Detección temprana y prevención de errores en calidad de software

INTRODUCCIÓN

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

- ✓ Sistemas de 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.

¿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: falló software, el módulo de control no se había probado lo suficiente.

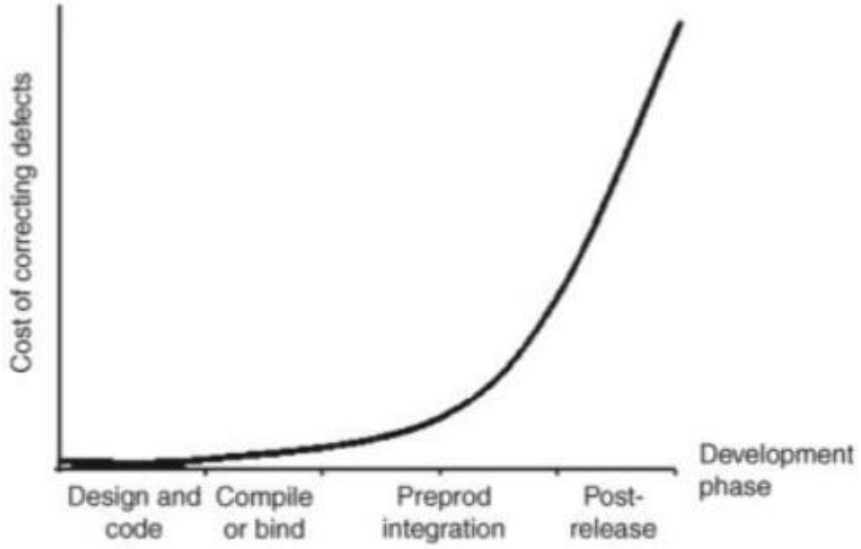
Veamos el siguiente video
https://www.youtube.com/watch?v=MGgsNzHOPQg

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.



Test (prueba)

(1) actividad por la cuál un ítem de prueba es evaluado y son creados reportes a partir de ello.
(2) grupo de actividades que conducen a facilitar el descubrimiento y la evaluación de propiedades de una o más pruebas. (verbo)

ISO/IEC/IEEE 29119
Software Testing

Test item (objeto de prueba)

Sistema, ítem de software, ítem, o work product (por ejemplo documento de requerimientos, diseño de especificaciones)

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
7743-5599	1. Descolgar auricular. 2. Marcar número de Juan 3. Esperar contestación.	(Juan): "Aló".

¿Funciona la suma?

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

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

Técnicas

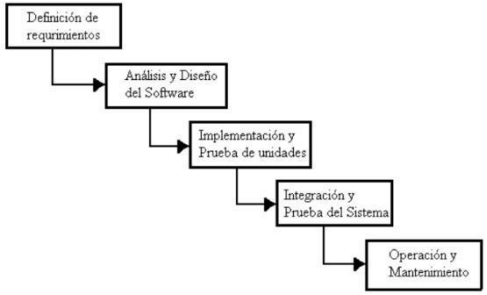
Valores de prueba	Acciones	Resultado esperado
0, 0	Suma(a, b)	0
0, b = no 0	Suma(a, b)	b
3, 4	Suma(a, b)	7
-2, -8	Suma(a, b)	-10
-3, 6	Suma(a, b)	3
Integer.MAX_VALUE, 6	Suma(a, b)	-2147483643

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

Niveles de Testing: la calidad en cada etapa del ciclo de vida

- **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).

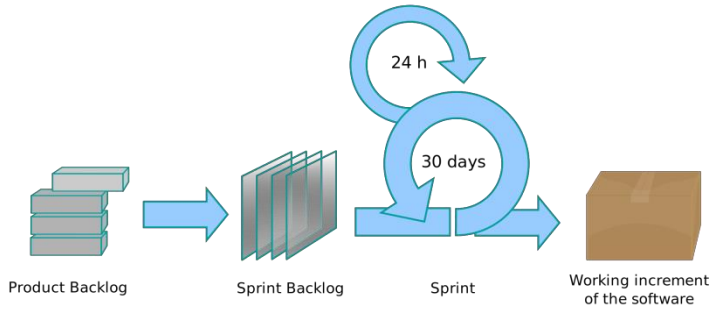
Modelo en cascada (clásico)



Modelo iterativo (RUP, XP, etc.)

Modelo evolutivo

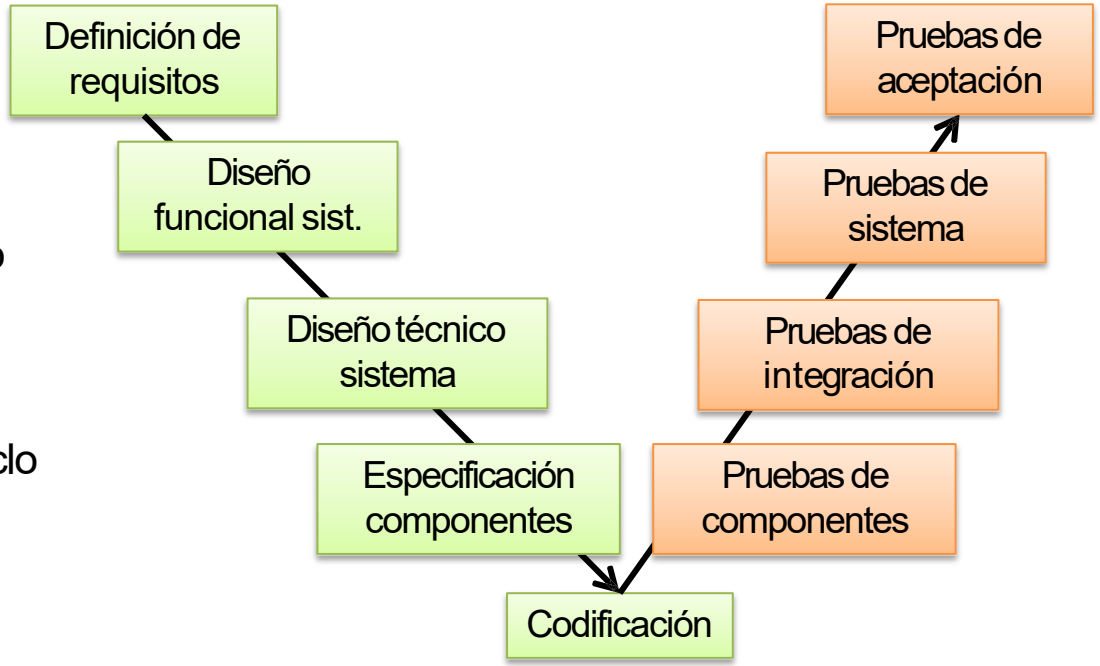
Modelos ágiles (SCRUM)



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.

