

# Prueba de software

Ingeniería de software 1

# Temario

- Prueba de software.
- Estrategias, niveles y tipos de prueba.
- Pruebas de caja blanca.
- Pruebas de caja negra.
- Proceso de prueba.
- Herramientas de prueba.
- Plan de prueba.

# **PRUEBA DE SOFTWARE**

# Prueba de software

- Definición de prueba [Glen Myers]
  - Es el proceso de ejecutar el software con el objetivo de encontrar defectos.
- Una prueba tiene éxito si **descubre errores**.
- Un buen caso de prueba es aquel con **alta probabilidad de descubrir un error** no encontrado hasta el momento.
  - Un programa hace lo que tiene que hacer.
  - Un programa no hace lo que no tiene que hacer.

# Prueba de software

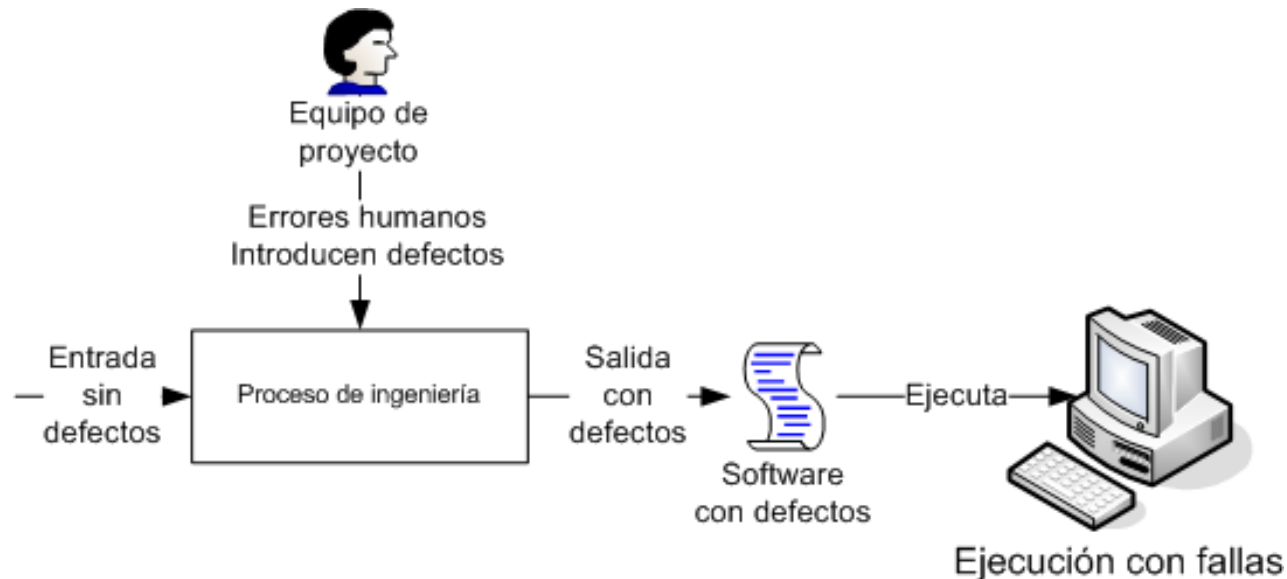
- Corolario sobre la prueba:
  - “La ausencia de evidencias no es evidencia de ausencia”.
- Si la prueba no descubre errores no se pueda afirmar que no existan.

# Prueba de software

- Error humano
  - Acción humana que produce un resultado incorrecto.
- Defecto / Falta [IEEE]
  - Anomalía del producto.
- Fallo [IEEE].
  - Desviación del componente o sistema de su comportamiento, servicio o resultado esperado.

# Prueba de software

- Error vs. Defecto / Fallo
  - Los sistemas tienen defectos a consecuencias de errores introducidos por el equipo de proyecto y se producen fallos en la ejecución.



# Prueba de software

- Principios de la prueba [Davis]
  - Debe existir **trazabilidad** de ida y vuelta entre los requerimientos y los casos de prueba.
  - Se debe **planificar** la prueba antes de construir el sistema.
  - Aplicar el principio de **Pareto** (80-20).
  - Las pruebas deben ir **de lo pequeño a lo grande**.
  - No son posibles pruebas exhaustivas.
  - Realizar las pruebas por un **equipo independiente**.



# Prueba de software

- **Verificar**

- *¿Estamos construyendo correctamente el producto?*
- Consiste en mostrar que el software cumple con las **especificaciones**.

- **Validar**

- *¿Estamos construyendo el producto correcto?*
- Consiste en mostrar que el software cumple con las **expectativas de cliente**.

