

# Tema 7: Introducción a las Pruebas de Software

BLOQUE IV: PRUEBAS DE LOS SISTEMAS SOFTWARE

Ingeniería del Software

Grado en Ingeniería Informática

Curso 2024/2025



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Introducción

- Cohete Ariane 5 (1996)
  - Qué pasó: Error en la conversión de datos hizo que explotara 37 segundos tras el lanzamiento
  - Impacto: Pérdida de \$370 millones
  - Causa: Software heredado no adaptado a nuevas condiciones
- Mars Climate Orbiter (1999)
  - Qué pasó: Uso inconsistente de sistemas métrico e imperial
  - Impacto: Pérdida de \$125 millones y un proyecto clave de Marte
  - Causa: Incompatibilidad en las unidades de medida
- Toyota Prius (2005)
  - Qué pasó: Fallo en el software apagaba el vehículo en movimiento
  - Impacto: Retirada de 160.000 vehículos
  - Causa: Mal manejo de excepciones en el código
- AWS (2017)
  - Qué pasó: Comando erróneo causó caída masiva de servidores en EE.UU.
  - Impacto: Afectación a empresas como Netflix y Airbnb
  - Causa: Error humano sin validaciones adecuadas



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Técnicas de V & V

¿Qué es V&V (Verificación y Validación)?

- Procesos de análisis y pruebas para:
  - Asegurar que el software satisface su especificación
  - Entregar la funcionalidad esperada

**Verificación** → ¿Estamos construyendo el producto correctamente?

- Asegurar que cumple con los requisitos funcionales y no funcionales
- Confirmar que la implementación es técnicamente correcta y se ajusta al diseño

**Validación** → ¿Estamos construyendo el producto correcto?

- Evaluar si satisface las necesidades del cliente
- Asegurar que es útil en su contexto operativo



# Técnicas de V & V

## Objetivos de V & V

- Mejorar la calidad: Asegurar que el software cumpla con los estándares esperados
- Detectar y corregir errores: Identificar fallos a tiempo para optimizar el sistema
- Garantizar características clave: Consistencia, confiabilidad, utilidad y eficacia
- Asegurar utilidad: Confirmar que el software funcione en su entorno de trabajo
- Establecer confianza: Basada en:
  - Propósito del sistema (crítico o no crítico)
  - Expectativas del usuario
  - Condiciones del mercado



# Técnicas de V & V

## Inspecciones del Software

- Son técnica estática de validación para revisar artefactos del desarrollo como:
  - Diagramas de requisitos
  - Diseño
  - Código fuente
- Su objetivo es detectar errores, omisiones o anomalías
- Características:
  - No requieren ejecución del sistema
  - Aplicables en todo el ciclo de desarrollo
  - Económicas y efectivas: Detectan más del 60% de fallos con menor costo que las pruebas dinámicas
- Limitaciones:
  - No garantizan que el software sea operativo
  - No evalúan fiabilidad ni aspectos no funcionales
  - Detectan fallos a nivel de módulos, no de sistema
- El SEI considera las inspecciones como una práctica esencial para mejorar procesos de software



# Técnicas de V & V

## Pruebas del Software

- Son técnica de validación dinámica, en ellas se ejecuta el sistema o prototipos
- Su objetivo es detectar fallos y evaluar funcionalidad comparando el comportamiento real con el esperado.
- Tipos de pruebas según las técnicas de V & V:
  - Pruebas de validación:
    - Confirman que el software satisface los requisitos
    - Evalúan rendimiento y fiabilidad
  - Pruebas de defectos:
    - Detectan inconsistencias con la especificación
    - Revelan errores fuera del uso esperado del sistema

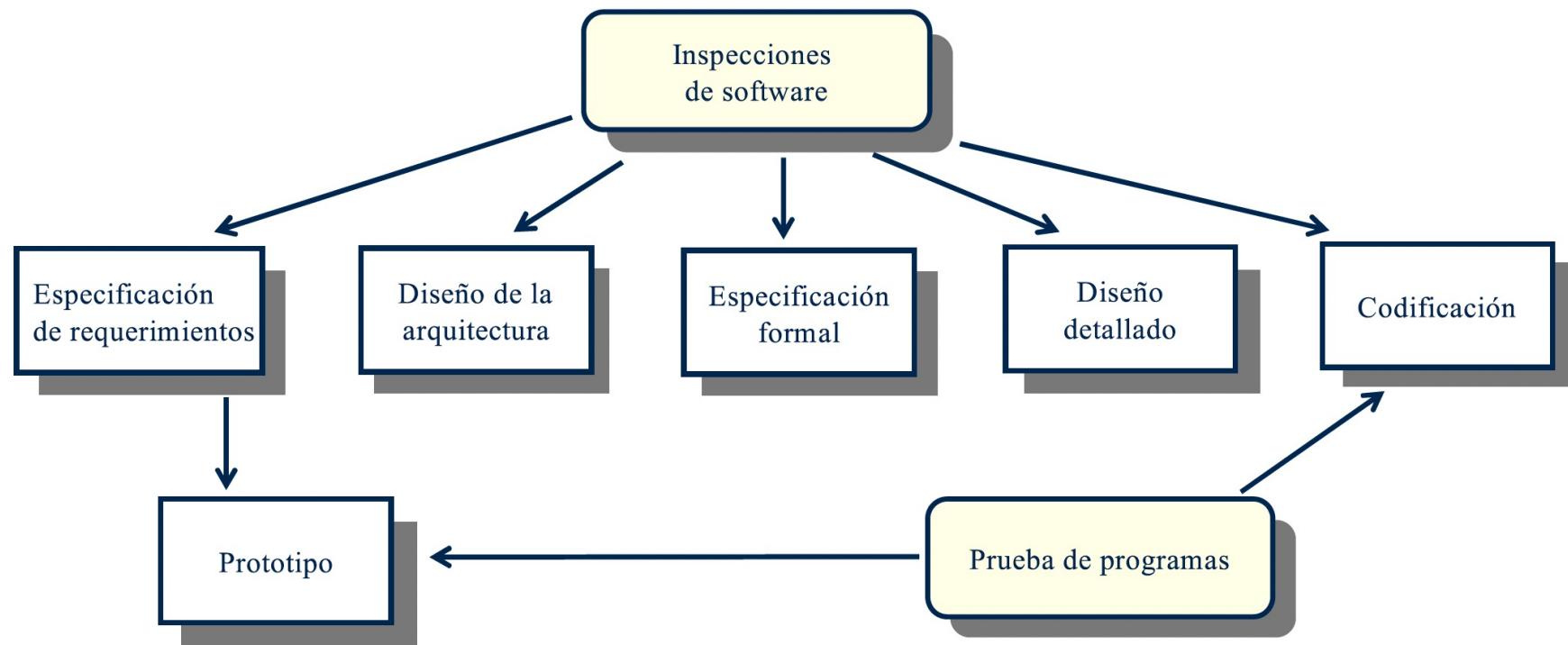


# Técnicas de V & V

- Pruebas: Principal técnica de validación
- Inspecciones: Detectan errores tempranos de forma económica
- Inspecciones y pruebas trabajan juntas para garantizar un software confiable
- Algunas metodologías usan la [programación por parejas](#) (*pair programming*, de *eXtreme Programming*), para:
  - Inspección continua del código
  - Mejora la calidad desde el inicio del desarrollo



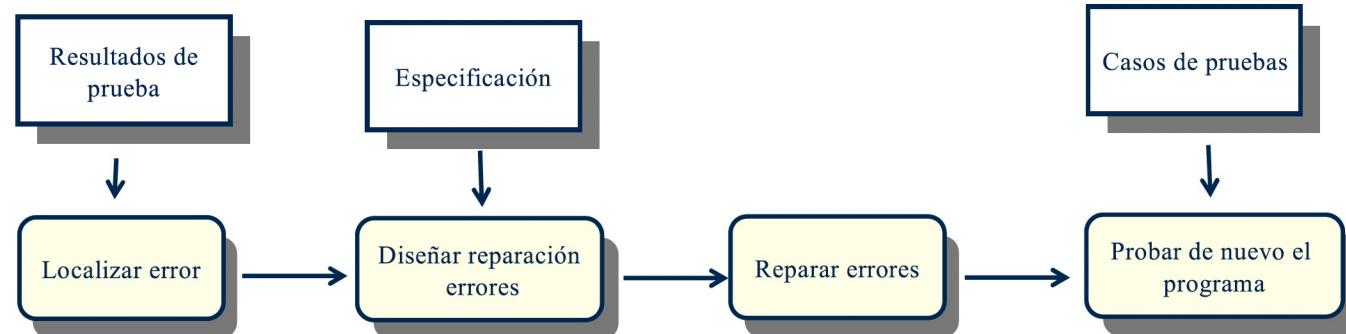
# Técnicas de V & V



# Técnicas de V & V

## Depuración del Software

- Es el proceso de localizar y corregir errores detectados en V&V
- Características:
  - Complejidad: Los errores pueden manifestarse lejos de su origen, dificultando su localización
  - Herramientas: Ayudan a analizar y solucionar problemas
  - Pruebas de regresión:
    - Verifican que las correcciones no generen nuevos fallos
    - Aseguran que errores relacionados también se resuelvan
- Es un ciclo iterativo en el que las correcciones y pruebas se repiten hasta lograr la estabilidad deseada