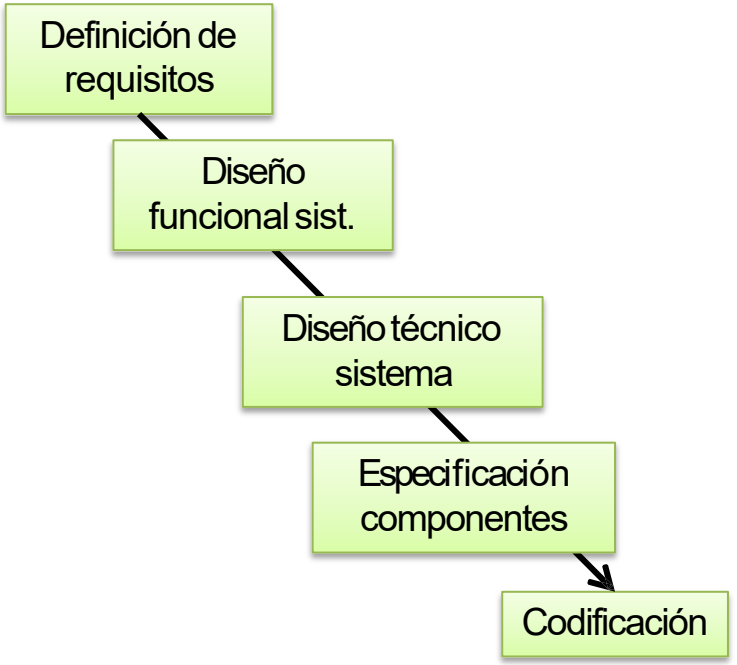
- Las pruebas (rama derecha) son diseñadas en paralelo al desarrollo (rama izquierda).
- Las actividades del proceso de pruebas tiene lugar a través del ciclo de vida software completo



Pruebas a través del ciclo de vida: Modelo-V

- **Rama de desarrollo software**

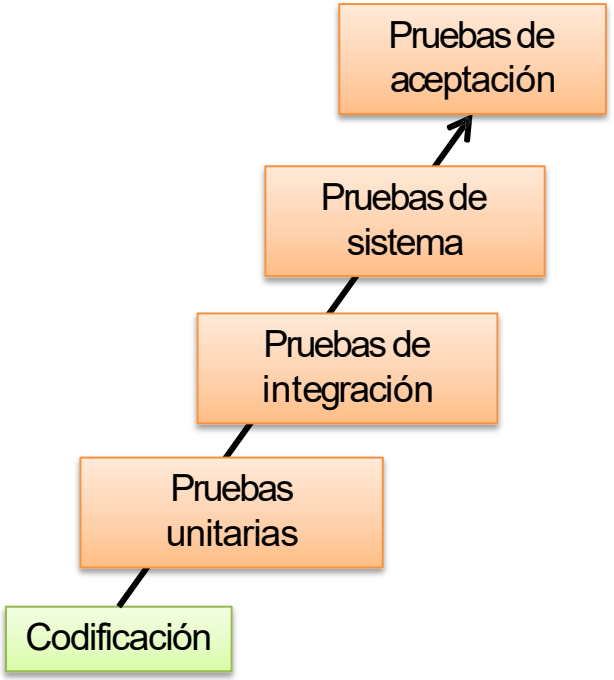
- ❑ Definición de requisitos
  - Documentos de especificación
- ❑ Diseño funcional del sistema
  - Diseño del flujo funcional del programa
- ❑ Diseño técnico del sistema
  - Definición de arquitectura/interfaces
- ❑ Especificación de los componentes
  - Estructura de los componentes
- ❑ Programación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



Pruebas a través del ciclo de vida: Modelo-V

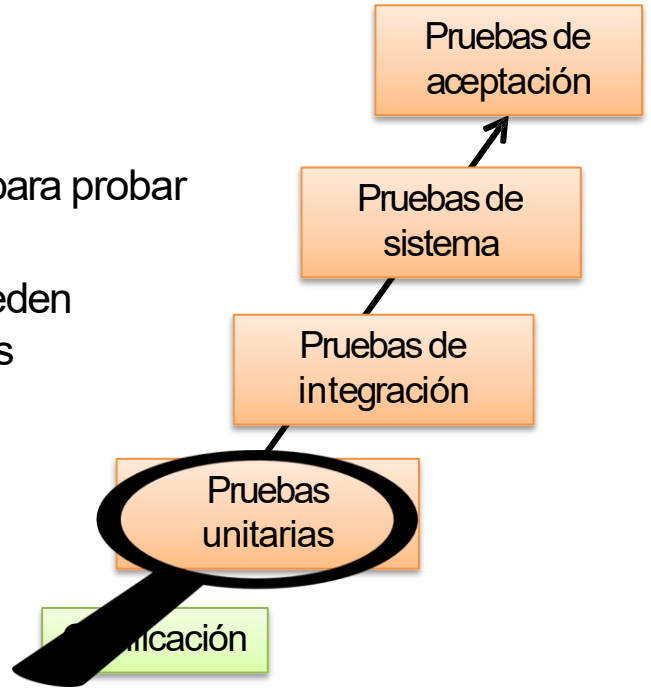
- **Rama de pruebas software**

- ❑ 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.)



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 podrá 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.