# **ESTRATEGIAS, NIVELES Y TIPOS DE PRUEBA**

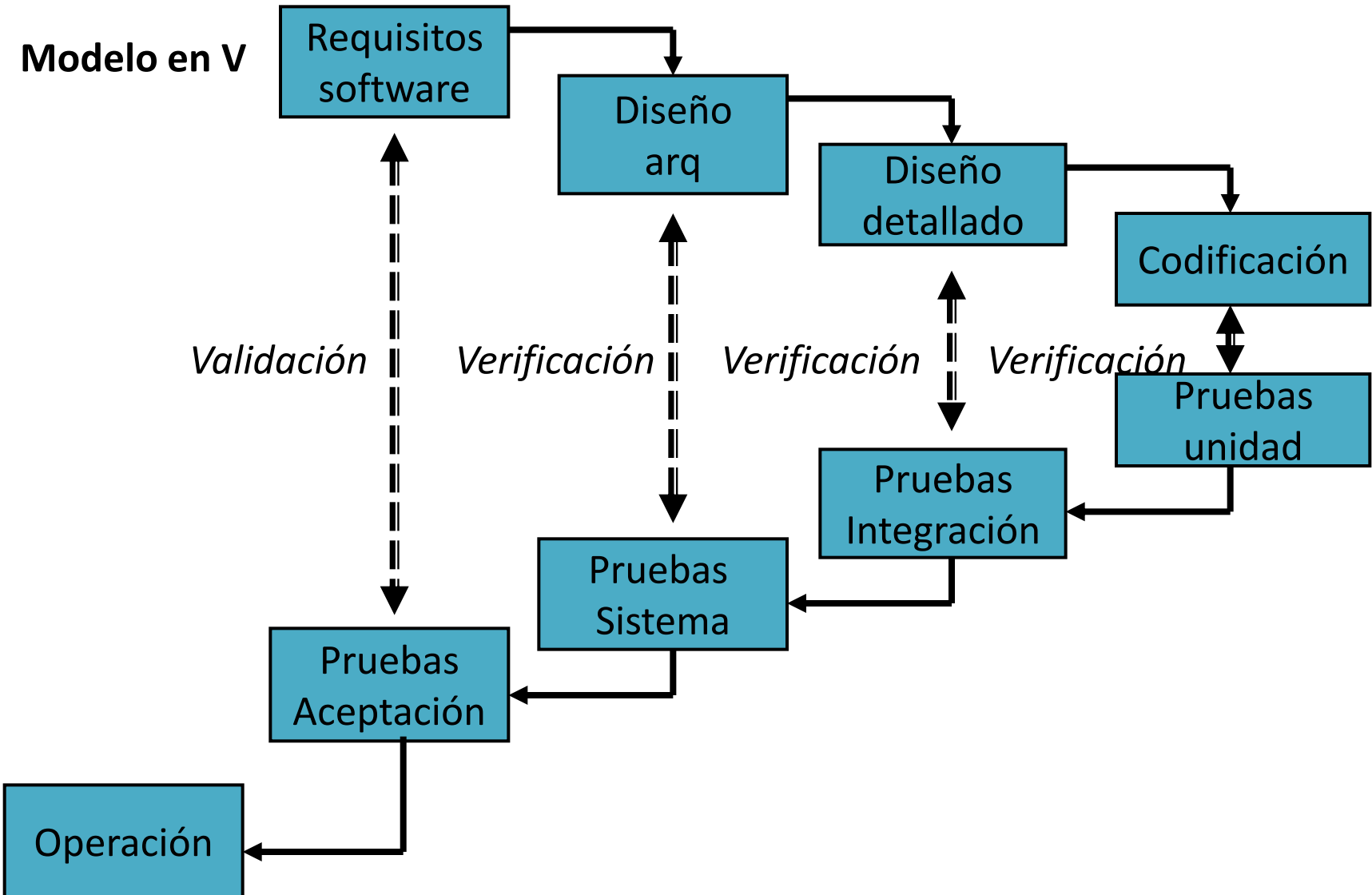
# Estrategias de prueba

- Una estrategia de prueba debe considerar:
  - planificación de la prueba
  - diseño de casos de prueba
  - ejecución de pruebas
  - análisis y reporte de defectos
- Características generales de las estrategias
  - Comienzan a nivel de módulo y se trabaja hacia la integración de todo el sistema
  - Según el momento son apropiadas distintas técnicas
  - Se deben asignar recursos y responsables
  - La prueba y la depuración son actividades diferentes, pero la depuración debe incluirse en la estrategia de prueba

# Estrategias de prueba

- Basada en escenarios o usos del sistema
  - Probar casos que reflejan el uso real del sistema
- Basada en riesgos
  - Priorizar el esfuerzo de las pruebas relativo al impacto probable del defecto
- Basada en funciones / estados
  - Probar cada función/estado del sistema. Uno a la vez.
- Basada en dominios
  - Se clasifican las entradas, procesamientos y salidas en clases de equivalencia
- Exploratorias
  - Se aprende del sistema en las propias pruebas

# Niveles de prueba



# Niveles de prueba

- Prueba unitaria
  - Verifica el funcionamiento de **programas o módulos en forma aislada**.
  - Generalmente la realiza el propio desarrollador.
  - se desarrollan programas conductores (aceptan los datos del caso de prueba, se los pasan al módulo e imprimen el resultado) y/o programas de resguardo (para reemplazar módulos subordinados)
- Prueba de integración
  - Verifica el funcionamiento resultante de **la interacción de un conjunto de componentes** integrados.
  - Generalmente implica una estrategia de integración.

# Niveles de prueba

- Prueba de sistema
  - Se preocupa de verificar (y/o validar) el funcionamiento de todo el sistema.
  - Compara el sistema contra los **requerimientos** funcionales y no funcionales.
- Prueba de aceptación
  - Es una prueba del sistema con el objetivo de **validarlo por parte del usuario**.
  - Contrasta la funcionalidad del sistema con los requerimientos de los usuarios.

# Tipos de prueba

- Pruebas alfa y beta
  - Son pruebas realizadas antes de la liberación del sistema, por el usuario final.
    - Prueba alfa: en la lugar de desarrollo o un entorno controlado.
    - Prueba beta: por el propio usuario en su ambiente operativo.
  - Se realizan con un grupo seleccionando de usuarios.
  - Generalmente se realizan sobre un número finito de casos de prueba seleccionados al inicio del proyecto.

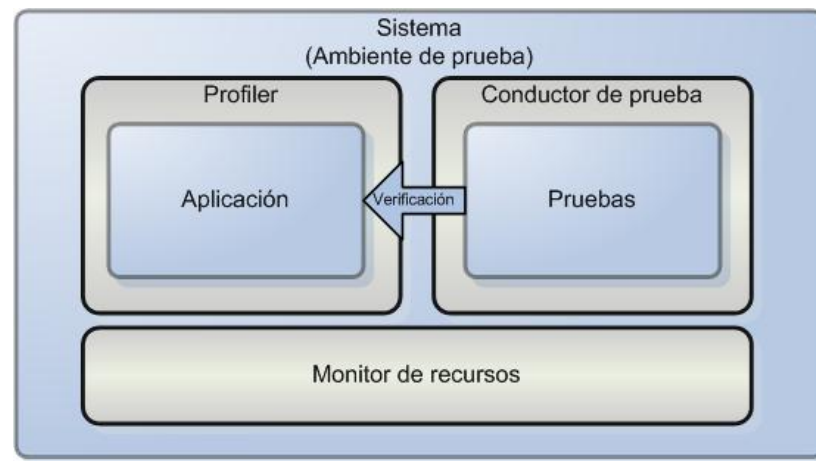


# Tipos de prueba

- Prueba de regresión
  - La prueba de sistema (o subsistema) se orienta a verificar que no se han introducido defectos cuando se modifica el software.
  - Volver a ejecutar un subconjunto de pruebas para asegurar que los cambios (debidos a pruebas u otros motivos) no han propagado efectos colaterales no deseados
  - Manuales o utilizando herramientas automáticas
  - 3 clases de casos de prueba:
    - muestra representativa de pruebas que ejercite todas las funciones
    - pruebas centradas en funciones que probablemente se vean afectadas por el cambio
    - pruebas centradas en los componentes que cambiaron

# Tipos de prueba

- Otros tipos de pruebas
  - Prueba de recuperación
  - Prueba de eficiencia (performance)
  - Prueba de seguridad
  - Prueba de carga
  - Prueba de usabilidad



# Técnicas de prueba

- Prueba de caja blanca (*estructural*)
  - Se analiza la **estructura lógica interna** de la aplicación.
  - Se pretende demostrar que los componentes internos de la aplicación se comportan adecuadamente.
- Prueba de caja negra (*funcional*)
  - Los casos de prueba se derivan de la **especificación** del módulo o programa.
  - El comportamiento del módulo se determina en función de sus **entradas y salidas**

# Ejercicio: Prueba de un Triángulo

- Programa Triangulo [Myers, 1979]
  - Construir casos de prueba para un programa con la siguiente especificación
    - El programa lee tres valores enteros
    - Los tres valores se interpretan como los lados de un triángulo.
    - Se imprime un mensaje indicando si el triángulo es escaleno, equilátero o isósceles.

