

Pruebas de  
SOFTWARE



PRUEBAS DE SOFTWARE





## PRUEBAS DE SOFTWARE

De lo anterior, se puede decir que las pruebas de software son estrictamente necesarias, para determinar de manera dinámica la calidad del software; de esta manera, se garantiza que se ha construido el sistema correcto y de la forma correcta.

### ORIENTACIONES GENERALES

### ACTIVIDADES

### PRUEBAS

### DISEÑO DE CASOS DE PRUEBA

### DEPURACIÓN DE ERRORES

### DOCUMENTACIÓN DE PRUEBAS





## PRUEBAS DE SOFTWARE



### ORIENTACIONES GENERALES

- Realizar revisiones técnicas efectivas, antes de comenzar la prueba.
- La prueba comienza en los componentes del software y fluye hacia la integración de todo el sistema.
- Seleccionar y aplicar la mejor técnica de prueba, de acuerdo al enfoque de desarrollo utilizado.
- Diseñar casos y esbozar el plan de prueba en la fase de diseño del software.
- Un buen caso de prueba permite mostrar un error o un fallo no detectado anteriormente.
- Es fundamental conocer el resultado esperado para determinar si el resultado de la prueba es correcto o no.
- Procurar que en la prueba, a parte de los desarrolladores, participen programadores que no hayan estado en la fase de codificación y también potenciales usuarios.
- Así como se prueba que el programa funcione correctamente para entradas válidas, también es igualmente importante, comprobar que el programa reaccione correctamente ante entradas no válidas.
- Documentar todo los casos de prueba. Esto permite realizar pruebas de regresión.





## PRUEBAS DE SOFTWARE



### ACTIVIDADES

La siguiente figura presenta las principales actividades que se deben desarrollar en la etapa de prueba en el marco del desarrollo software.



*Tareas básicas en la prueba de software*



## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |           |                               |           |                               |
|-----------|-------------------------------|-----------|-------------------------------|
| <b>01</b> | <b>PRUEBAS DE CAJA BLANCA</b> | <b>05</b> | <b>PRUEBAS DE INTEGRACIÓN</b> |
| <b>02</b> | <b>PRUEBAS DE CAJA NEGRA</b>  | <b>06</b> | <b>PRUEBAS DE VALIDACIÓN</b>  |
| <b>03</b> | <b>ESTRATEGIA DE PRUEBAS</b>  | <b>07</b> | <b>PRUEBAS DEL SISTEMA</b>    |
| <b>04</b> | <b>PRUEBAS UNITARIAS</b>      | <b>08</b> | <b>PRUEBAS DE ACEPTACIÓN</b>  |



## PRUEBAS DE SOFTWARE

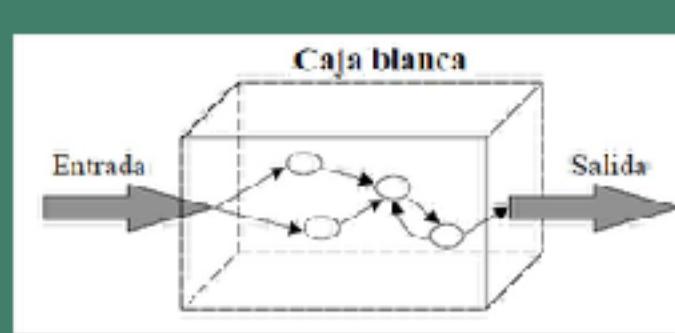


### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

Las pruebas de caja blanca, permiten probar la lógica interna del programa y su estructura, realizando las siguientes acciones:



- Ejecución de todas las sentencias (al menos una vez).
- Recorrido de todos los caminos independientes de cada componente.
- Comprobación de todas las decisiones lógicas.
- Comprobación de todos los bucles.
- Implementación de situaciones extremas o límites.

**PRUEBA DE CAMINO BÁSICO**

**PRUEBA DE BUCLE**

**PRUEBA DE CONDICIÓN**

**PRUEBA DE ESTRUCTURA DE DATOS LOCALES**



## PRUEBAS DE SOFTWARE



### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CAMINO BÁSICO

1 2 3 4

*Esta técnica fue propuesta por Tom McCabe y consiste en definir un conjunto básico de caminos usando la medida de complejidad llamada complejidad ciclomática (VG). La complejidad ciclomática determina el número de caminos a probar, mediante la siguiente fórmula:*

$$V(G) = \#Aristas - \#Nodos + 2$$

*Los pasos en las pruebas de caminos básicos son:*

1. Dibujar grafo de flujo.
2. Determinar la complejidad ciclomática del grafo.
3. Determinar los caminos linealmente independientes.
4. Diseñas los casos de prueba.