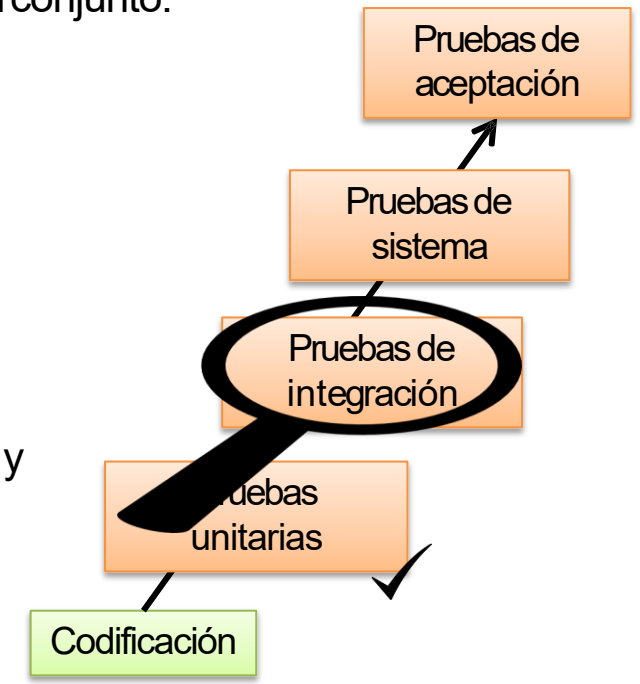
PRUEBAS DE INTEGRACIÓN: Alcance

- ✓ 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 podrá ser obtenidos a partir de:
  - Especificación de interfaces, diseño de la arquitectura y modelo datos



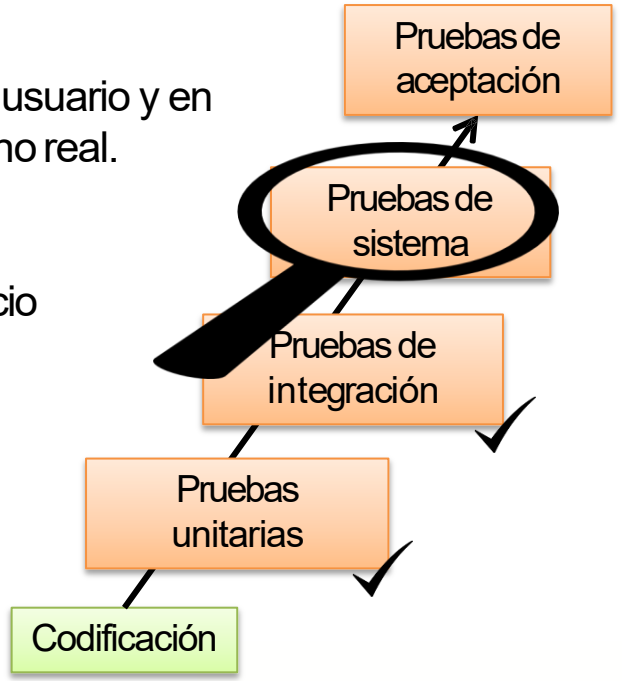
PRUEBAS DE SISTEMA: Alcance

- ✓ 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.

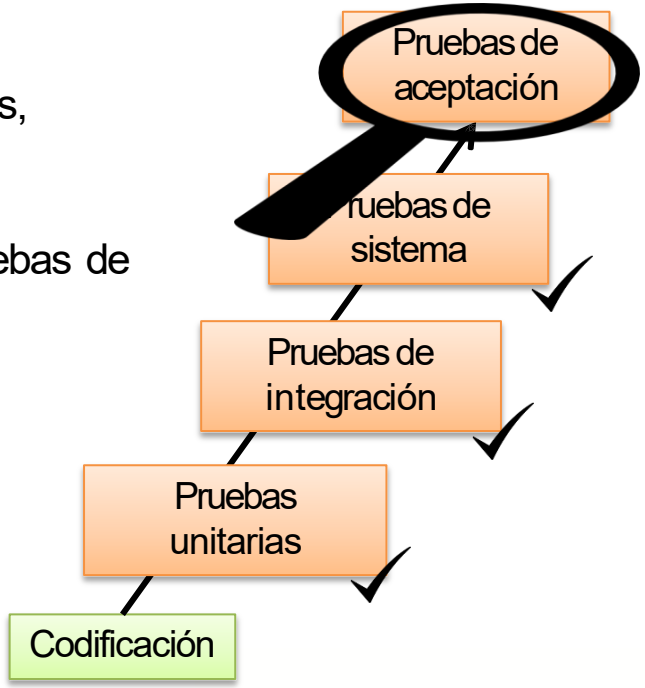
- ✓ Los casos de prueba podrá ser obtenidos a partir de:
  - Especificaciones funcionales, casos uso, procesos negocio