# **PRUEBAS DE CAJA BLANCA**

# Pruebas de caja blanca

- Las pruebas de caja blanca buscan asegurar la cobertura:
  - Se ejecutan todos los caminos.
  - Se ejecutan todas la alternativas (true y false).
  - Se ejecutan todos los loops.
  - Se ejecutan todas la estructuras internas de datos.
- Dos técnicas:
  - Prueba del camino básico.
  - Pruebas de estructuras de control.

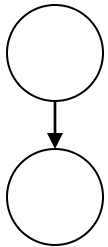
# Pruebas de caja blanca

- Técnica de caminos básicos [McCabe]
  - Busca definir un conjunto básico de caminos de ejecución.
  - Se basa en una métrica para definir este conjunto.
  - Utiliza una notación de grafos para llegar a la métrica.

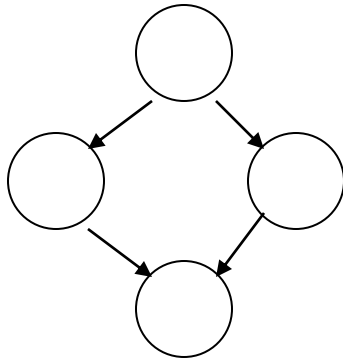
# Pruebas de caja blanca

- Grafo de flujo

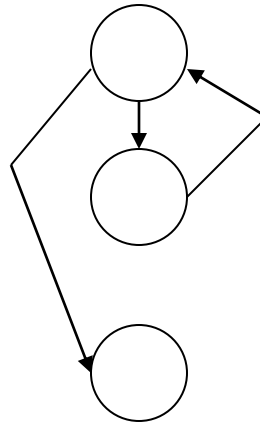
Secuencia



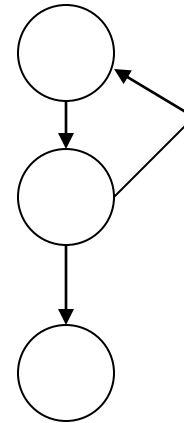
IF



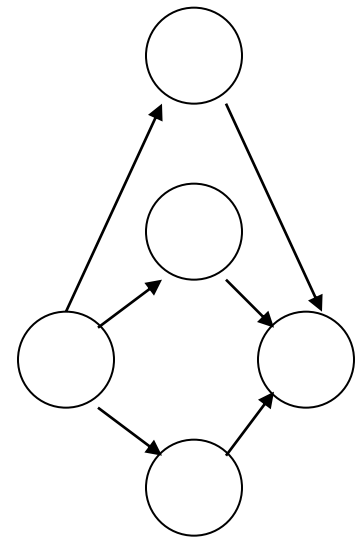
WHILE



UNTIL



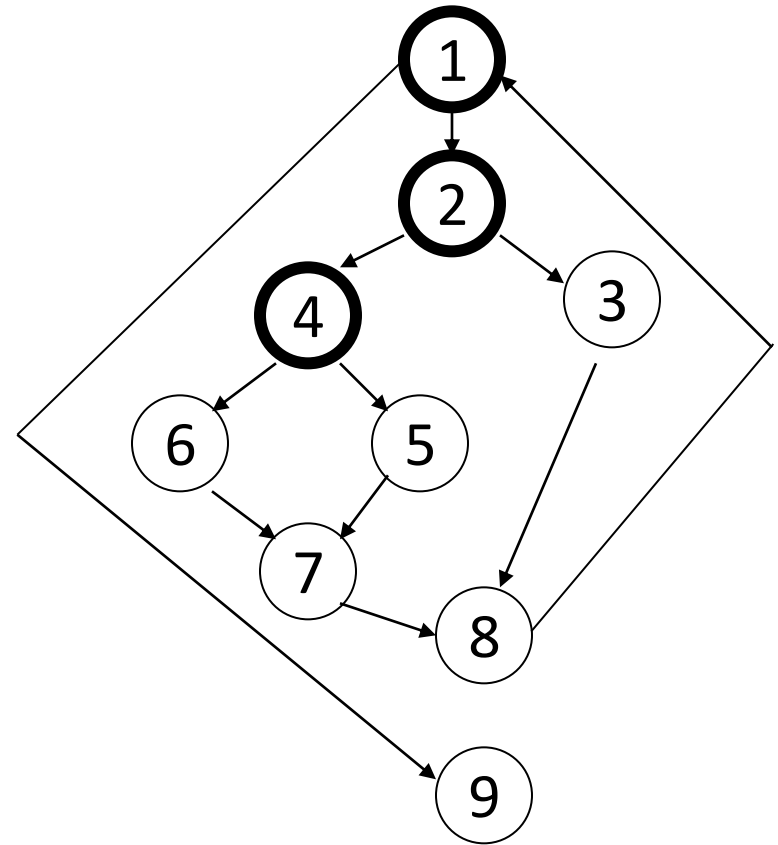
CASE





# Pruebas de caja blanca

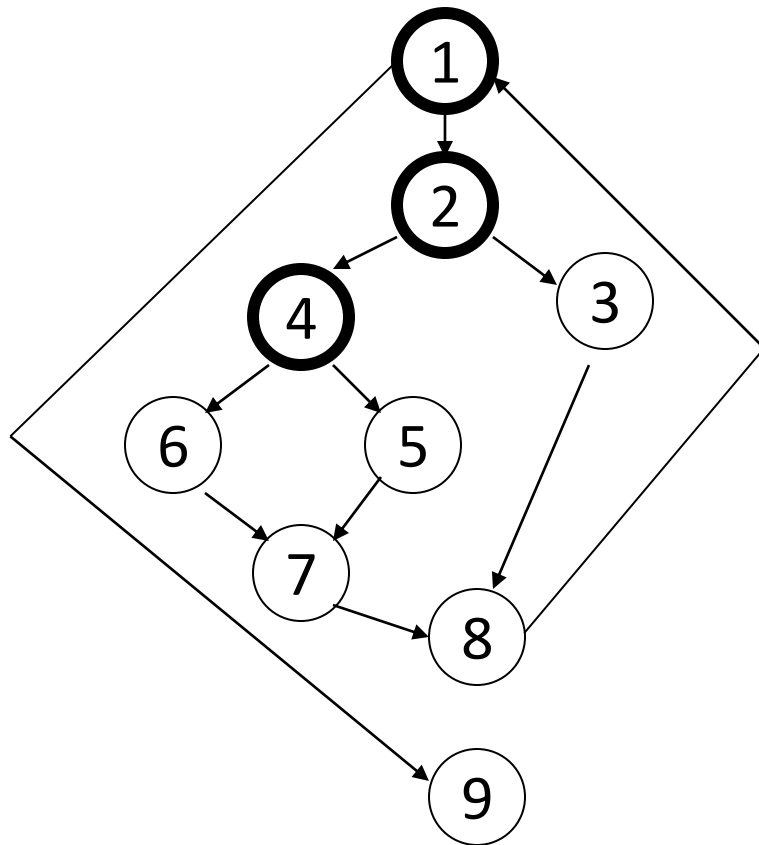
```
1: do while queden registros
    leer registros;
2:   if registro.campo_1 = 0 then
3:       procesar registro;
        guardar en buffer;
        incrementar contador;
4:   elseif registro.campo_2 = 0 then
5:       reiniciar contador;
6:   else
        procesar registro;
        grabar archivo;
7:   endif
8: enddo
9: end
```