# Índice

1. Introducción
2. Técnicas de V & V
3. **Pruebas del Software**
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Pruebas del Software

- Las pruebas son actividades donde se ejecuta un sistema bajo condiciones específicas
- Se registran resultados para evaluar aspectos como:
  - Rendimiento
  - Usabilidad
  - Seguridad
- Verifican que el sistema:
  - Cumple los requisitos establecidos
  - Está libre de errores



# Pruebas del Software

Algunas de las definiciones que se utilizan en esta fase del ciclo de vida son según el estándar IEEE, 1990:

- **Pruebas:** Ejecución de un sistema bajo condiciones especificadas para observar y evaluar resultados
- **Caso de prueba:** Conjunto de entradas, condiciones y resultados esperados para un objetivo específico
- **Defecto:** Error en el software causado por una acción humana, como un proceso o paso incorrecto
- **Fallo:** Incapacidad de un sistema para cumplir funciones según los requisitos. Es un desvío del comportamiento esperado, visto por el usuario
- **Error:** Acción humana que lleva a un resultado incorrecto



# Pruebas del Software

## Objetivos de las Pruebas

- Imposibilidad de prueba exhaustiva: No se puede probar todo el comportamiento del software, ni en programas simples
- Detectar defectos en el software; encontrar un error es un éxito de la prueba
- Mito: Un defecto implica que somos malos profesionales y que debemos sentirnos culpables → todo el mundo comete errores
- Encontrar un defecto significa un éxito para la mejora de la calidad



# Pruebas del Software

Principios clave de las pruebas:

- Definir resultados esperados: Cada caso de prueba debe especificar qué resultado se espera, para compararlo con el real
- Evitar auto-pruebas: El programador no debe probar su propio código, ya que puede estar sesgado
- Inspección cuidadosa: Revisar a fondo los resultados para detectar posibles defectos
- Cobertura de casos: Incluir datos de entrada válidos e inválidos en los casos de prueba
- Centrarse en:
  - Probar si el software no hace lo que debe hacer
  - Probar si el software hace lo que no debe hacer, es decir, si provoca efectos secundarios adversos



# Pruebas del Software

Principios clave de las pruebas:

- Recursos adecuados: No asumir que no hay defectos, siempre asignar suficientes recursos a las pruebas
- Encontrar múltiples defectos: Si se encuentra un error, es probable que haya más
- Creatividad en las pruebas: Las pruebas son tan creativas como el desarrollo del software.
- Planificación sistemática: Diseñar pruebas de manera organizada para detectar la mayor cantidad de defectos con el menor esfuerzo.

Las pruebas deben ser planificadas cuidadosamente para detectar defectos de manera eficiente y efectiva con el mínimo consumo de tiempo y esfuerzo



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
- 4. Tipos de Pruebas**
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Tipos de Pruebas

Funcionales	No Funcionales
Pruebas Unitarias Pruebas de Comportamiento Pruebas de Integración Pruebas del Sistema Pruebas de Aceptación Pruebas de Interfaz Pruebas Alfa y Beta	Pruebas de Rendimiento Pruebas de Carga Pruebas de Estrés Pruebas de Estabilidad Pruebas de Escalabilidad Pruebas de Seguridad Pruebas de Usabilidad Pruebas de Compatibilidad Pruebas de Localización e Internacionalización



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
- 4. Tipos de Pruebas**
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Tipos de Pruebas

## Pruebas Funcionales

- Se centran en verificar que el software cumpla con los requisitos funcionales (lo que el sistema **debe hacer**)
- Características:
  - Basadas en los requisitos del sistema
  - Evaluación de entradas, salidas, procesos y comportamiento general del software
  - Pruebas de caja negra (no dependen de cómo se implementó la funcionalidad)
- Las pruebas funcionales aseguran que el software cumpla con las expectativas definidas sin considerar su implementación interna



# Tipos de Pruebas

## Pruebas Funcionales

### Tipos de Pruebas Funcionales

- **Pruebas Unitarias:** Validan componentes o funciones individuales
- **Pruebas de Componentes:** Verifican módulos o unidades completas
- **Pruebas de Integración:** Aseguran que los módulos interactúan correctamente
- **Pruebas del Sistema:** Evalúan el sistema completo en su entorno
- **Pruebas de Aceptación (UAT):** Confirman que el sistema satisface al cliente o usuario final
- **Pruebas de Regresión:** Verifican que los cambios no afecten las funciones existentes
- **Pruebas de Interfaz de usuario (UI):** Evalúan la interacción con la interfaz gráfica
- **Pruebas Alfa y Beta:**
  - **Alfa:** Hechas internamente antes de lanzar el producto
  - **Beta:** Hechas por usuarios finales para identificar problemas adicionales



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
- 4. Tipos de Pruebas**
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Tipos de Pruebas

## Pruebas No Funcionales

- Evalúan aspectos no relacionados con funcionalidades específicas, sino con atributos de calidad del sistema, como el rendimiento, la seguridad y la usabilidad
- Comprueban **cómo** el sistema realiza sus funciones, no solo qué hace
- Contribuyen a mejorar la experiencia del usuario y la robustez del software
- Pueden utilizar enfoques de caja negra o caja blanca, dependiendo de los objetivos
- Las pruebas no funcionales complementan la funcionalidad del sistema, asegurando su desempeño y experiencia general



# Tipos de Pruebas

## Pruebas No Funcionales

### Tipos de Pruebas No Funcionales

- **Pruebas de Rendimiento:**
  - **Pruebas de Carga:** Comportamiento del sistema con un aumento de usuarios o transacciones
  - **Pruebas de Estrés:** Evaluación de límites del sistema bajo condiciones extremas
  - **Pruebas de Estabilidad:** Funcionamiento continuo durante largos períodos
  - **Pruebas de Escalabilidad:** Capacidad para manejar incrementos en la carga
- **Pruebas de Seguridad:**
  - Detectan vulnerabilidades y evalúan la capacidad de protección del sistema
  - Incluyen pruebas de autenticación, cifrado y simulación de ataques
- **Pruebas de Usabilidad:** Evalúan la facilidad de uso e intuitividad de la interfaz para el usuario final
- **Pruebas de Compatibilidad:** Verifican el funcionamiento correcto en diferentes dispositivos, sistemas operativos y navegadores
- **Pruebas de Localización e Internacionalización:** Aseguran que el software funcione correctamente en diferentes idiomas, regiones y formatos de datos



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Desarrollo Guiado por Pruebas (TDD)

- El desarrollo guiado por pruebas es una metodología que pone las pruebas en el centro del desarrollo de software
- Contrasta con enfoques tradicionales al priorizar la calidad desde el inicio:
  - **Metodología en cascada:** Pruebas al final del ciclo; detección tardía de errores y altos costos de corrección
  - **Metodologías ágiles:** Pruebas integradas en iteraciones, pero a menudo relegadas por limitaciones de tiempo/costo
- Intenta solucionar el problema con:
  - Pruebas antes del código: Sirven como especificaciones claras y guían el desarrollo
  - Código mínimo necesario: Evita complejidad innecesaria al implementar solo lo requerido para superar las pruebas



# Desarrollo Guiado por Pruebas (TDD)

- Ventajas de TDD:
  - Diseño enfocado en requisitos: Asegura el cumplimiento de los requerimientos funcionales
  - Mayor calidad del código: Reduce errores y facilita el mantenimiento
  - Documentación implícita: Las pruebas describen el comportamiento esperado del sistema
  - Confianza en cambios: Las pruebas automatizadas detectan regresiones
- Desarrollo guiado por el comportamiento, BDD (*Behavior-Driven Development*):
  - Describe el comportamiento del sistema en un lenguaje comprensible para desarrolladores y no técnicos
  - Utiliza escenarios como: “Dado-Cuando-Entonces”
  - Promueve mejor comunicación entre equipos técnicos y de negocio



# Desarrollo Guiado por Pruebas (TDD)

- Flujo de trabajo del TDD sigue un ciclo iterativo:
  - **Red** (Escribir una prueba que falle): Garantiza que la prueba es válida y la funcionalidad aún no existe
  - **Green** (Hacer que la prueba pase): Implementa el código mínimo necesario para superar la prueba
  - **Refactor** (Mejorar el código): Optimiza el código manteniendo el cumplimiento de las pruebas



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Estrategia de Aplicación de Pruebas

