disponible en  
<http://www.stsc.hill.af.mil/CrossTalk/2006/02/0602Clemons.pdf>
3. Chidamber, S.R. y Kemerer, C.F., A metrics suite for object-oriented design ,**IEEE Trans. Software Engineering**, 20(6), 476-493, 1994.
4. Churcher, N.I. and Shepperd, M.J., Towards Conceptual Framework for Object-Oriented Metrics, **ACM Software Engineering Notes**, 20 (2), 67-76, 1995.
5. Dolado, J.J. y Fernández, L. (coordinadores). "Medición para la Gestión en la Ingeniería del Software". Ra-ma, 2000.
6. Fenton, N.E. y Pfleeger, S.L., **Software metrics. A rigorous & practical approach** , 1997.
7. Fenton, N.E. Y Kitchenham B., **Validating Software Meaures**, Journal of Software Testing, Verification and Reliability 1(2): 27-42, 1991
8. Genero M., Piattini M., Calero C. (coordinadores), **Metrics for Software Conceptual Models**, Imperial College Press, 2005.
9. IEEE Software Engineering Standards,. Standard 610.12-1990, 1993.
10. Lorenz, M. and Kidd, J., **Object\_oriented Software Metrics**, Prentice Hall 1994.
11. McConnell, S., **Desarrollo y gestión de proyectos informáticos**, Mc Graw Hill 1997.
12. Putnam, Lawrence H and Myers W., **Five Core Metrics**, DH Publishing, 2003
13. Pressman, R.S., **Ingeniería del Software. Un enfoque práctico**, Mc Graw Hill, 2010.
14. Sánchez, S., Sicilia, M.A., Rodríguez, D. **Ingeniería del Software. Un enfoque desde la guía SWEBOk**. IBERGACETA PUBLICACIONES, 2011.
15. Wohlin C. Et al. **Experimentation in Software Engineering: An Introduction**. Kluwer Academic Publisher, 2000
16. <http://www.javiergarzas.com/calidad-software>
17. <http://www.javiergarzas.com/2012/03/herramientas-de-calidad-software.html>
18. <http://www.javiergarzas.com/herramientas-software-recomendadas>
19. <http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/388>
20. <http://testlink.org/>
21. <http://demo.testlink.org>
22. <http://iso25000.com/>
23. JTC1/SC7: <http://www.jtc1-sc7.org>
24. Información ISO/IEC 29119: <http://www.softwaretestingstandard.org>
25. Red REPRIS <http://in2test.lsi.uniovi.es/repris/>
26. <http://www.softwaretestingstandard.org/>

0 1 1  
1 0 1 0  
0 1 1 0  
1 0 1 1



# GRACIAS

Dr. Julián Alberto García García  
*Julian.garcia@iwt2.org*

Dr. Francisco José Domínguez Mayo  
*fjdominguez@us.es*

**TESTEA**  
Learning by testing



[www.aragon.es/testea](http://www.aragon.es/testea)



[testea@itainnova.es](mailto:testea@itainnova.es)



Ingeniería Web y Testing Temprano  
Universidad de Sevilla