# Pruebas de caja blanca

- Complejidad Ciclomática (CC)
  - Define el **número de caminos independientes** del conjunto básico de un programa.
    - Camino independiente: tiene que poseer al menos una arista no recorrida en otro camino.
  - Proporciona el límite superior de pruebas a realizar.
  - El límite superior nos asegura que se han recorrido todos los caminos al menos una vez.

# Pruebas de caja blanca



Caminos  
independientes.

1: 1 - 9.

2: 1 - 2 - 3 - 8 - 1 - 9.

3: 1 - 2 - 4 - 5 - 7 - 8 - 1 - 9.

4: 1 - 2 - 4 - 6 - 7 - 8 - 1 - 9.

Límite superior = 4.

# Pruebas de caja blanca

- Cálculo de la Complejidad Ciclomática ( $V$ ) del grafo  $G$ :
  - $V(G)$  = número de regiones del grafo.
  - $V(G) = A - N + 2$ .
    - Siendo  $A$  el nro. de aristas y  $N$  nro. de nodos.
  - $V(G) = \text{Nodos predicado} + 1$
- Para el ejemplo anterior.
  - $V(G) = 4$ .
  - $\Rightarrow$  Límite superior = 4.
  - $\Rightarrow$  Cantidad de casos de prueba = 4.

# Pruebas de caja blanca

- Aplicación de la técnica de caminos básicos:
  - Dibujar el correspondiente grafo de flujo.
  - Determinar la complejidad ciclomática del grafo.
  - Determinar el conjunto de caminos básicos.
  - Preparar casos de prueba.
    - Datos de entrada: ...
    - Resultados esperados: ...

# Pruebas de caja blanca

- Prueba de condicionales
  - Asegura la cobertura del código, probando todos los condicionales con valores de verdadero y falso en sus expresiones lógicas.
  - Los condicionales pueden ser simples o compuestos.

# Pruebas de caja blanca

- Prueba de condicionales
  - Condición lógica simple:
    - Variable lógica (ES\_VERDADERO).
    - Expresión relacional (por ej.  $A > B$ ),  
Siendo A y B expresiones aritméticas.
    - Puede incluir el NOT.
  - Condición lógica compuesta:
    - Dos o más condiciones simples.
    - Incluyendo operadores lógicos (OR, AND, NOT).
    - Y paréntesis.

# Pruebas de caja blanca

- Defectos posibles en un condicional:
  - En variable lógica.
  - En operador relacional.
  - En expresión aritmética.
  - En operadores lógicos.
  - En paréntesis.



# Pruebas de caja blanca

- Tipos de cobertura
  - Sentencia
    - Se cubren todas las sentencias de los condicionales
  - Decisión
    - Se cubren los verdadero y falso de cada decisión
  - Condición
    - Se cubre el verdadero y falso de cada condición de una decisión
  - Múltiples condiciones
    - Se cubren todas las combinaciones de condición en una decisión

# Pruebas de caja blanca

- Ejemplo

```
public int foo(int a, int b, int x) {  
    if ( a > 1 && b == 0){  
        x = x / a;  
    }  
    if ( a == 2 || x > 1){  
        x = x + 1;  
    }  
    return x;  
}
```

# Pruebas de caja blanca

- Prueba de condicionales
  - Dada una condición compuesta  $C$  ( $A > B$  OR  $A > H$ , por ej.), ejecútense al menos una vez la rama TRUE y la rama FALSE y cada condición simple de  $C$ .
  - Dada una expresión relacional ( $A > B$ , por ej.), ejecútense tres pruebas .

# Pruebas de caja blanca

- Prueba de bucles
  - Ejercita los bucles (loops) del código.
  - Siendo  $n$  la cantidad de pasos del loop.
    - Pasarlo totalmente por alto.
    - Pasar una sola vez.
    - Pasar dos veces.
    - Hacer  $m$  pasos, siendo  $m < n$ .
    - Hacer  $n-1$ ,  $n$  y  $n+1$  pasos.

# Pruebas de caja blanca

```
String[] textos = new String[]{ "linea1", "linea2",  
    "linea3" };  
  
for (int pos = 0; pos < LINEAS_IMPRIMIR; pos ++)  
{  
    lineas += textos[pos] + "\n";  
}
```

# Pruebas de caja blanca

- Prueba de bucles anidados
  - Comenzar por el bucle mas interior. Establecer los valores mínimos de los restantes.
  - Ejecutar las pruebas de bucles simples para el más interior, manteniendo los valores para los restantes.
  - Progresar hacia fuera hasta probar todos los bucles.

# Pruebas de caja blanca

- Prueba de bucles concatenados
  - Si son independientes.
    - Aplicar prueba de loops simples.
  - Si no son independientes.
    - Aplicar prueba de loops anidados.
  - No independientes.
    - Si el valor de salida del primer loop condiciona el del segundo.