- Pretende integrar el diseño de los casos de prueba en pasos bien coordinados, creando niveles de prueba con objetivos específicos
- También permite la coordinación del personal del desarrollo, del departamento de garantía de la calidad y del cliente, definiendo los roles y su ejecución. Se seguirán las siguientes etapas:
  - Comenzar las pruebas a nivel de módulo
  - Continuar con la integración del sistema completo y su instalación
  - Finalizar con la aceptación del producto por parte del cliente

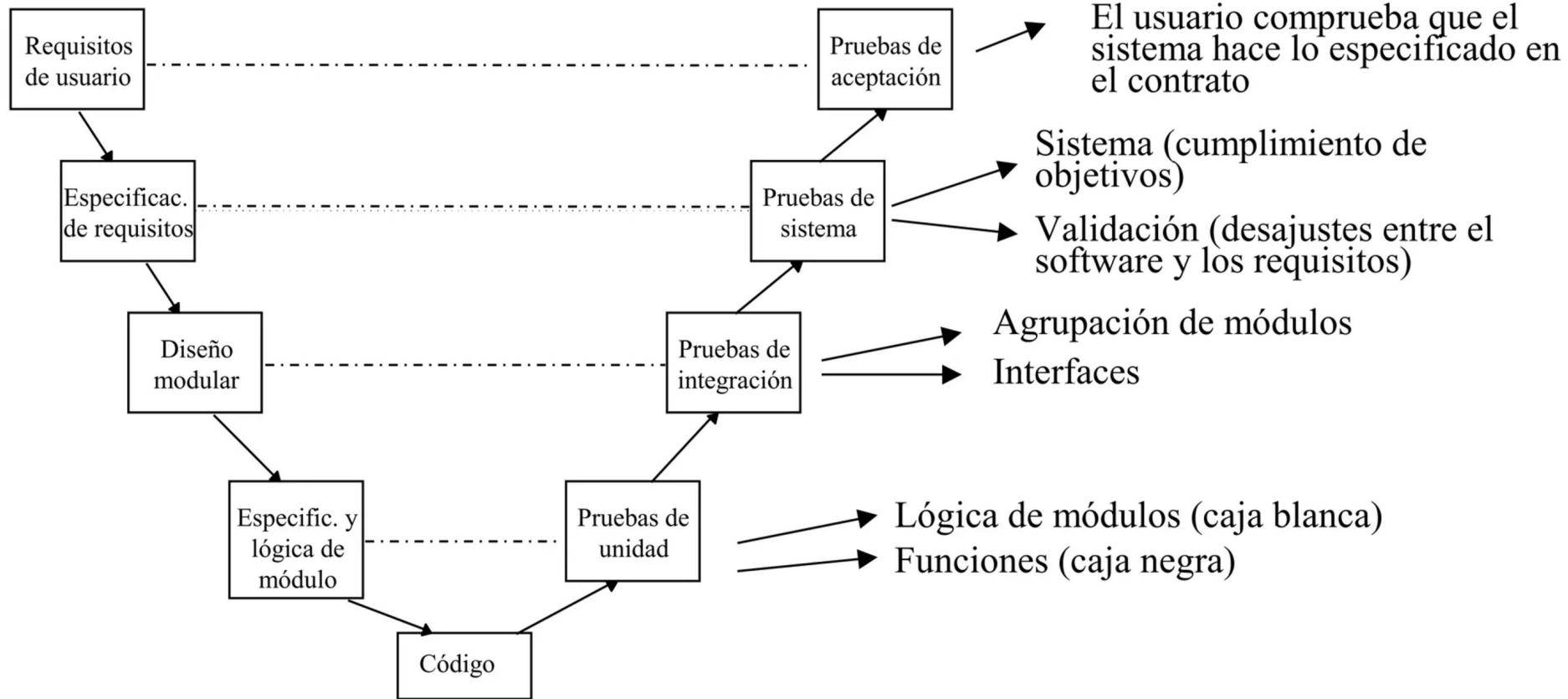


# Estrategia de Aplicación de Pruebas

- Se comienza en la prueba de cada módulo, que normalmente la realiza el propio personal de desarrollo en su entorno
- Con el esquema del diseño del software, los módulos probados se integran para comprobar sus interfaces en el trabajo conjunto (**prueba de integración**)
- El software totalmente ensamblado se prueba como un conjunto para comprobar si cumple o no tanto los requisitos funcionales como los requisitos de rendimiento, seguridad, etc. (**prueba funcional o de validación**)
- El software ya validado se integra con el resto del sistema (por ejemplo, elementos mecánicos, interfaces electrónicas, etc.) para probar su funcionamiento conjunto (**prueba del sistema**)
- Por último, el producto final se pasa a la prueba de aceptación para que el usuario compruebe en su propio entorno de explotación si lo acepta como está o no (**prueba de aceptación**)



# Estrategia de Aplicación de Pruebas



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Estrategia de Aplicación de Pruebas

## Prueba Unitaria

- Son pruebas formales que permiten declarar que un módulo está listo y terminado (no las informales que se realizan mientras se desarrollan los módulos)
- Las unidades de prueba pueden ser uno o más módulos del mismo programa
- Al menos uno de los módulos no ha sido probado previamente
- El conjunto de módulos es el objeto de una prueba
- La prueba de unidad puede abarcar desde un módulo hasta un grupo de módulos (incluso un programa completo)
- Estas pruebas suelen realizarlas el propio personal de desarrollo, pero evitando que sea el propio programador del módulo



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

- Están relacionadas con la integración de los componentes del software hasta obtener con el producto global que debe entregarse
- Implican una secuencia ordenada de pruebas que parte desde los componentes (módulos) y culmina en el sistema completo
- Su objetivo es la prueba de las interfaces (flujos de datos) entre componentes o módulos
- El orden de integración afecta a:
  - La forma de preparar casos
  - Las herramientas necesarias
  - El orden de codificar y probar los módulos
  - El coste de la depuración y preparación de casos



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

Tipos de integración:

- **Integración incremental:** Se añade un módulo a la vez al conjunto ya probado, aumentando progresivamente el número de módulos hasta formar el sistema completo
  - **Ascendente:** Comienza desde los módulos más bajos (hoja)
  - **Descendente:** Comienza desde el módulo principal (raíz) y a su vez pueden ser en profundidad y en anchura
- **Integración no incremental:** Se prueba cada módulo individualmente y luego todos se integran y prueban de una vez

Las pruebas unitarias e integración a menudo se realizan de manera simultánea



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Integración Incremental Ascendente

- Se agrupan los módulos de bajo nivel que realizan funciones específicas, reduciendo así los pasos de integración
- Se crea un **módulo impulsor (driver)** para cada grupo, que simula las llamadas a los módulos, introduce datos de prueba y recoge los resultados
- Se prueba cada grupo utilizando su impulsor
- Los impulsores se reemplazan por los módulos de nivel superior, y se crean nuevos impulsores hasta llegar al módulo raíz

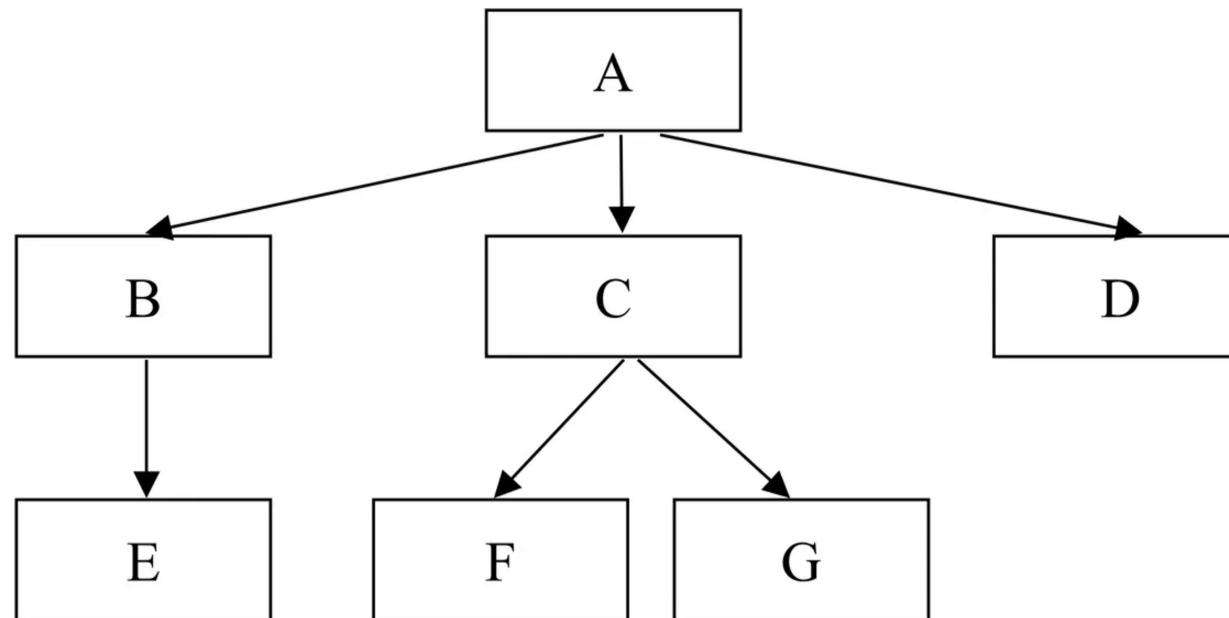


# Estrategia de Aplicación de Pruebas

## Prueba de Integración

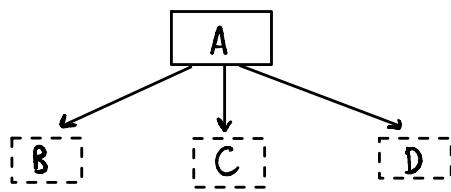
### Integración Incremental Ascendente

- Supongamos la siguiente estructura de programa:

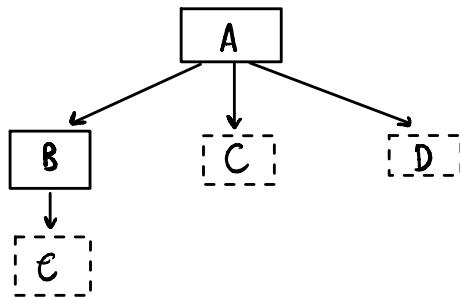


## Anchura

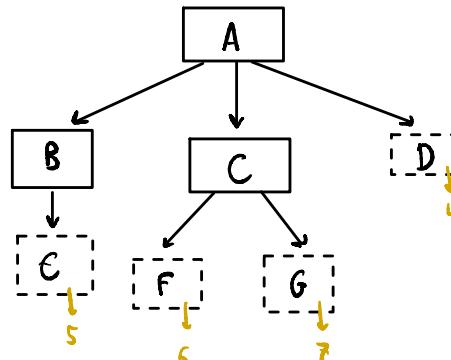
Fase 1



Fase 2



Fase 3

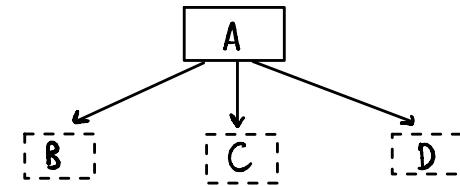


:

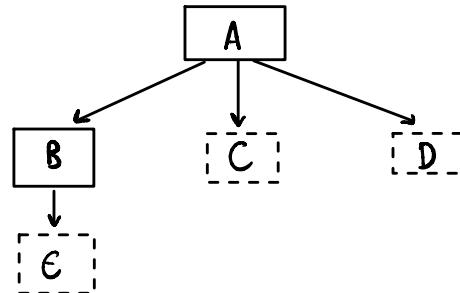
Fase 7

## Profundidad

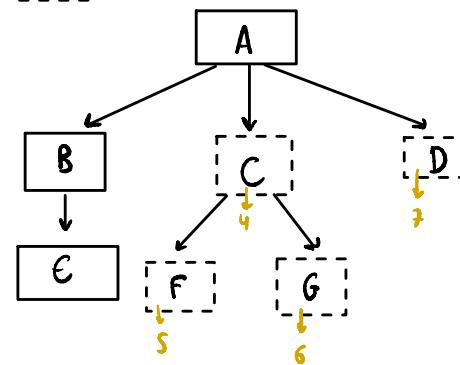
Fase 1



Fase 2



Fase 3



:

Fase 7

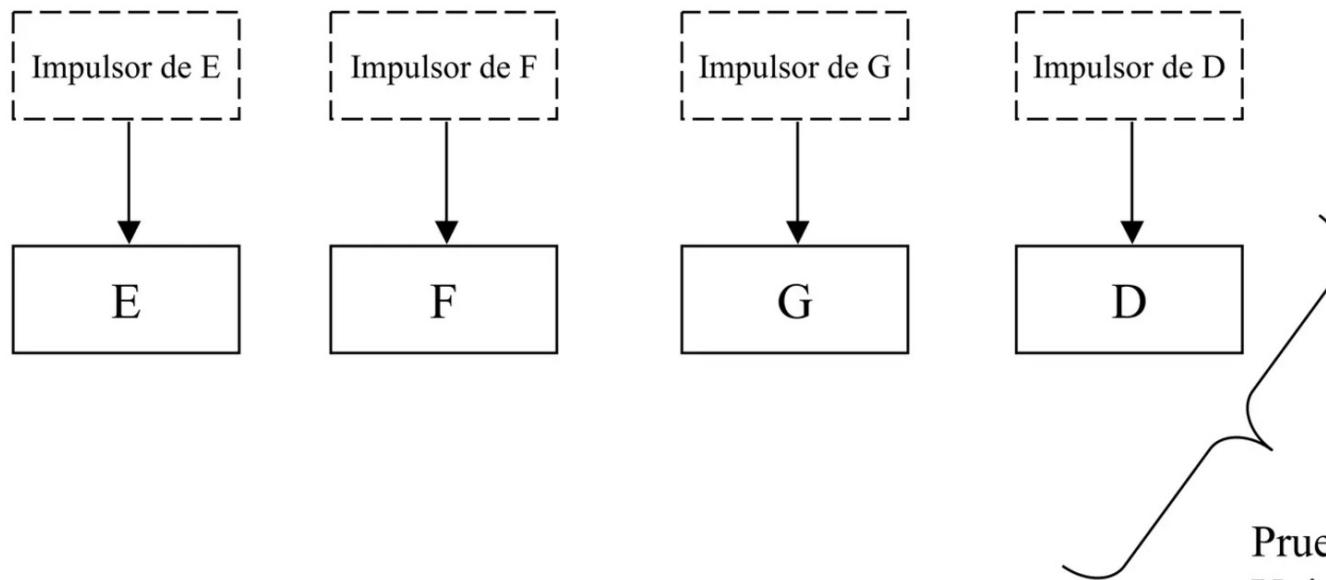
# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Integración Incremental Ascendente

1<sup>a</sup> fase

Se definen los impulsores para nodos hoja y se ejecutan

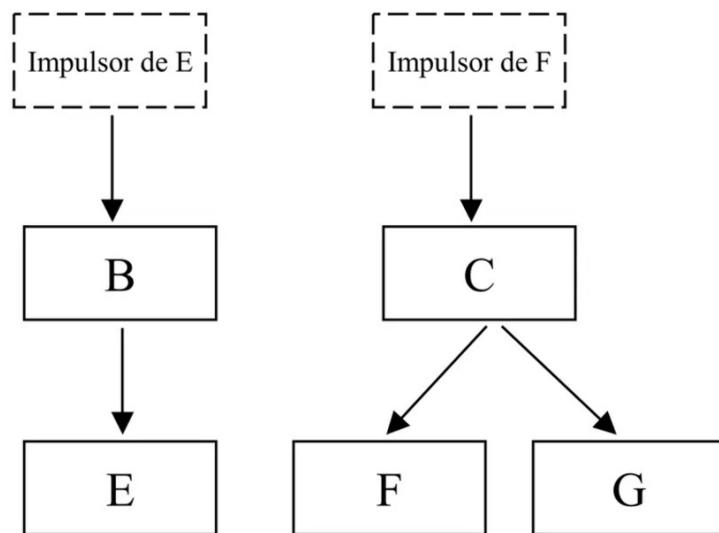


# Estrategia de Aplicación de Pruebas

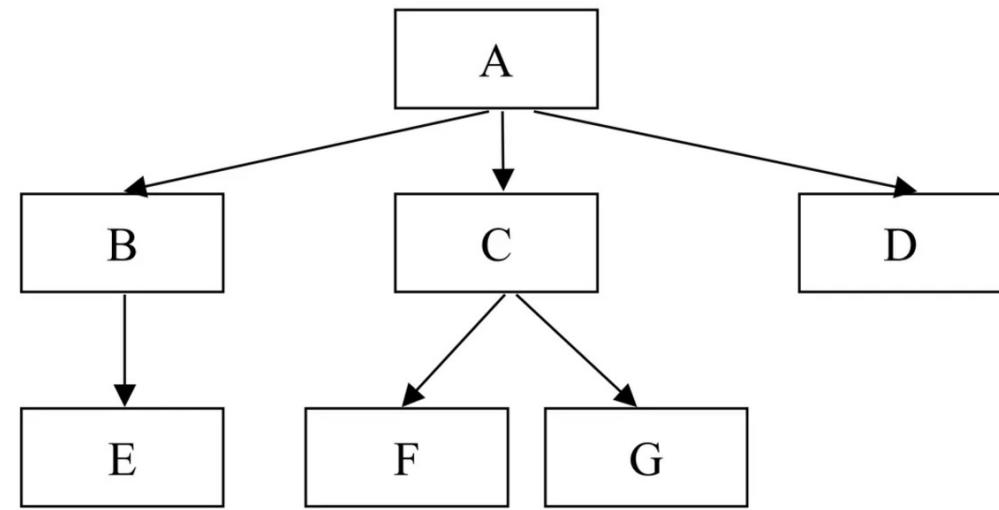
## Prueba de Integración

### Integración Incremental Ascendente

2<sup>a</sup> fase



3<sup>a</sup> fase



Prueba de unidad de E

Prueba de integración de B con E



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Integración Incremental Descendente