- ✓ Para la generación de casos de prueba se utilizan técnicas de caja negra

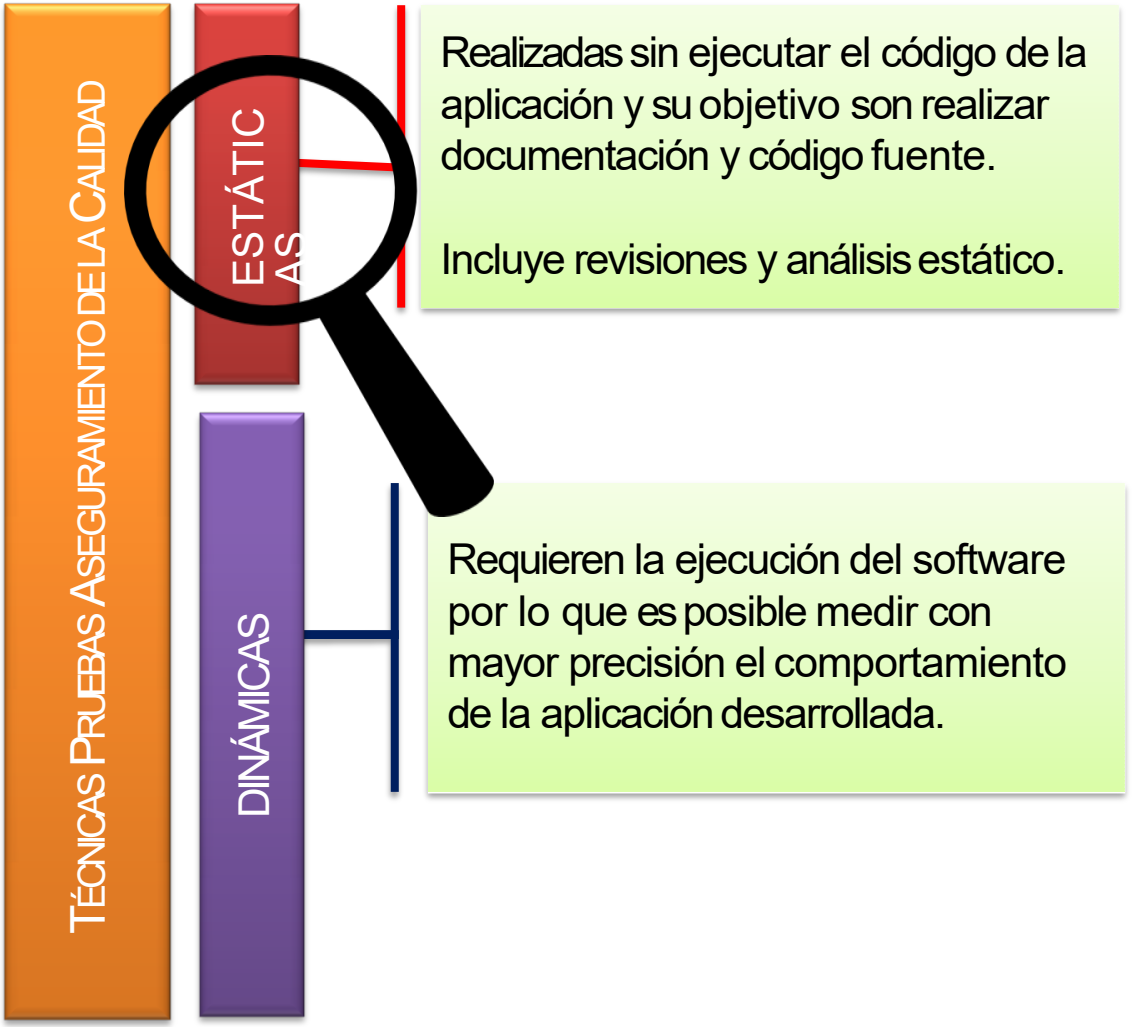


PRUEBAS DE ACEPTACIÓN: Alcance

- ✓ Son pruebas para aceptar formalmente el software. Son las pruebas de sistema del cliente/usuario.
- ✓ Esconveniente 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.

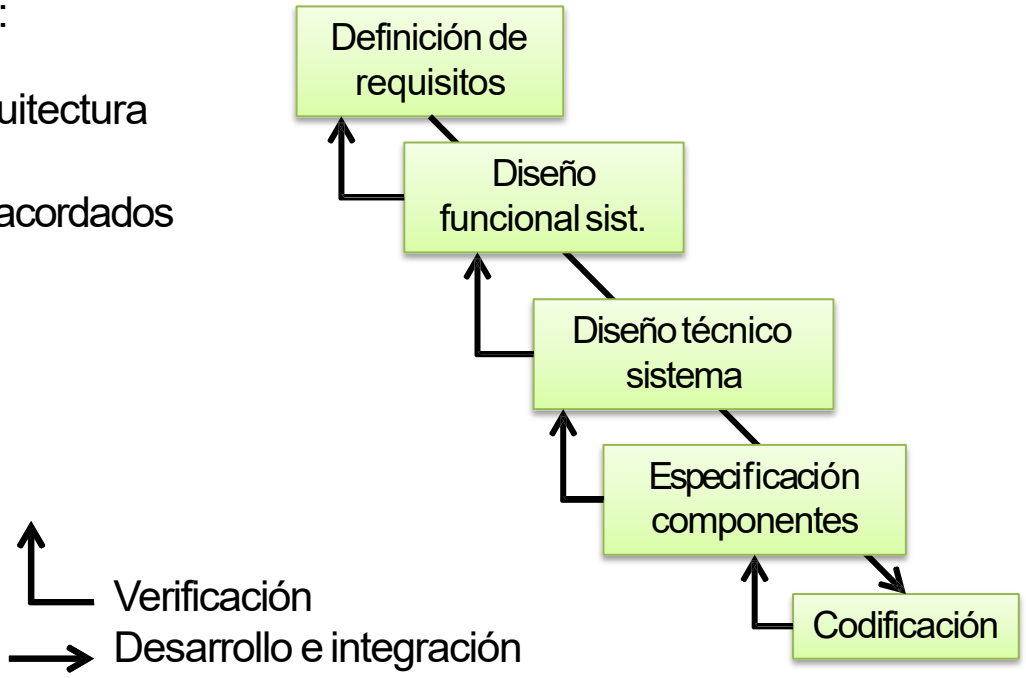


Técnicas de Testing: métodos para generar un caso de prueba



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 (por ejemplo. Código fuente, script, etc.) sin ejecutar el objeto de prueba.

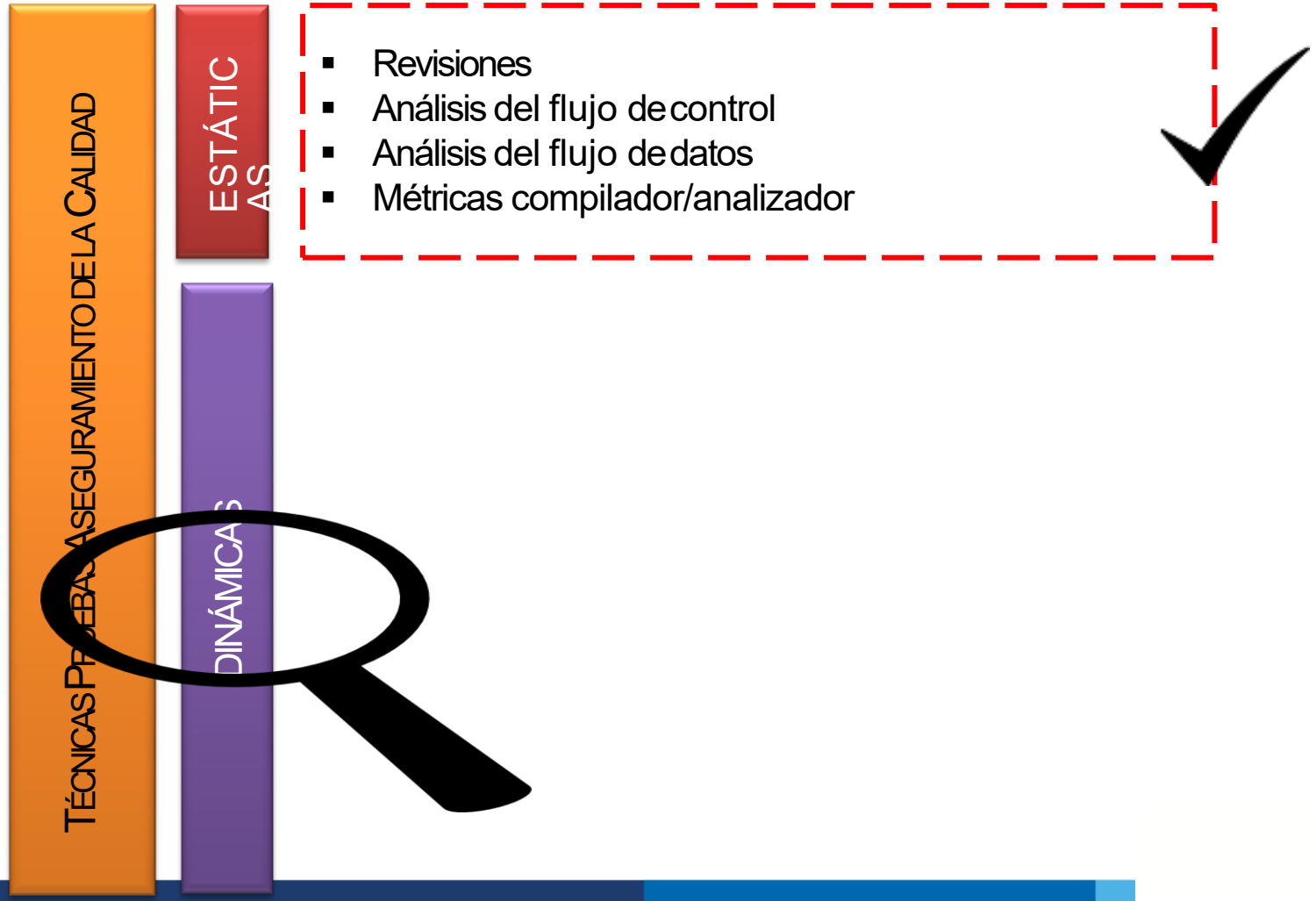
- ✓ Aspectos a detectar:

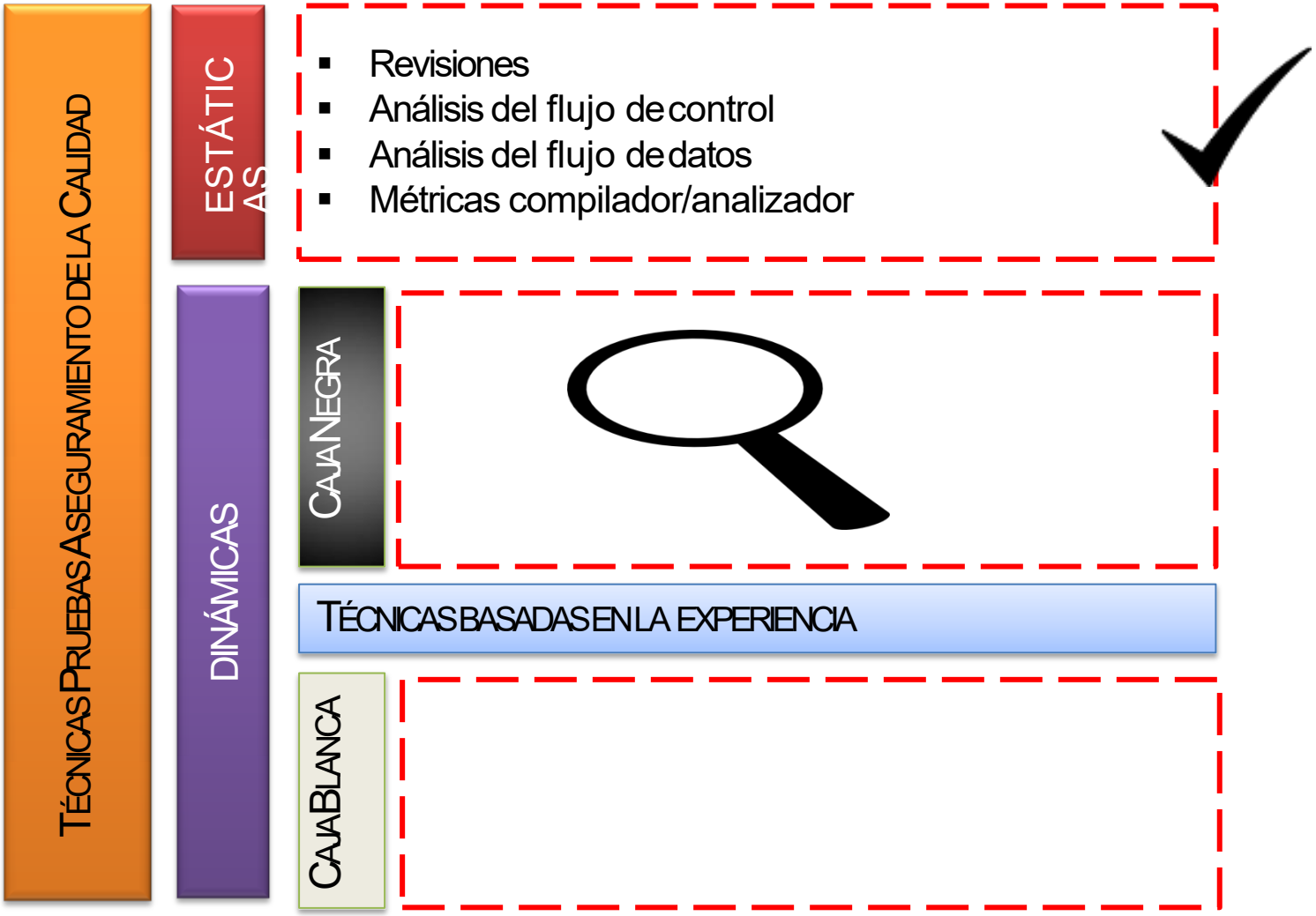
  - Reglas y estándares de programación
  - Diseño de un programa (análisis del flujo de control)
  - Uso de datos (análisis flujo de datos)
  - Complejidad de la estructura de un programa (métricas, e.g., nº ciclomático)

- ✓ Existen herramientas → compiladores, analizadores

  - Detectar lógica errónea (bucles potencialmente infinitos)
  - Detectar estructuras lógicas complejas, inconsistencias entre interfaces, etc.
  - Supone menor esfuerzo que una inspección

Técnicas de Testing: métodos para generar un caso de prueba




TÉCNICAS PRUEBAS ASEGURAMIENTO DE LA CALIDAD
ESTÁTICAS
DINÁMICAS
Revisiones
Análisis del flujo de control
Análisis del flujo de datos
Métricas compilador/analizador
CAJA NEGRA
TÉCNICAS BASADAS EN LA EXPERIENCIA
CAJA BLANCA

- ✓ El tester observar 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

- ✓ **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 <= X <= 100
   - 1ra CE no válida: X > 100
   - 2da CE no válida: X < 0
   - 3ra CE no válida: X = decimal
   - 4ta CE no válida: X = 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 más de 15 ni menos de 5 caracteres*

Condición	Descripción de los casos de prueba
Entre 5 y 15 caracteres	• 1 caso con nº de caracteres identificador = 15 • 1 caso con nº de caracteres identificador = 5 • 1 caso con nº de caracteres identificador = 16 • 1 caso con nº de caracteres identificador = 4

✓ 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**.