# **PRUEBAS DE CAJA NEGRA**



# Pruebas de caja negra

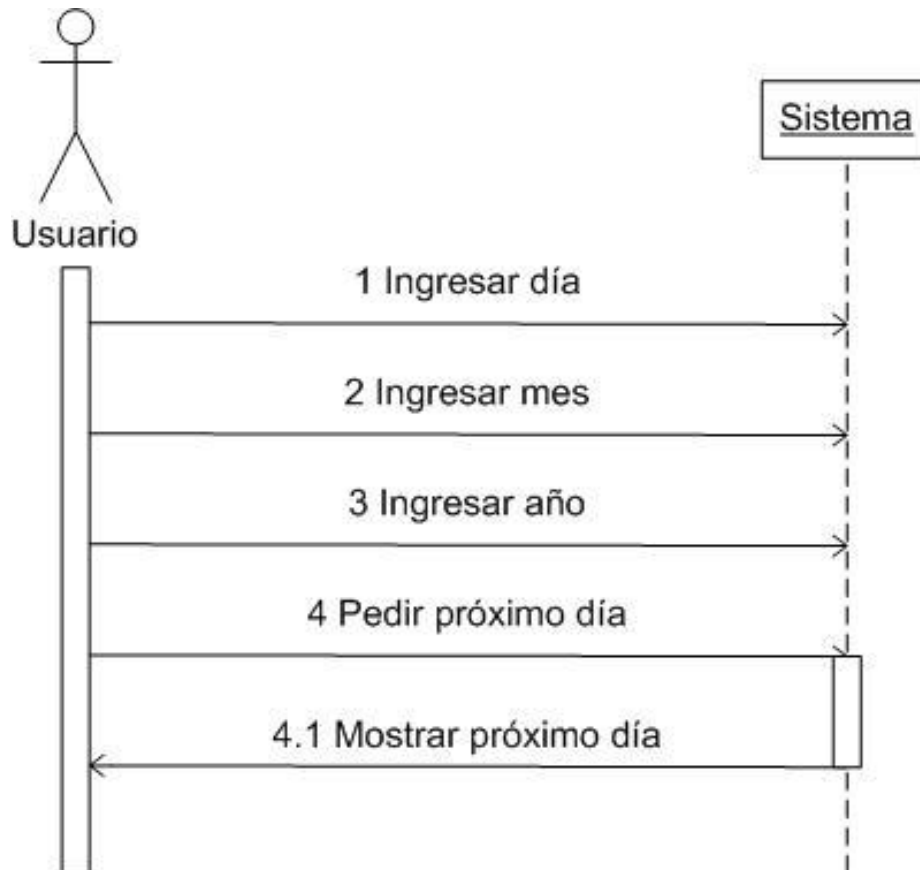
- Parten de la **especificación** del programa o módulo.
- Los casos de prueba generados deben ayudar a encontrar:
  - Funciones incorrectas o ausentes.
  - Errores en interfaz.
  - Errores en acceso a base de datos.
  - Errores de desempeño.
  - Errores de inicialización y finalización.

# Pruebas de caja negra

- Basados en escenarios o uso del sistema
  - Se especifica el caso de uso.
  - Se especifican distintos valores para las distintas variables de los casos de uso.
  - Se generan n casos de prueba por caso de uso.
  - Para la especificación de valores se puede usar:
    - Partición equivalente.
    - Análisis de valores límites.

# Pruebas de caja negra

- Dada una fecha (DD/MM/AAAA) determinar cual es el día siguiente.



# Pruebas de caja negra

- Partición Equivalente
  - Un conjunto de pruebas forma una clase de equivalencia si:
    - Todos los casos prueban lo mismo
    - Si uno encuentra un defecto el otro también
    - Si uno no encuentra un defecto el otro tampoco

# Pruebas de caja negra

- Partición Equivalente
  - *Clase de equivalencia*: representa un conjunto de datos válidos y otro de datos no válidos. Ej.: 100 - 200, 300 - ...

# Pruebas de caja negra

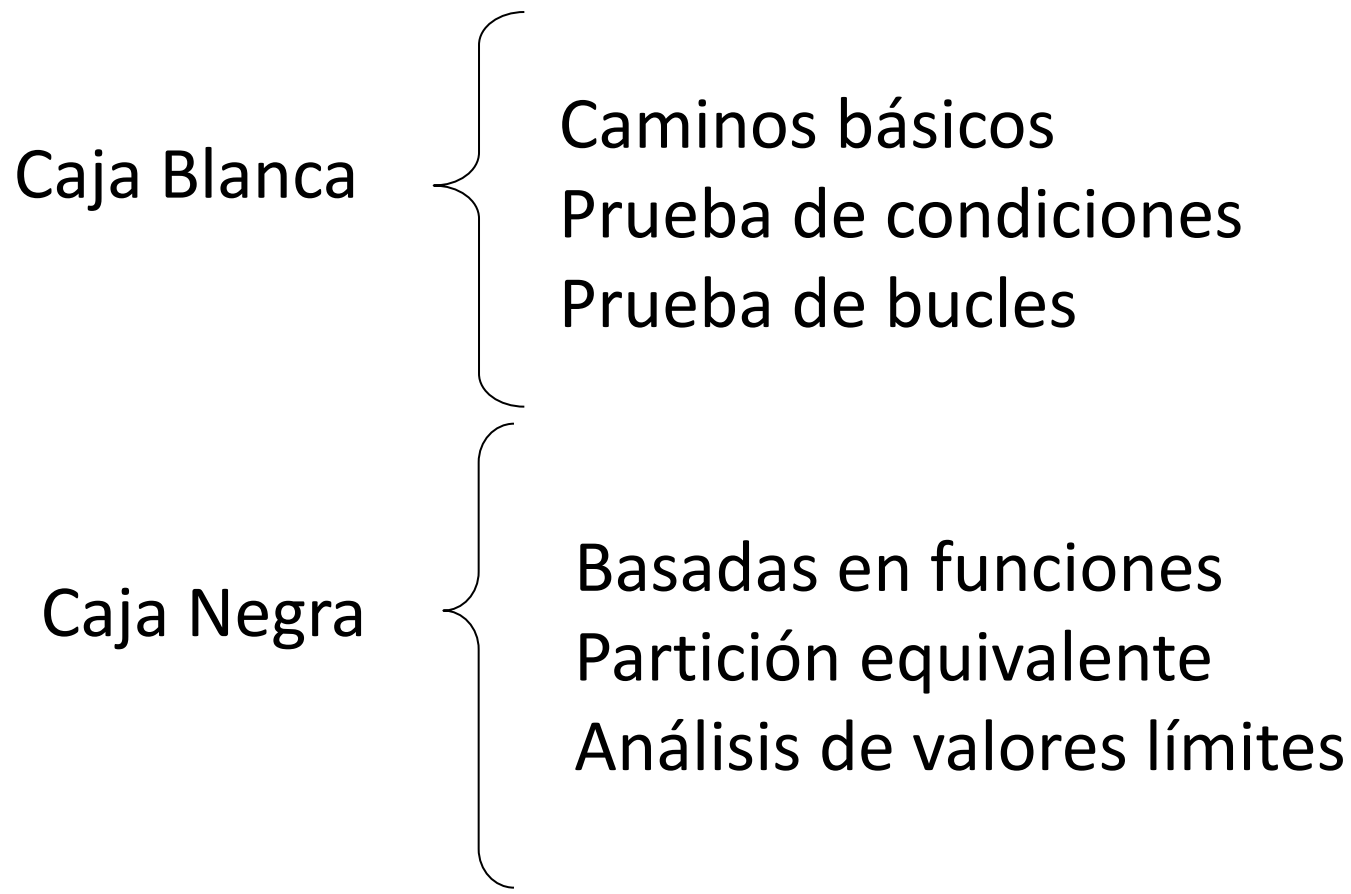
- Partición Equivalente
  - Para un programa que acepta cualquier número entre 1 y 99 las clases de equivalencia son:
    - Cualquier número entre 1 y 99 es valido
    - Cualquier número menor que 1 es invalido
    - Cualquier número mayor que 99 es invalido
    - Si no es un número es invalido

# Pruebas de caja negra

- Análisis de valores límites
  - Similar a Partición equivalente pero considerando los límites.
  - Reglas.
    - Si una condición de entrada es:
      - Rango entre a y b, diseñar casos de prueba para a y b, por debajo de a y por encima de b.
      - Nro. de valores, probar el máx. y mín, y los valores justo por encima del máx. y por debajo del mín.
  - Aplicar estas reglas para condiciones de salida y estructuras de datos.