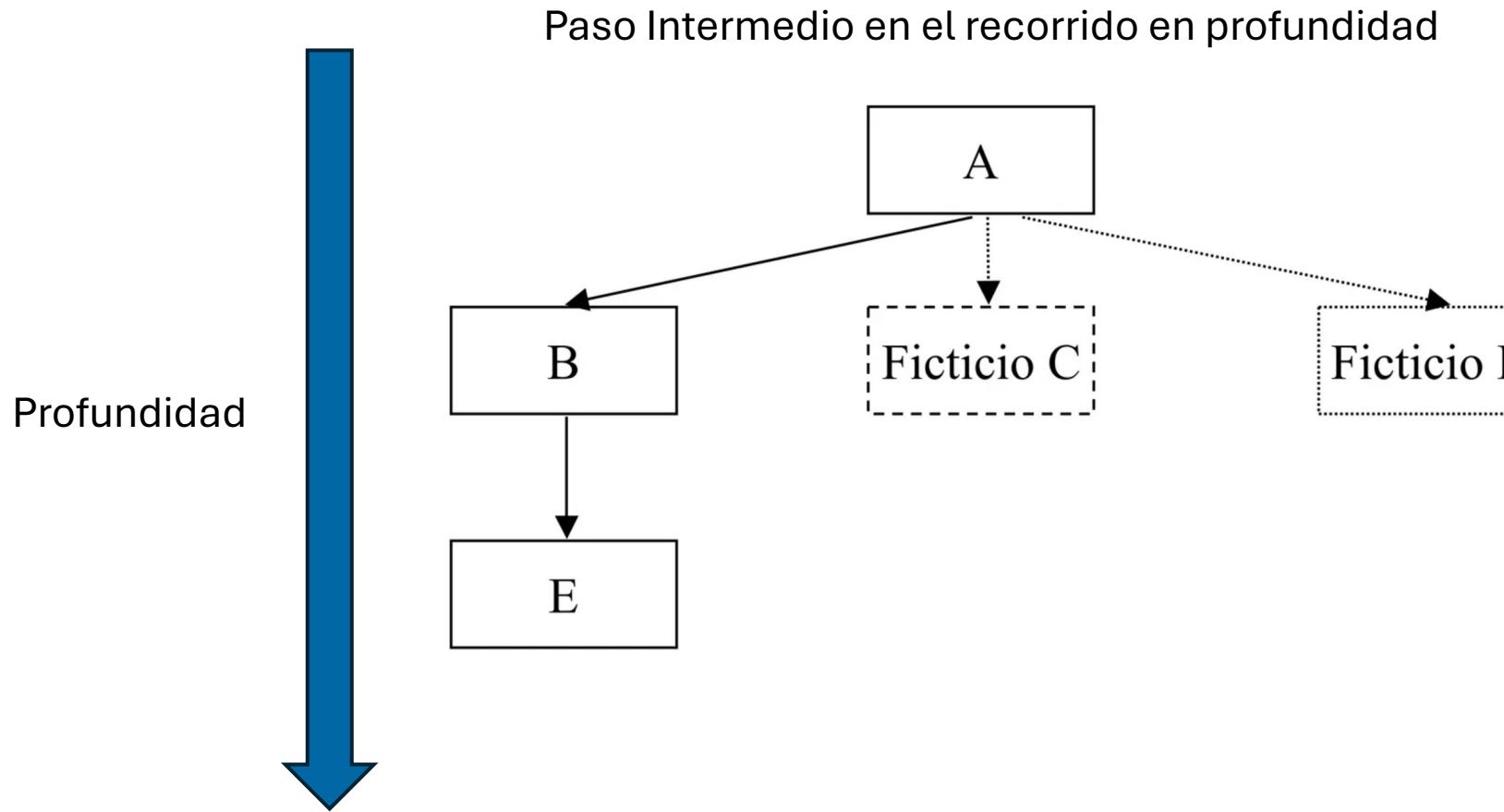
- Se comienza probando el módulo raíz con módulos ficticios
- Luego, se reemplazan los módulos ficticios por los reales, uno a uno
- Se realizan pruebas cada vez que se incorpora un nuevo módulo
- Se recomienda repetir pruebas anteriores para evitar nuevos errores
- Existen dos formas básicas de hacer esta prueba de integración descendente:
  - Primero en **Profundidad**: completando ramas del árbol (A, B, E, C, F, G, D)
  - Primero en **Anchura**: completando niveles de jerarquía (A, B, C, D, E, F, G)



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Integración Incremental Descendente

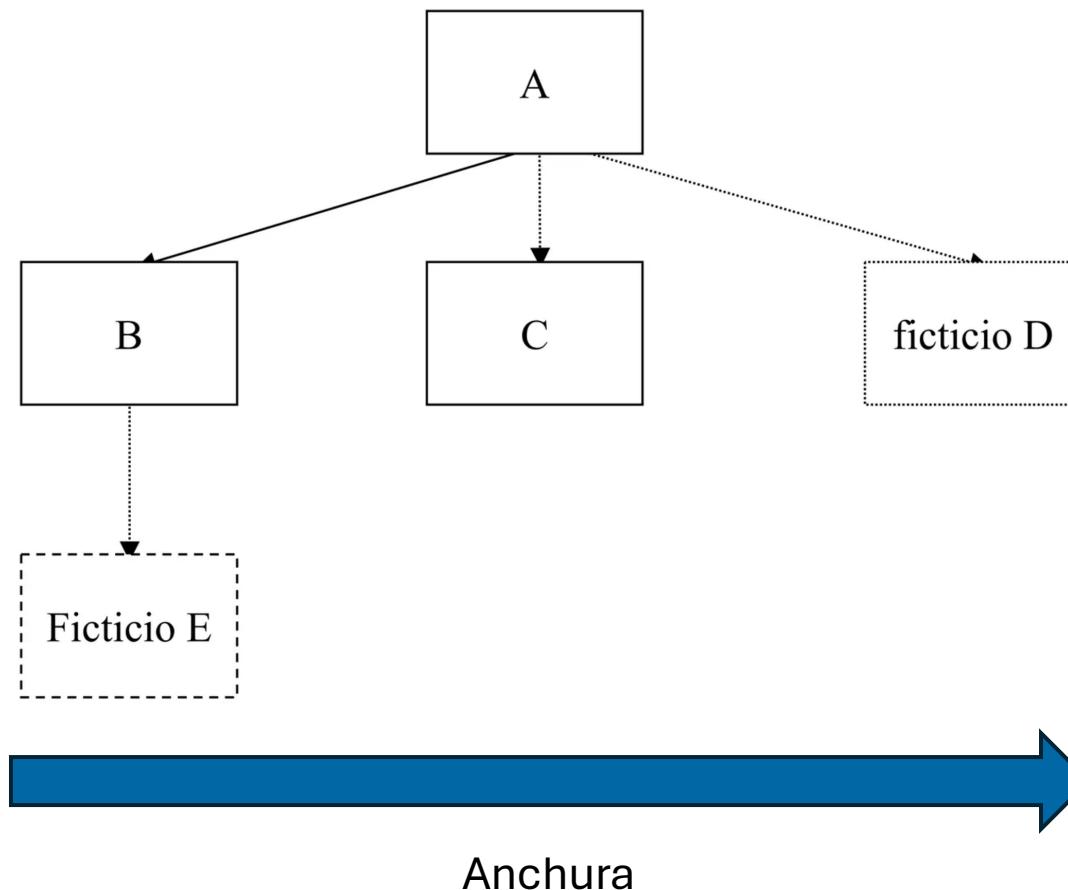


# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Integración Incremental Descendente

Paso Intermedio en el recorrido en anchura



# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Comparación Integración Incremental Ascendente y Descendente