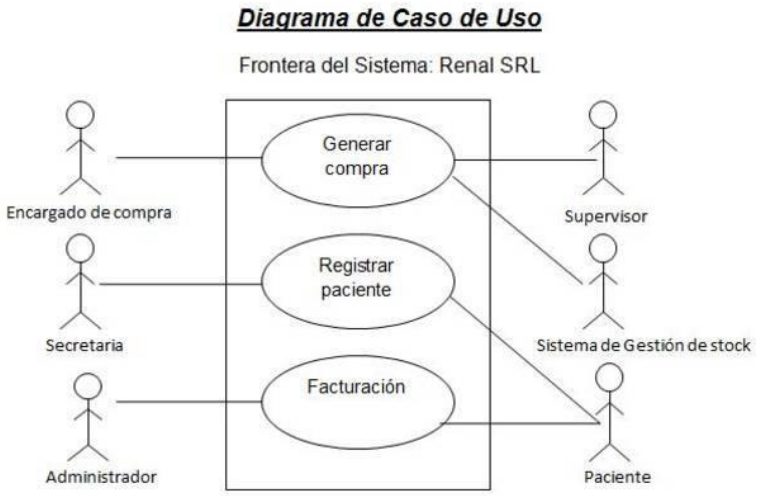
- Cada caso de uso describe una cierta interacción usuario-sistema. Elementos: precondiciones, pasos del comportamiento del sistema, post condiciones.

✓ Beneficios

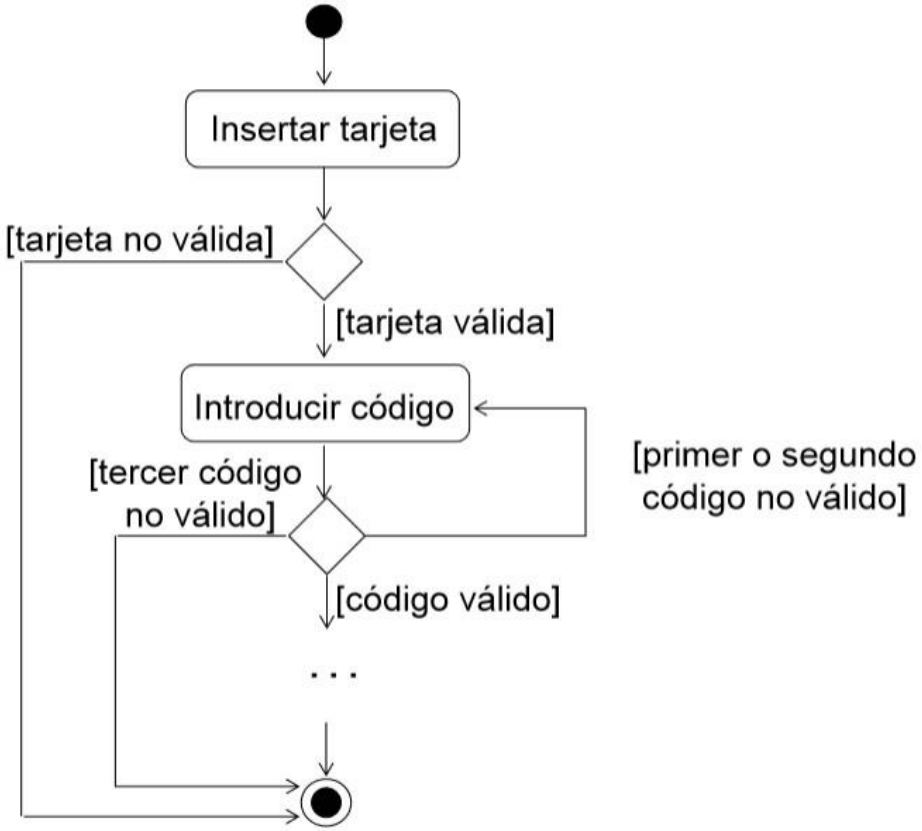
- Pruebas apropiadas para pruebas de aceptación y de sistema.
- Útil para diseñar pruebas con el cliente/usuario
- Puede ser combinadas con otras técnicas basadas en la especificación

✓ 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



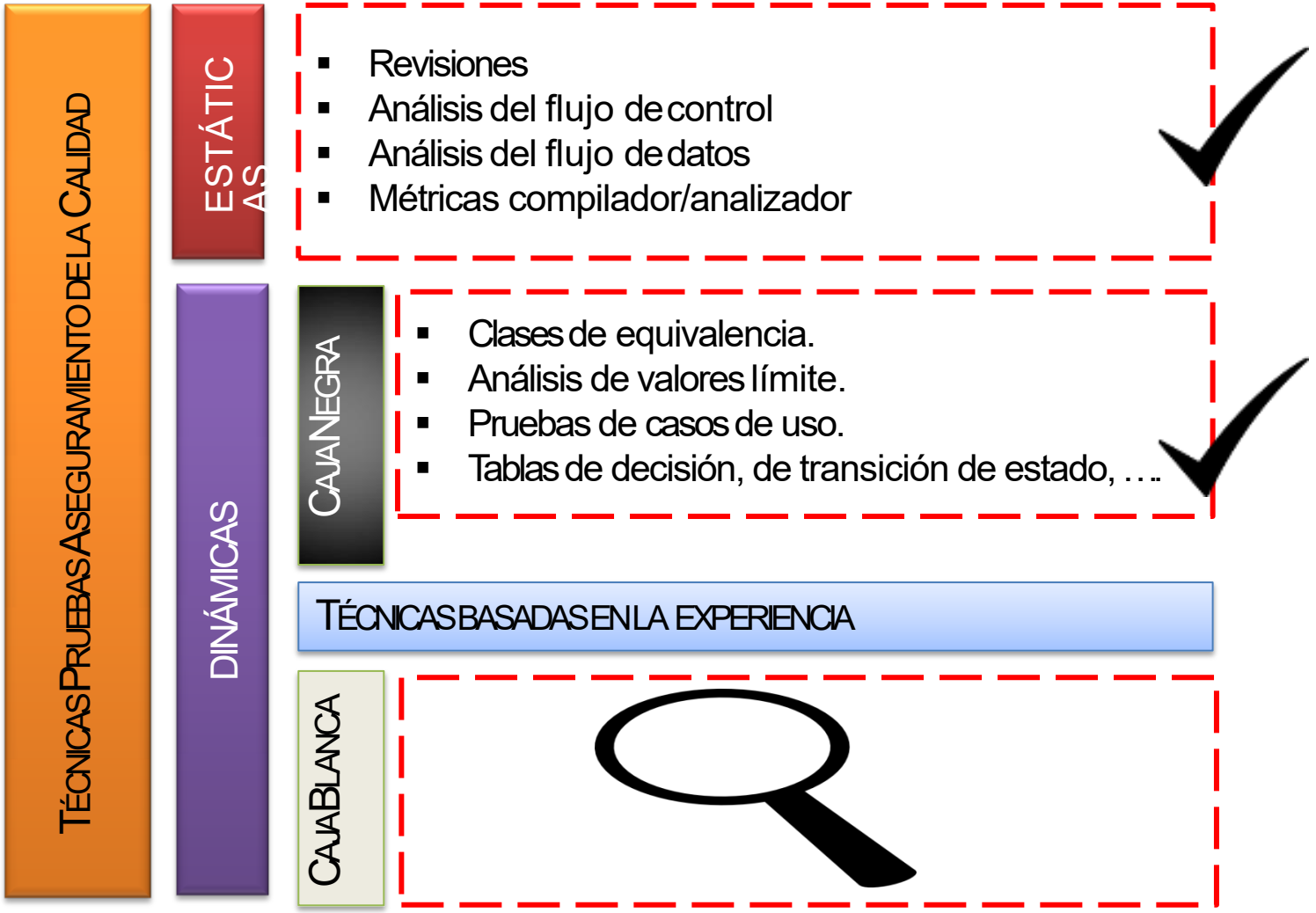
Prueba 1. Insertar tarjeta (no válida); fin