# Técnicas de prueba

Caja Blanca



Caminos básicos  
Prueba de condiciones  
Prueba de bucles

Caja Negra

Basadas en funciones  
Partición equivalente  
Análisis de valores límites



# Otros tipos de prueba

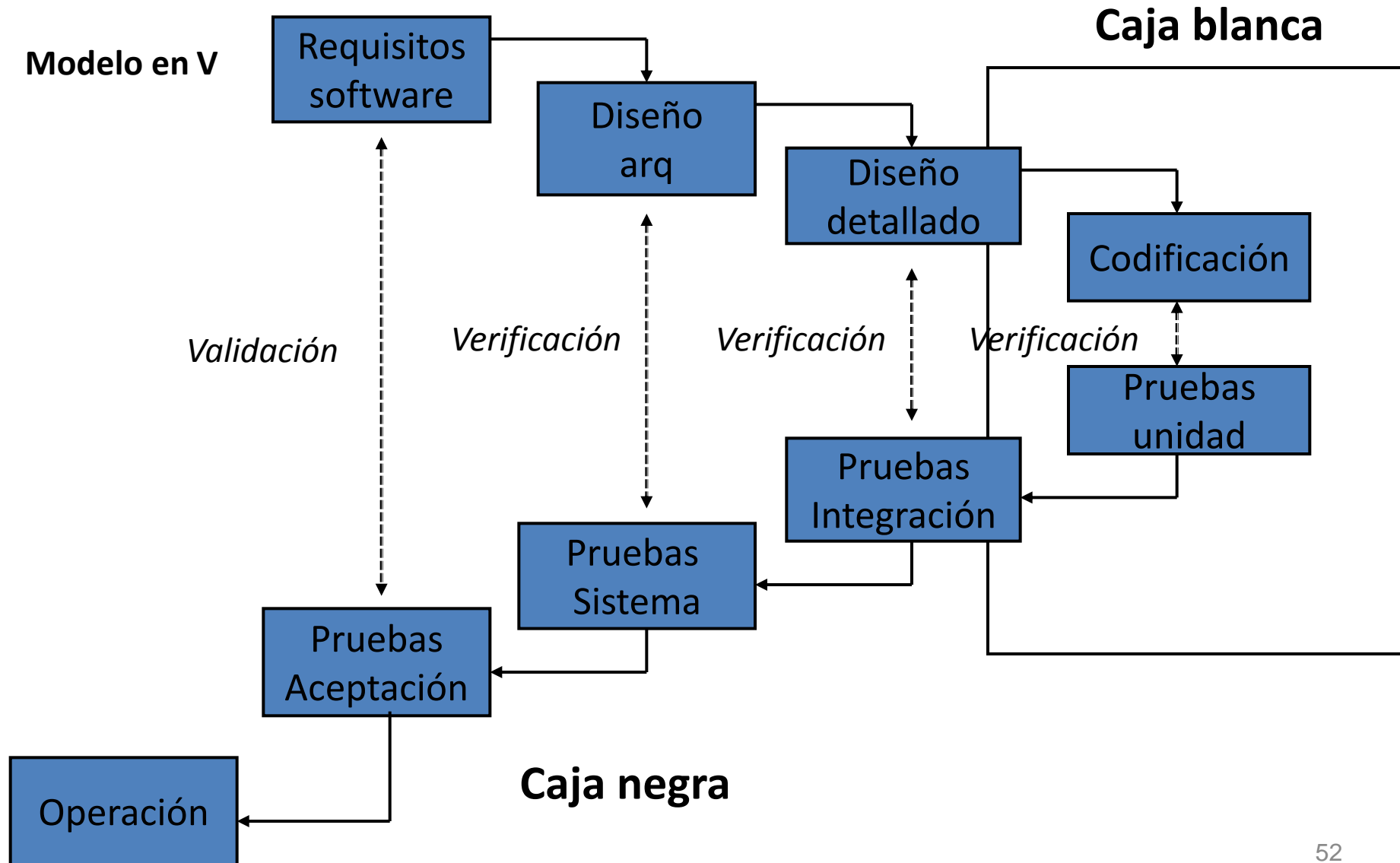
- Prueba de sobrecarga
- Prueba randómica
- Prueba exploratoria
- Prueba de GUI
- Prueba de usabilidad