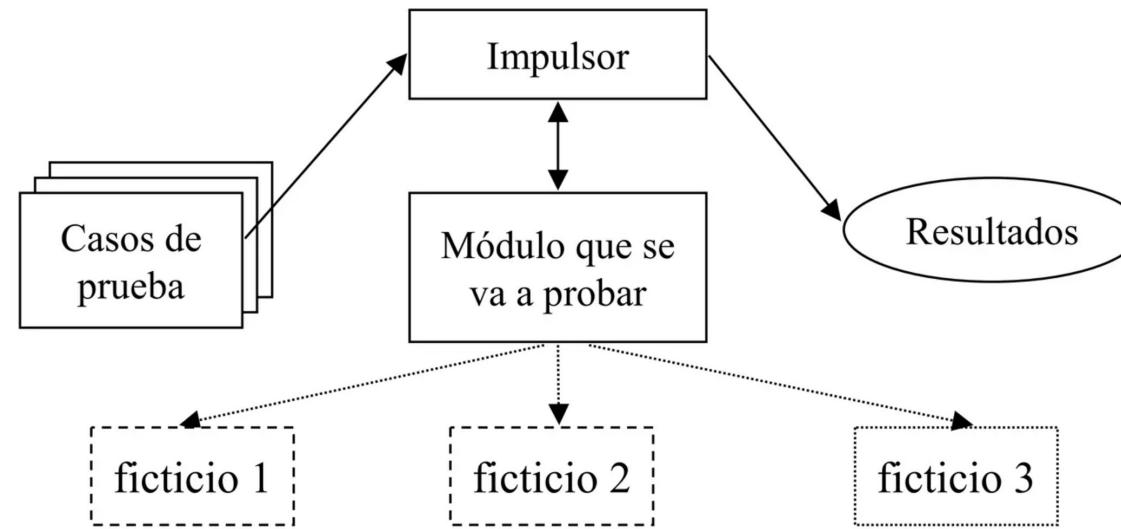
	Ascendente	Descendente
Ventajas	<ul style="list-style-type: none"><li>Los fallos en las partes inferiores del sistema se detectan más rápidamente</li><li>Los casos de prueba son más sencillos debido a que los módulos inferiores tienen funciones específicas</li><li>Es más fácil observar los resultados de las pruebas, ya que los módulos inferiores elaboran los datos</li></ul>	<ul style="list-style-type: none"><li>Detecta defectos en niveles superiores temprano</li><li>Facilita el manejo de casos de prueba al incluir funciones de entrada/salida</li><li>Permite mostrar una estructura previa del programa</li></ul>
Desventajas	<ul style="list-style-type: none"><li>Se necesitan módulos impulsores adicionales que deben ser codificados</li><li>El sistema completo no se prueba hasta que se añade el último módulo</li></ul>	<ul style="list-style-type: none"><li>Los módulos ficticios son complejos de crear</li><li>Difícil manejar casos de prueba antes de integrar entrada/salida</li><li>A veces, no se pueden crear casos de prueba debido a los detalles de los módulos inferiores</li><li>Resultados difíciles de observar, ya que provienen de módulos inferiores</li><li>Puede retrasar la finalización de pruebas si parece que el programa ya funciona</li></ul>

# Estrategia de Aplicación de Pruebas

## Prueba de Integración

### Proceso No Incremental

- Cada módulo que tiene que ser probado necesita lo siguiente:
  - Un módulo impulsor, que envía los datos de prueba y muestra los resultados
  - Uno o más módulos ficticios que simulan la función de cada módulo subordinado llamado por el módulo que se va a probar
- Una vez probado cada módulo por separado, se ensamblan todos de una vez y se prueban en conjunto



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Estrategia de Aplicación de Pruebas

## Prueba del Sistema

- Es el proceso de prueba de un sistema integrado de hardware y software para comprobar lo siguiente:
  - Cumplimiento de todos los requisitos funcionales del producto completo en un entorno de sistema
  - El funcionamiento y rendimiento en las interfaces hardware, software, de usuario y de operador
  - Adecuación de la documentación de usuario
  - Ejecución y rendimiento en condiciones límite y de sobrecarga
- Las pruebas se basan en:
  - Casos basados en los requisitos gracias a técnicas de caja negra aplicadas a las especificaciones
  - Casos necesarios para probar el rendimiento del sistema (pruebas de carga)
  - Casos basados en el diseño de alto nivel aplicando técnicas de caja blanca a los flujos de datos de alto nivel



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Estrategia de Aplicación de Pruebas

## Prueba de Aceptación

- Es una prueba planificada y organizada formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente
- Sus características principales son las siguientes:
  - Participación del usuario
  - Está enfocada hacia la prueba de los requisitos de usuario especificados
  - Está considerada como la fase final del proceso para asegurar que el producto sea adecuado para uso
- Recomendaciones:
  - Debe contarse con criterios de aceptación previamente aprobados por el usuario
  - Validar también la documentación y los procedimientos de uso (auditoría)
  - Realizar las pruebas en el entorno real de uso, con el personal correspondiente
    - En sistemas contratados, se realiza bajo el control del equipo de desarrollo (pruebas Alfa)
    - Para productos de interés general, se entrega a usuarios de confianza para obtener retroalimentación (pruebas Beta)



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Técnicas de Diseño de Casos de Prueba

- Su objetivo es garantizar la detección de defectos con una cantidad eficiente de recursos
- La idea fundamental consiste en seleccionar los casos más representativos que cubran el máximo posible de escenarios
  - Alcanzar el **máximo** nivel de **cobertura** posible con el **mínimo número de casos de prueba**
- Si no se detectan errores, se puede asumir con cierta confianza que el programa está libre de defectos



# Técnicas de Diseño de Casos de Prueba

Existen varios enfoques para el diseño de casos de prueba:

- Enfoque **basado en requisitos**
  - Diseña pruebas para verificar el cumplimiento de los requisitos del sistema
  - Útil en pruebas de sistema, ya que un requisito puede involucrar varios componentes
  - Cada requisito tiene un caso de prueba asociado
- Enfoque **funcional** (o **caja negra**)
  - Identifica grupos de datos (particiones) de entrada y salida con características comunes
  - Diseña pruebas para cubrir todas las entradas y salidas posibles
- Enfoque **estructural** (o **caja blanca**)
  - Basado en la estructura interna del programa
  - Asegura que cada parte del código (sentencias) se ejecute al menos una vez
- Enfoque **aleatorio**
  - Utiliza modelos estadísticos para simular entradas al programa
  - Incluye técnicas como la siembra de fallos:

$$fallo\ real = \frac{fallo\ real\ detectados}{fallo\ intencional\ detectados}$$



# Índice

1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Técnicas de Diseño de Casos de Prueba

## Pruebas Basadas en Requisitos

- Los requisitos deben ser comprobables
- Para cada requisito, se diseñan pruebas específicas que validen su cumplimiento

### Ejemplo de Requisitos

1. Permitir búsquedas en todas las bases de datos o en un subconjunto
2. Mostrar vistas adecuadas para leer documentos
3. Cada solicitud debe incluir un identificador único (ORDER\_ID)

### Ejemplo de Pruebas (Requisito 1)

- Buscar en una base de datos: elementos presentes y ausentes
- Buscar en dos bases de datos: elementos presentes y ausentes
- Buscar en más de dos bases de datos: elementos presentes y ausentes
- Seleccionar una base y buscar elementos presentes y ausentes
- Seleccionar múltiples bases y buscar elementos presentes y ausentes



# Índice

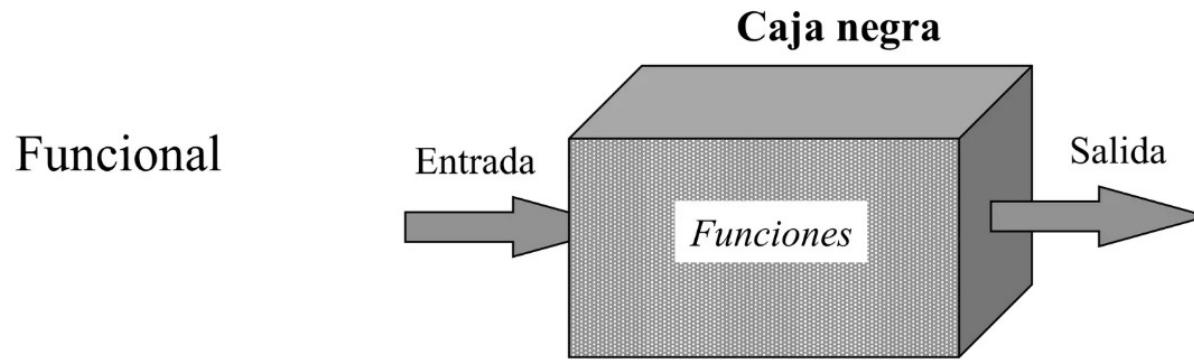
1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. **Pruebas Funcionales o de Caja Negra**
  3. Pruebas Estructurales o de Caja Blanca



# Técnicas de Diseño de Casos de Prueba

## Pruebas Funcionales o de Caja Negra

- Se centran en funciones, entradas y salidas del sistema → No es viable probar todas las combinaciones posibles, por lo que es crucial seleccionar casos de prueba efectivos



Criterios para un buen caso de prueba funcional:

- Optimización: reduciendo la cantidad total de casos necesarios al abarcar el mayor número de posibilidades de entrada en una sola prueba
- Cobertura amplia: representar un conjunto amplio de casos similares, para identificar defectos tanto en las entradas probadas como en otras entradas relacionadas



# Técnicas de Diseño de Casos de Prueba

## Pruebas Funcionales o de Caja Negra

### **Técnica: Prueba de Partición o Clases de Equivalencia**

Las cualidades que deben definir a un buen caso de prueba son:

- Cada caso debe cubrir el máximo número de entradas
- Debe tratarse el dominio de valores de entrada dividido en un número finito de clases de equivalencia, que cumplan la siguiente propiedad:

La prueba de un valor representativo de una clase permite suponer razonablemente que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase

- El método de diseño de casos consistirá entonces en:
  - Identificación de las clases de equivalencia
  - Creación de los casos de prueba correspondientes



# Técnicas de Diseño de Casos de Prueba

## Pruebas Funcionales o de Caja Negra

### **Técnica: Prueba de Partición o Clases de Equivalencia**

Pasos para identificar clases de equivalencia:

- Identificación de las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada
- A partir de ellas, se identifican clases de equivalencia que pueden ser:
  - De datos válidos
  - De datos no válidos o erróneos
- Asignar un número identificador único a cada clase de equivalencia creada
- Crear casos de prueba que incluyan todas las clases de equivalencia válidas posibles en una sola ejecución
- Crear un caso de prueba por cada clase no válida, para asegurar que cada condición no válida sea probada de manera aislada y no acumular los errores de varias entradas no válidas



# Técnicas de Diseño de Casos de Prueba

## Pruebas Funcionales o de Caja Negra

### Técnica: Prueba de Partición o Clases de Equivalencia

Existen algunas reglas que ayudan a identificar las clases de equivalencia:

- **Regla 1:** Si se especifica un rango de valores para los datos de entrada, se creará una clase válida y dos clases no validas
- **Regla 2:** Si se especifica un número de valores finito y consecutivo, se creará una clase válida y dos no válidas
- **Regla 3:** Si se especifica una situación del tipo «debe ser» o booleana, se identifica una clase válida y una no válida
- **Regla 4:** Si se especifica un conjunto de valores admitidos y se sabe que el programa trata de forma diferente cada uno de ellos, se identifica una clase válida por cada valor y una no válida (cualquier otro caso)
- **Regla 5:** En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores



**Regla 1:** entre 5 y 20

Clase válida  $\rightarrow 5 \leq \text{número} \leq 20$

Clase no válida  $\left\{ \begin{array}{l} \text{número} < 5 \\ \text{número} > 20 \end{array} \right.$

**Regla 2:** titulares eventos bancarios 1 a 4

Clase válida  $\rightarrow 1 \leq \text{titulares} \leq 4$

Clase no válida  $\left\{ \begin{array}{l} \text{titulares} < 1 \\ \text{titulares} > 4 \end{array} \right.$

# Técnicas de Diseño de Casos de Prueba

## Pruebas Funcionales o de Caja Negra

### Técnica: Prueba de Partición o Clases de Equivalencia

Ejemplo: Aplicación bancaria en la que el operador debe proporcionar un código, un nombre y una operación

Información de entrada	Regla	Clases Válidas		Clases No Válidas	
		ID	Dominio	ID	Dominio
Código área  Nº de 3 dígitos que no empieza con 0 ni 1)	1	1	200 ≤ código ≤ 999	2	código < 200
	3			3	código > 999
Nombre para identificar la operación	2	5	6 caracteres	4	no es número
Orden  Una de las siguientes →	4	8 9 10 11	cheque depósito pago factura retirada de fondos	6 7 12	menos de 6 caracteres más de 6 caracteres ninguna orden válida

- Habría que diseñar casos de prueba que cubran todas las clases de equivalencia, tanto válidas como no válidas, y para las no válidas en casos de prueba distintos



# Índice

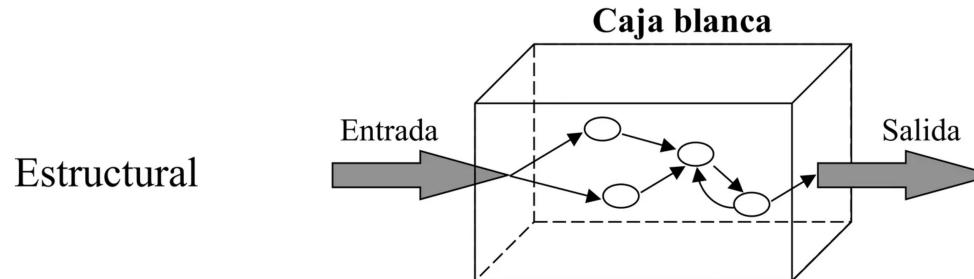
1. Introducción
2. Técnicas de V & V
3. Pruebas del Software
4. Tipos de Pruebas
  1. Pruebas Funcionales
  2. Pruebas No Funcionales
5. Desarrollo Guiado por Pruebas (TDD)
6. Estrategia de Aplicación de Pruebas
  1. Prueba Unitaria
  2. Prueba de Integración
  3. Prueba del Sistema
  4. Prueba de Aceptación
7. Técnicas de Diseño de Casos de Prueba
  1. Pruebas Basadas en Requisitos
  2. Pruebas Funcionales o de Caja Negra
  3. Pruebas Estructurales o de Caja Blanca



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

- Se centra en la estructura interna del programa → analiza los caminos de ejecución



- Se usan para diseñar casos de prueba basados en caminos importantes para garantizar un nivel aceptable de detección de defectos
- **Ejemplo:** Un programa de 50 líneas con 25 sentencias if en serie da lugar a 33,5 millones de secuencias posibles
- Para ello se pueden derivar casos de prueba que:
  - Garanticen que se ejecutan por lo menos una vez todos los caminos independientes de cada módulo
  - Se ejecutan todas las decisiones lógicas (caras verdaderas y falsas)
  - Se ejecutan todos sus bucles en sus límites
  - Se ejecutan las estructuras de datos internas para asegurar su validez



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

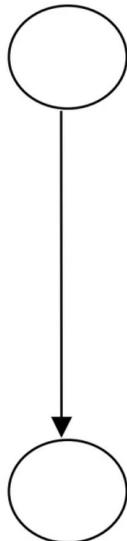
- No requieren representación gráfica específica, pero suelen usar como base los **Diagramas de Flujo de Control**
- Se realizan siguiendo los siguientes pasos:
  1. Identificar y marcar cada condición en decisiones en el código (IF-THEN, CASE-OF, WHILE, UNTIL)
  2. Agrupar las sentencias entre condiciones según estructuras básicas
  3. Numerar condiciones y grupos de sentencias con identificadores únicos
  4. Reordenar condiciones en decisiones multicondicionales en orden decreciente de restricción para facilitar el diseño de pruebas



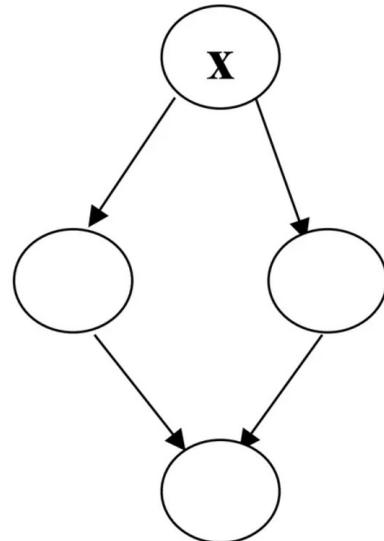
# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

- Construcciones básicas en los diagramas de flujo de control



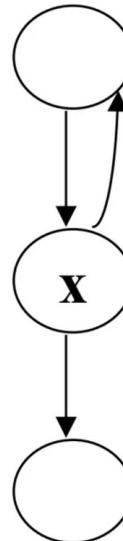
**Secuencia**



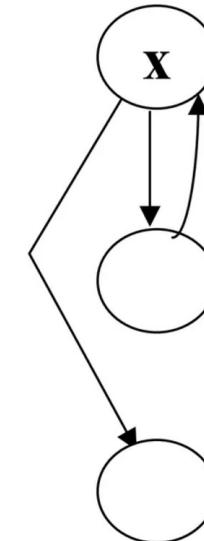
**Si x entonces...**  
**(If x then...else...)**

**Consejos:**

- Separar todas las condiciones
- Agrupar sentencias ‘simples’ en bloques
- Numerar todos los bloques de sentencias y también las condiciones