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

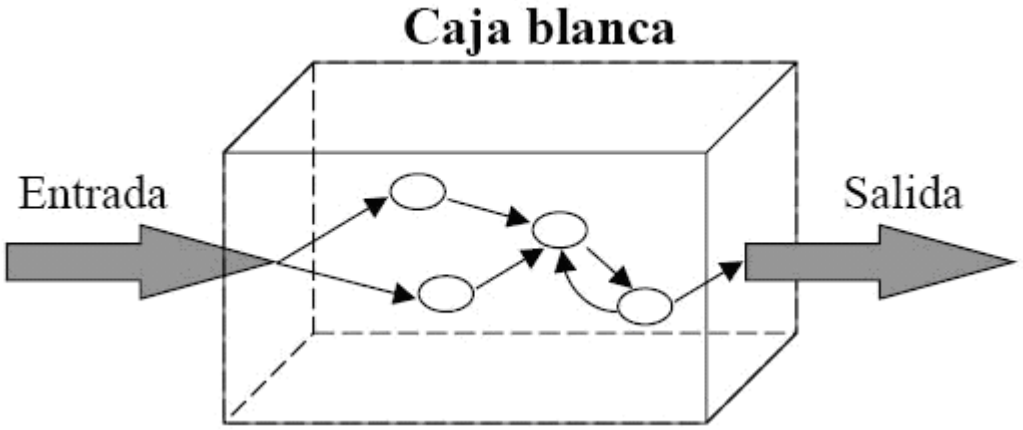
Prueba 3. Insertar tarjeta (válida); introduce código (válido);…; fin

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

Prueba 5. …


TÉCNICAS PRUEBAS ASEGURAMIENTO DE LA CALIDAD
ESTÁTICAS
Revisiones
Análisis del flujo de control
Análisis del flujo de datos
Métricas compilador/analizador
DINÁMICAS
CAJA NEGRA
Clases de equivalencia.
Análisis de valores límite.
Pruebas de casos de uso.
Tablas de decisión, de transición de estado, ….
TÉCNICAS BASADAS EN LA EXPERIENCIA
CAJA BLANCA

- ✓ 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.



- ✓ 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.
  - …

CAJABLANCA | TÉCNICAS DINÁMICAS DE TESTING

✓ Método: Cobertura de Sentencias

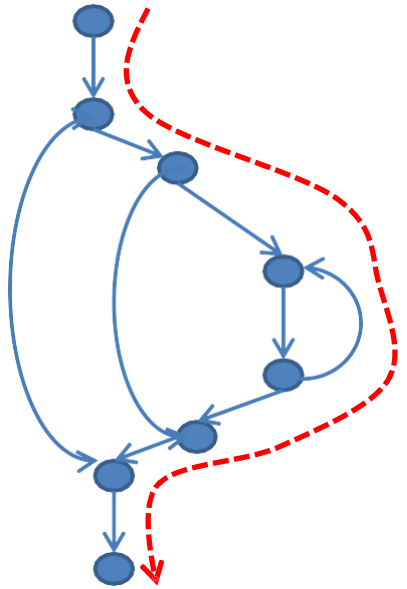
- Técnica basada en el análisis del gráfico del flujo de control (sentencias = nodos; flujo de control = aristas). El foco de atención es la sentencia del código !!
- **Objetivo**: lograr la cobertura de un porcentaje específico (C_0) de todas las sentencias. Cada sentencia se debe ejecutar, al menos,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

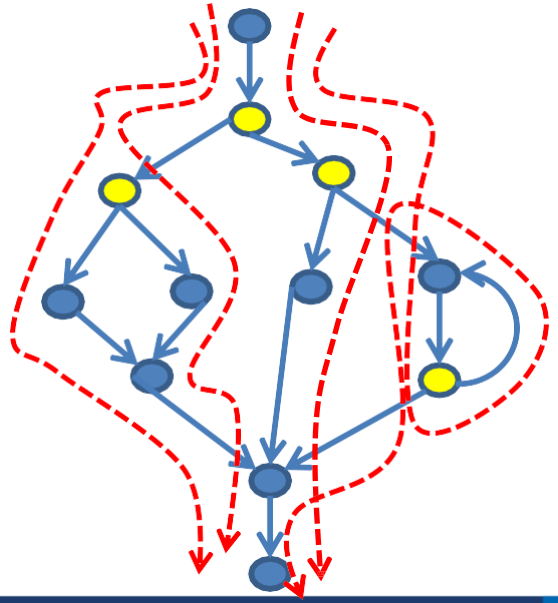
Un único caso prueba.



✓ 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% * (nº decisiones ejecutadas / 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.