# **PROCESO DE PRUEBA**

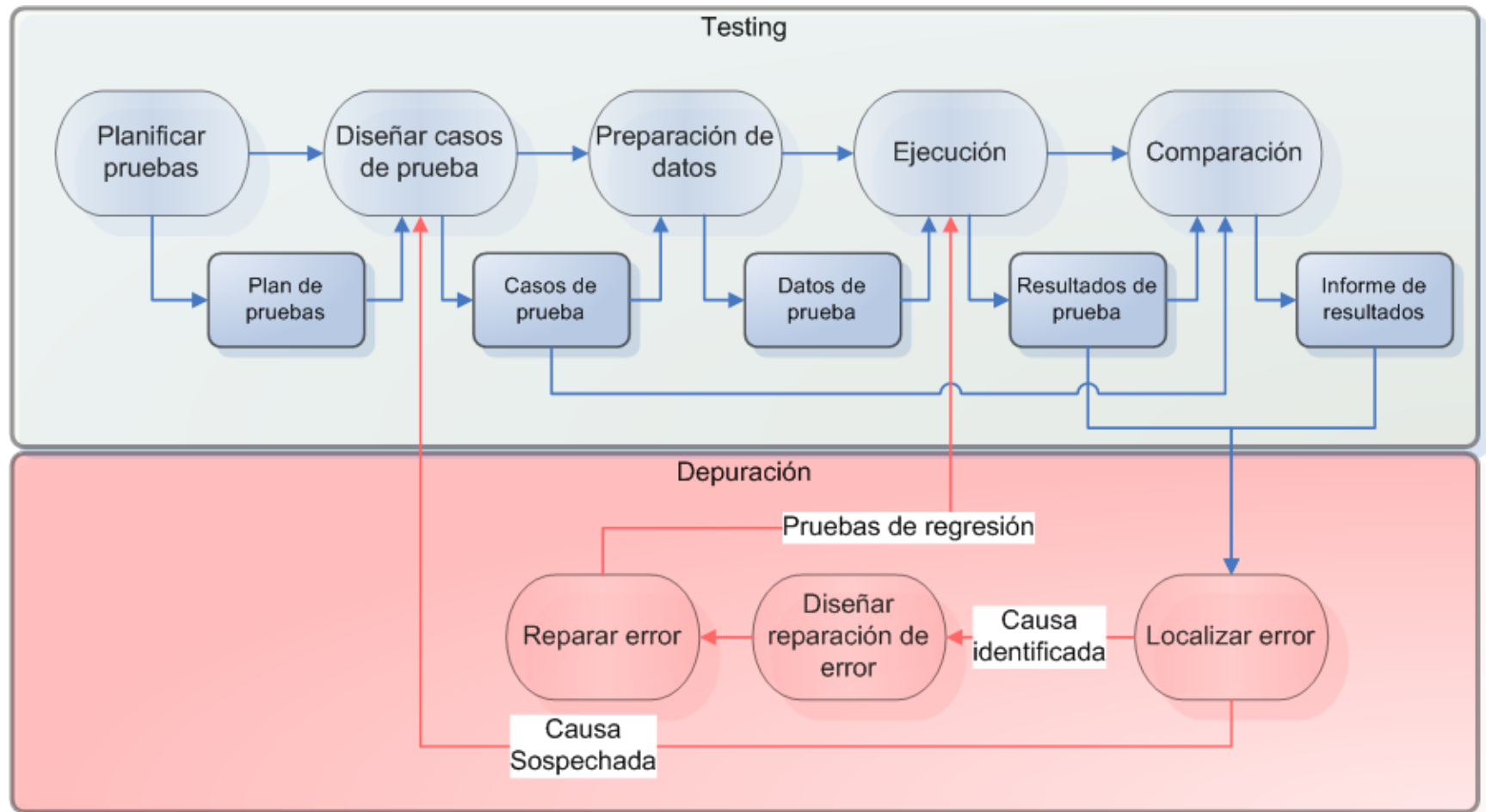
# Proceso de prueba

- Prueba de caja blanca vs caja negra  
No son antagónicas. Son complementarias.
- Caja Blanca.
  - Se aplica a la pruebas de bajo nivel del software.
  - Se aplica en las primeras fases de la prueba.
- Caja Negra.
  - Se aplica a las pruebas de alto nivel (desde el punto de vista del usuario).
  - Se aplica en las fases más tardías de la prueba.

# Proceso de prueba



# Proceso de prueba



# Proceso de prueba

- Plan de prueba
  - 1. Introducción
    - Propósito del documento, Alcance del plan
    - Audiencia, Terminología
  - 2. Misión de las pruebas
    - Descripción del proyecto
    - Misión de las pruebas durante el proyecto y sus fases
  - 3. Elementos a probar
    - Alcance de las pruebas
  - 4. Enfoque de las pruebas
    - Estrategias y niveles de prueba
    - Identificación y justificación de las pruebas

# Proceso de prueba

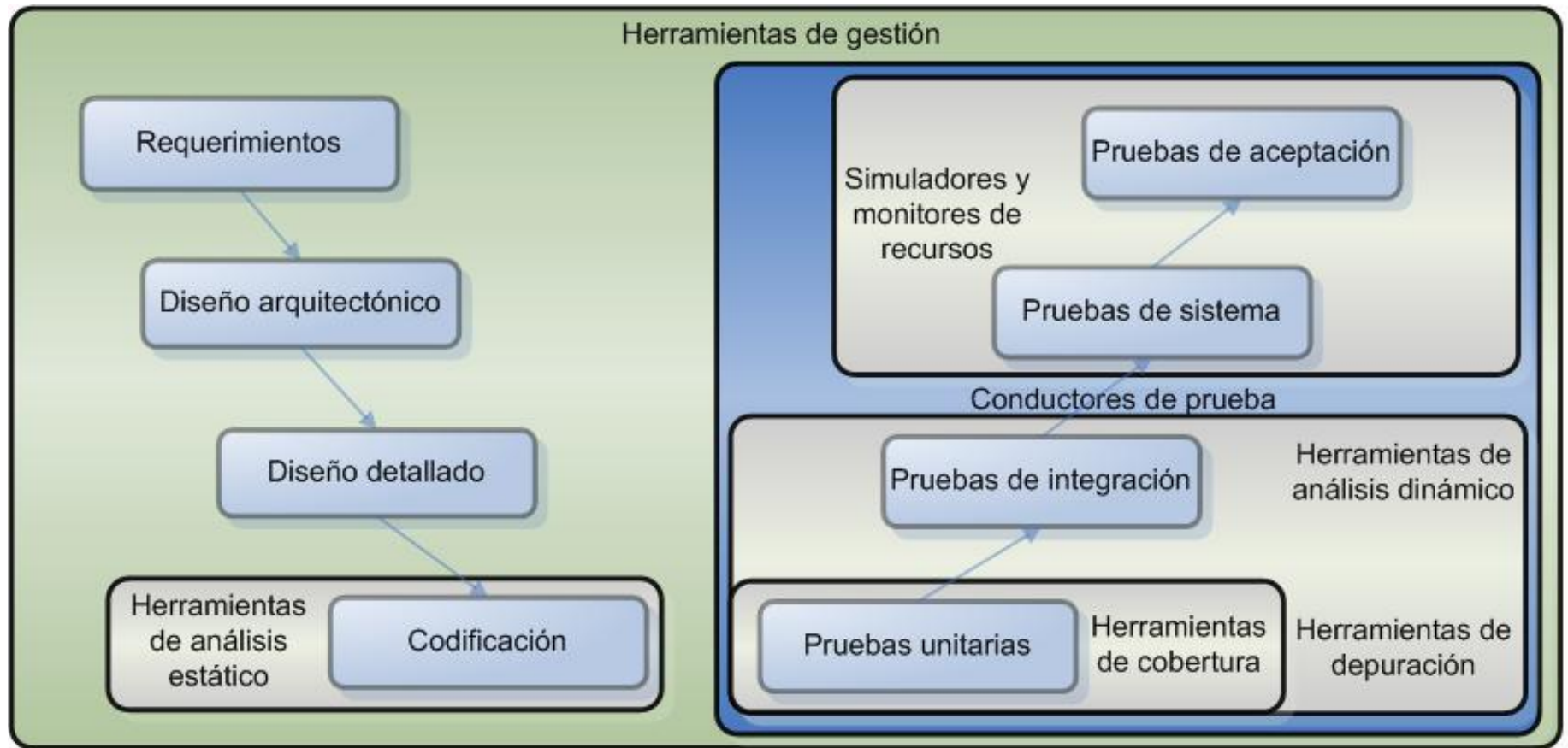
- Plan de prueba
  - 5. Entregables
    - Reportes de prueba
    - Mecanismos de comunicación de incidencias
  - 6. Proceso de prueba
    - Tareas de pruebas
  - 7. Ambientes de prueba
  - 8. Roles y responsabilidades
  - 9. Cronograma de pruebas

# Automatización pruebas

- Herramientas para automatizar la prueba
  - Análisis de código.
    - Estáticas.
    - Dinámicas.
  - Ejecución de pruebas.
  - Generación de casos de prueba.
  - Entornos automáticos de prueba.