**Hacer... hasta x**  
**(Do...until x)**  
**Repetir**



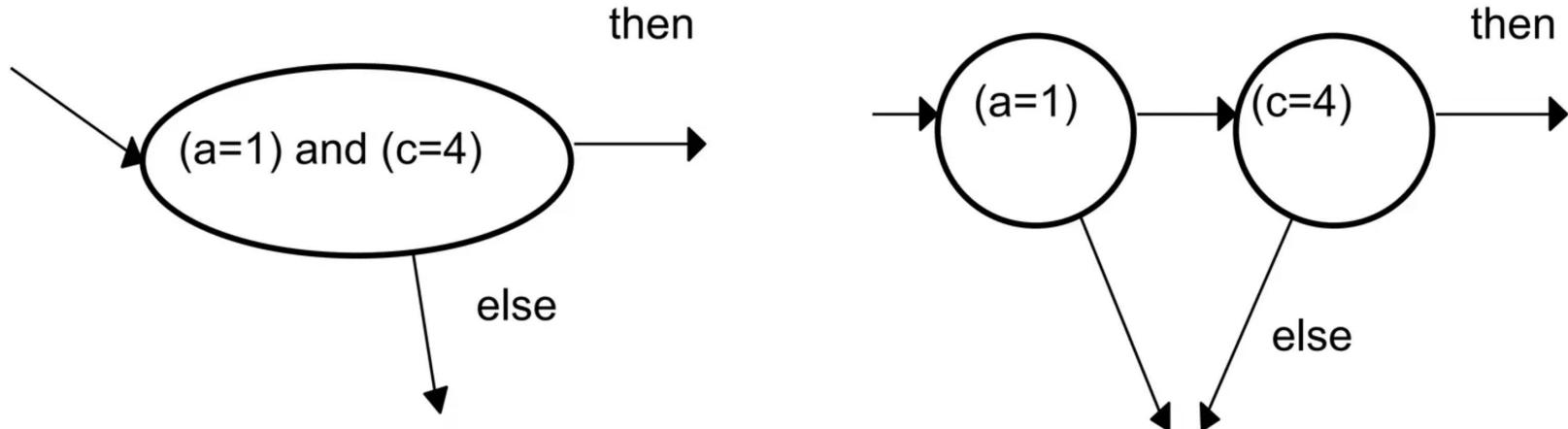
**Mientras x hacer...**  
**(While x do...)**



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

- Ejemplo de descomposición de una decisión multicondicional



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

- La cobertura de caminos es una secuencia de sentencias desde el inicio hasta el final del programa. Es el criterio principal en pruebas de caja blanca
- Probar todos los caminos puede ser impracticable, incluso para programas pequeños
- Como solución se propone usar **caminos de prueba**, que recorren cada buble como máximo una vez
- Fue propuesta por McCabe
- Mide los caminos independientes usando la **Complejidad Ciclomática  $V(G)$**



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

- La Complejidad Ciclomática aporta el límite superior que indica el número máximo de pruebas necesarias para ejecutar cada sentencia al menos una vez
- Camino independiente: Recorrido que introduce nuevas sentencias o condiciones
  - En un grafo de flujo de control, pasa por al menos una arista no recorrida previamente

La complejidad de McCabe se puede calcular de cualquiera de estas 3 formas

1.  $V(G) = a - n + 2$ , siendo  $a$  el número de arcos o aristas del grafo y  $n$  el número de nodos.
2.  $V(G) = r$ , siendo  $r$  el número de regiones cerradas del grafo.
3.  $V(G) = c + 1$ , siendo  $c$  el número de nodos de condición.



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

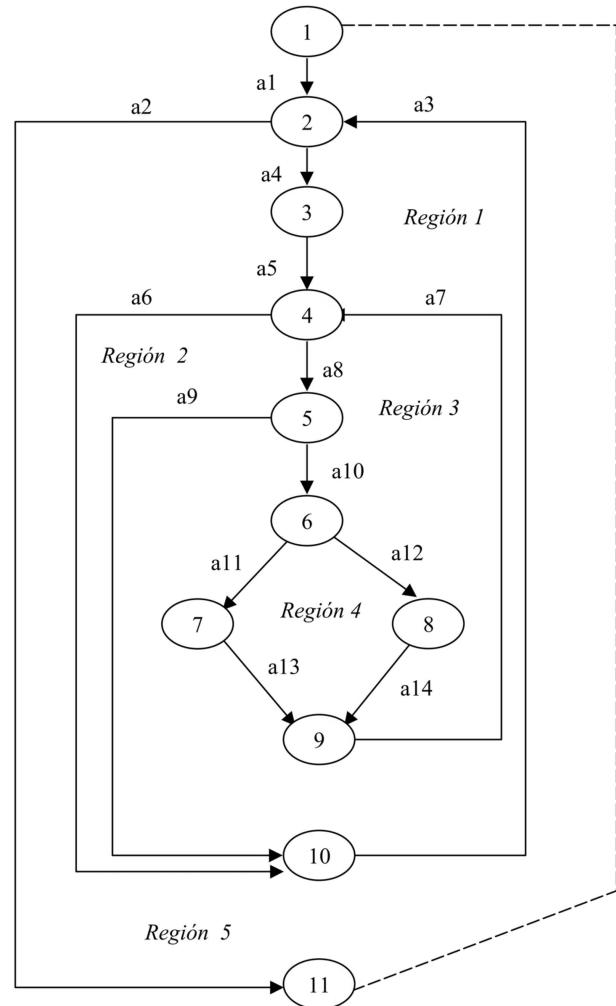
- Un nodo predicado contiene una condición y tiene dos o más salidas (aristas)
- Criterio de prueba de McCabe: Usar tantos casos de prueba como caminos independientes (calculados como  $V(G)$ )
- La experiencia en este campo asegura que:
  - $V(G)$  es el mínimo de casos de prueba necesarios
  - Si  $V(G) > 10$ , aumenta la probabilidad de defectos en el módulo → puede ser útil dividir el módulo
- Procedimiento para derivar casos de prueba:
  1. Dibujar el grafo de flujo de control
  2. Calcular la complejidad ciclomática de dicho grafo
  3. Identificar caminos linealmente independientes
  4. Preparar casos de prueba para ejecutar cada camino básico



# Técnicas de Diseño de Casos de Prueba

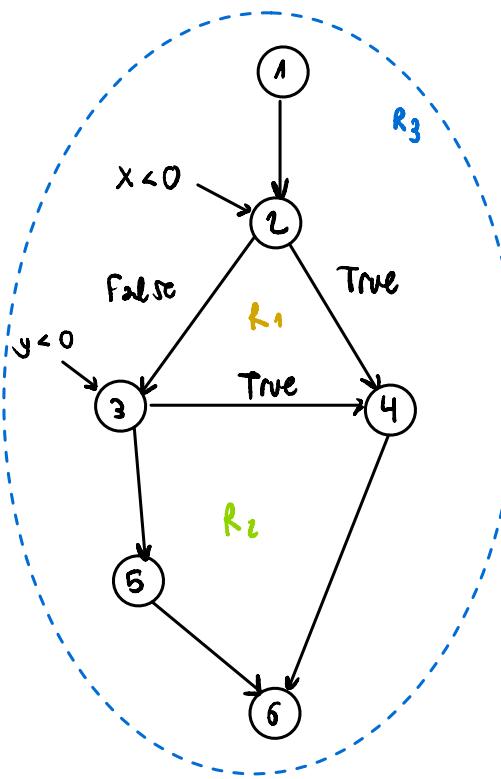
## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico



- a)  $V(G) = 14 - 11 + 2 = 5$
- b) 5 regiones cerradas
- c) 5 condiciones





$$\begin{aligned}
 V(G) &= F - E + 2 = 3 \\
 &= 3 \text{ regions} \\
 &= 2 + 1 = 3
 \end{aligned}$$

Camino 1: 1, 2, 3, 5, 6

Camino 2: 1, 2, 4, 6

Camino 3: 1, 2, 3, 4, 6

	1	2	3	4	5	6	n-1
1	0	1	0	0	0	0	0
2	0	0	1	1	0	0	1
3	0	0	0	1	1	0	1
4	0	0	0	0	0	1	0
5	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0

$$V(G) = 2 + 1 = 3$$

# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

```
PROCEDURE imprime_media (VAR x, y : real;)
VAR resultado : real;
1 resultado := 0;
2 IF (x < 0 OR y < 0)
3 THEN
4     WRITELN("x e y deben ser positivos");
5 ELSE
6     resultado := (x + y)/2
7     WRITELN("La media es: ", resultado);
8 ENDIF
9 END imprime_media
```



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

```
PROCEDURE imprime_media (VAR x, y : real;)
VAR resultado : real;
1  resultado := 0; 3
2  IF (x < 0 OR y < 0) 3
3  THEN 2
4      WRITELN("x e y deben ser positivos"); 4
5  ELSE
6      resultado := (x + y)/2
7      WRITELN("La media es: ", resultado);
8  ENDIF 6
9  END imprime_media
```



# Técnicas de Diseño de Casos de Prueba

## Pruebas Estructurales o de Caja Blanca

### Técnica: Prueba del Camino Básico

- Para mecanizar la determinación del conjunto básico de caminos se utilizará una **matriz de conexiones**:
  - Matriz cuadrada basada en el número de nodos
  - Las entradas representan las conexiones entre nodos (aristas)
- Cálculo de  $V(G)$ :
  1. Añadir una columna con el sumatorio de cada fila menos 1
  2. La suma de esta columna más 1 es la complejidad ciclomática
- Usos adicionales de la matriz:
  - Probabilidad de ejecución de una arista
  - Tiempo, memoria o recursos requeridos por un enlace
- Una vez calculada la complejidad ciclomática por los distintos métodos se procede a preparar los casos de prueba que recorran cada camino independiente del conjunto básico