CAJABLANCA | TÉCNICAS DINÁMICAS DE TESTING

✓ 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 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 OR B < 6. En los casos de prueba para la cobertura de condición múltiple podrían ser (por ejemplo):

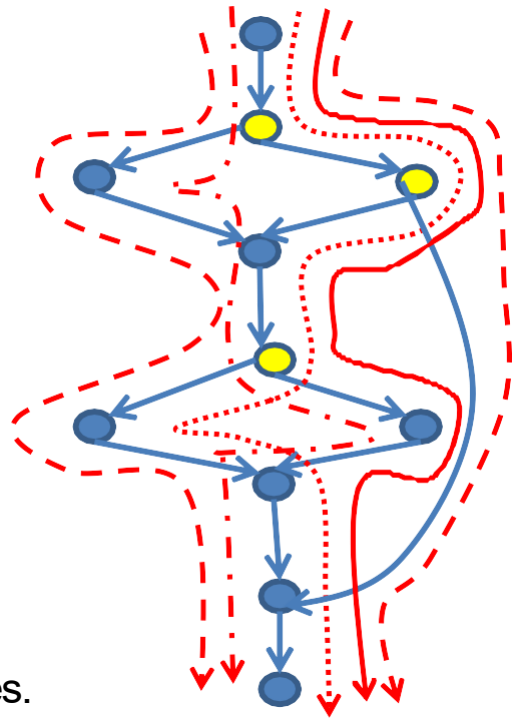
A = 6 (true)	B = 9 (false)	A > 2 OR B < 6 (true)
A = 6 (true)	B = 2 (true)	A>2 OR B< 6 (true)
A = 1 (false)	B = 2 (true)	A>2 OR B < 6 (true)
A = 1 (false)	B = 9 (false)	A>2 OR B <6 (false)

- 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

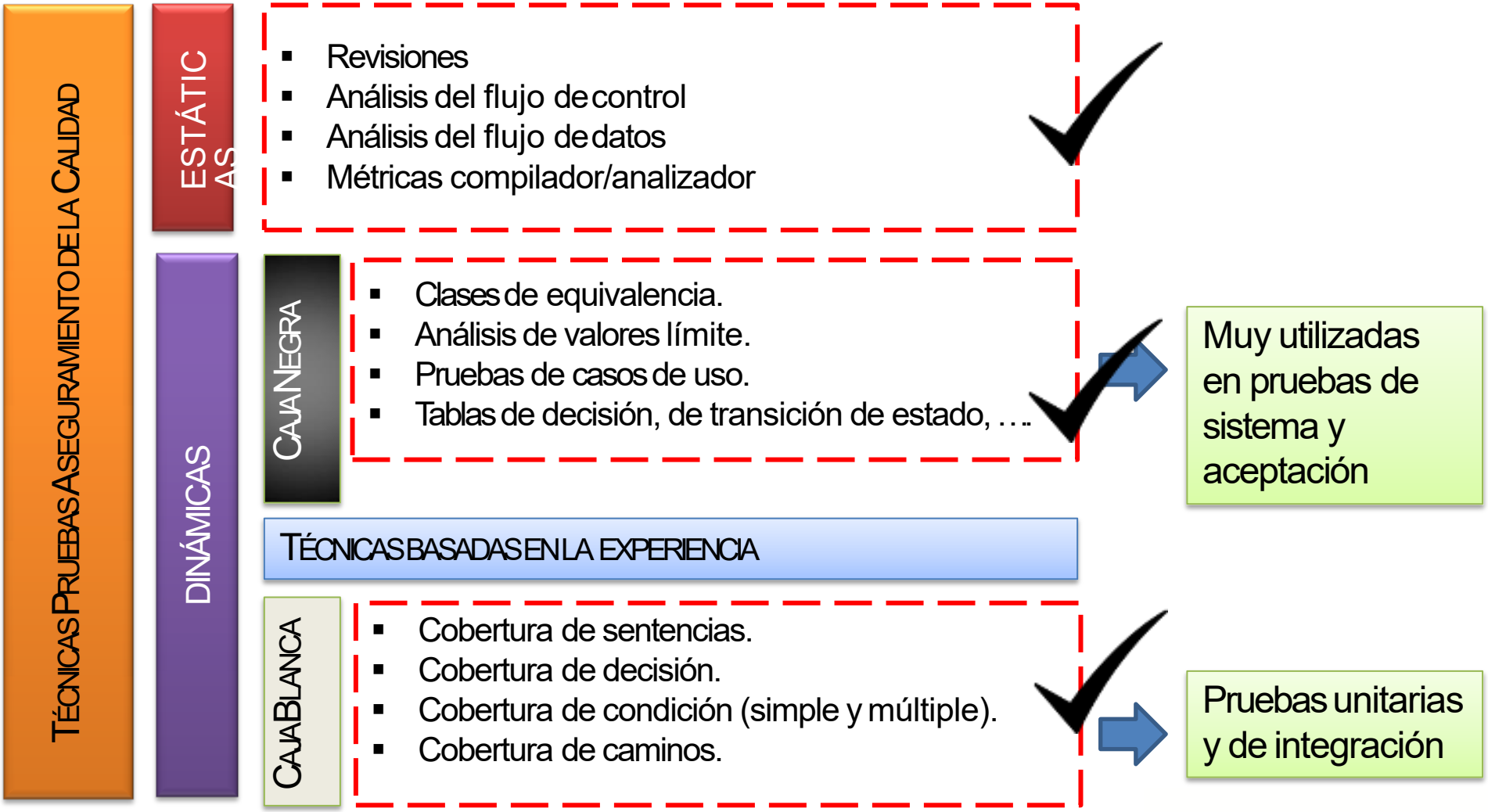
✓ Método: Cobertura de Camino

- 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% * (nº caminos cubiertos / nº total caminos)

- ✓ **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.


TÉCNICAS PRUEBAS ASEGURAMIENTO DE LA CALIDAD
ESTÁTICAS
DINÁMICAS
Revisiones
Análisis del flujo de control
Análisis del flujo de datos
Métricas compilador/analizador
CAJA NEGRA
Clases de equivalencia.
Análisis de valores límite.
Pruebas de casos de uso.
Tablas de decisión, de transición de estado, ….
Muy utilizadas en pruebas de sistema y aceptación
TÉCNICAS BASADAS EN LA EXPERIENCIA
CAJA BLANCA
Cobertura de sentencias.
Cobertura de decisión.
Cobertura de condición (simple y múltiple).
Cobertura de caminos.
Pruebas unitarias y de integración