# Automatización pruebas



# Proceso de prueba

- Condiciones para la prueba
  - Especificar requerimientos en forma cuantificable.
  - Desarrollar perfil de usuarios del sistema.
  - Explicitar los objetivos de la prueba.
    - Grado de cobertura.
    - Tiempo medio entre fallas.
    - Costo encontrar/depurar errores.
  - Construir software que se “autopruebe”.
  - Realizar revisiones formales de los CP y la prueba.
  - Mejora continua del proceso de prueba.

# Proceso de prueba

- ¿Cuándo Parar de Probar?
  - “La ausencia de evidencias no es evidencia de ausencia”.
  - La prueba nunca termina.
    - Termina probando el cliente.
  - Se termina cuando se consumió el presupuesto o el tiempo disponibles.
  - Otra solución: un modelo estadístico.

# Proceso de prueba

- Modelo estadístico
  - $f(t) = (1/p) \ln(I_0 p t + 1)$ .
  - $f(t)$  = nro. acumulado de defectos que se esperan durante la prueba en un tiempo  $t$ .
  - $I_0$  = intensidad de defectos en el inicio.
  - $p$  = reducción exponencial de intensidad de defectos a medida que se corrige.
  - $I(t) = I_0 / (I_0 p t + 1)$ . [derivada de  $f(t)$ ].

# Proceso de prueba

- Siembra de fallas
  - Permiten estimar número de fallas no detectados en un programa.
  - Insertan en el programa un número conocido de fallas (equipo independiente).
  - El equipo de desarrollo trata de encontrar las fallas.
  - El número de fallas no detectadas se utiliza como indicador de las fallas que restan por detectar (incluso las no sembradas).

# Proceso de prueba

- Siembra de fallas
  - Sostiene la relación:
    - $S/TS = NS/TNS$ , siendo:
      - $S$  = fallas sembradas detectadas
      - $TS$  = total fallas sembradas
      - $NS$  = fallas no sembradas detectadas
      - $TNS$  = total de fallas no sembradas
  - Índice de confianza:
    - $C = S/(S+NS+1)$ .

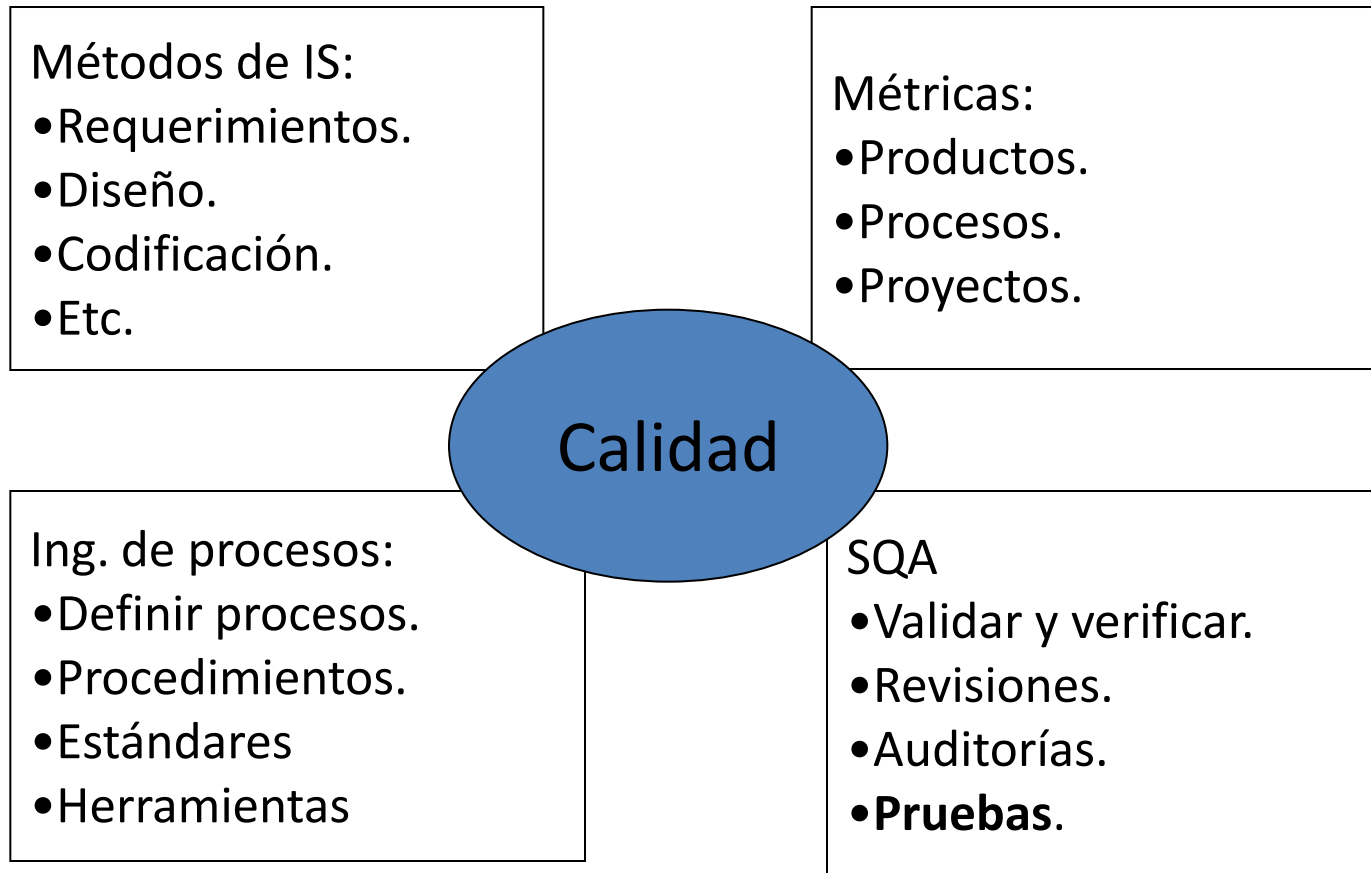
# **CONTEXTO DE LA CALIDAD**

# Contexto de la calidad

- La prueba no es el único elemento para **asegurar la calidad**, sino que es parte de una red.
  - La calidad no son solo controles sino una **filosofía de trabajo**.
  - La calidad no la asegura un equipo especializado, sino **todo el equipo** de trabajo.
  - La calidad no es algo que se compra sino algo que **se construye**.



# Contexto de la calidad



# Referencias

- Ingeniería del Software, 4ta. Edición, Pressman.
- Testing Computer software 2nd ed., Kaner et. Al
- Software Engineering, 6th ed., I. Sommerville.