## PRUEBAS DE SOFTWARE

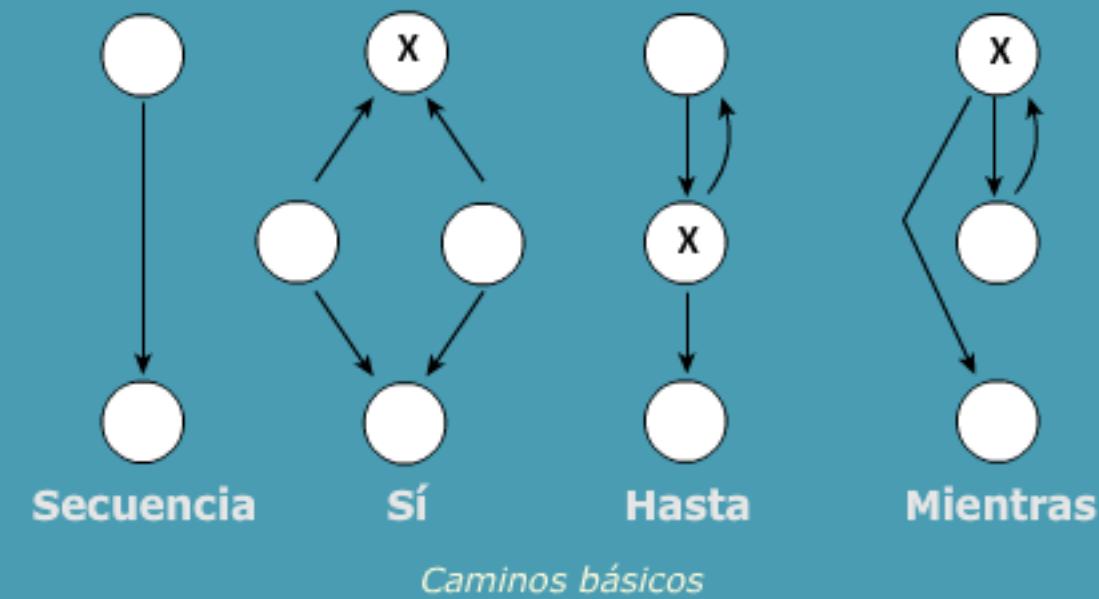
### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CAMINO BÁSICO

1 2 3 4



Ejemplo:

Realizar la prueba de camino básico para el siguiente módulo:

```
Public intMCD(int x, int y)
{
    While (x != y) → a
    {
        If (x > y)      → b
        x = x - y;     → c
        else
            y = y - x; → d
    }                   → e
    return x;          → f
}
```



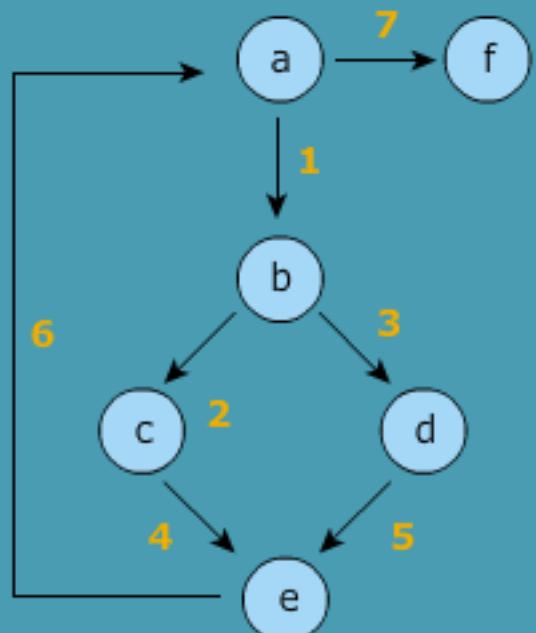
## PRUEBAS DE SOFTWARE

### PRUEBAS

**01**

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CAMINO BÁSICO

**1 2 3 4****1. Grafo de flujo:****2. Complejidad Ciclomática:**

$$V(G) = \# \text{Aristas} - \# \text{Nodos} + 2$$

$$V(CDM) = 7 - 6 + 2 = 3$$



## PRUEBAS DE SOFTWARE

### PRUEBAS

**01**

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CAMINO BÁSICO

**1 2 3 4**

##### 3. Caminos linealmente independientes:

Como la complejidad ciclomática es 3, entonces existen tres caminos linealmente independientes.

##### 4. Casos de prueba:

Caminos	Aristas							Casos de Prueba
	1	2	3	4	5	6	7	
af	0	0	0	0	0	0	1	x=1, y=1, return=1
abdeaf	1	0	1	0	1	1	1	x=1, y=2, return=1
abceaf	1	1	0	1	0	1	1	x=2, y=1, return=1

Casos de prueba ejemplo de camino básico



## PRUEBAS DE SOFTWARE

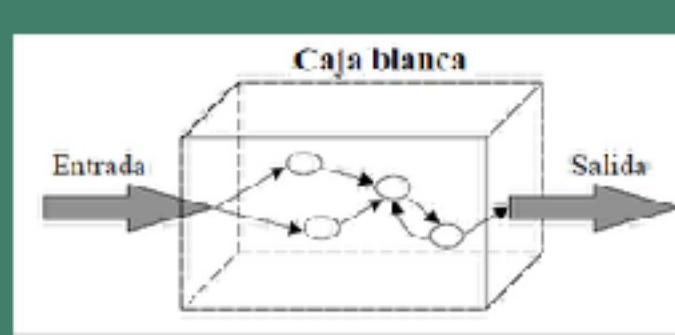


### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

Las pruebas de caja blanca, permiten probar la lógica interna del programa y su estructura, realizando las siguientes acciones:



- Ejecución de todas las sentencias (al menos una vez).
- Recorrido de todos los caminos independientes de cada componente.
- Comprobación de todas las decisiones lógicas.
- Comprobación de todos los bucles.
- Implementación de situaciones extremas o límites.

**PRUEBA DE CAMINO BÁSICO**

**PRUEBA DE CONDICIÓN**

**PRUEBA DE BUCLE**

**PRUEBA DE ESTRUCTURA DE DATOS LOCALES**



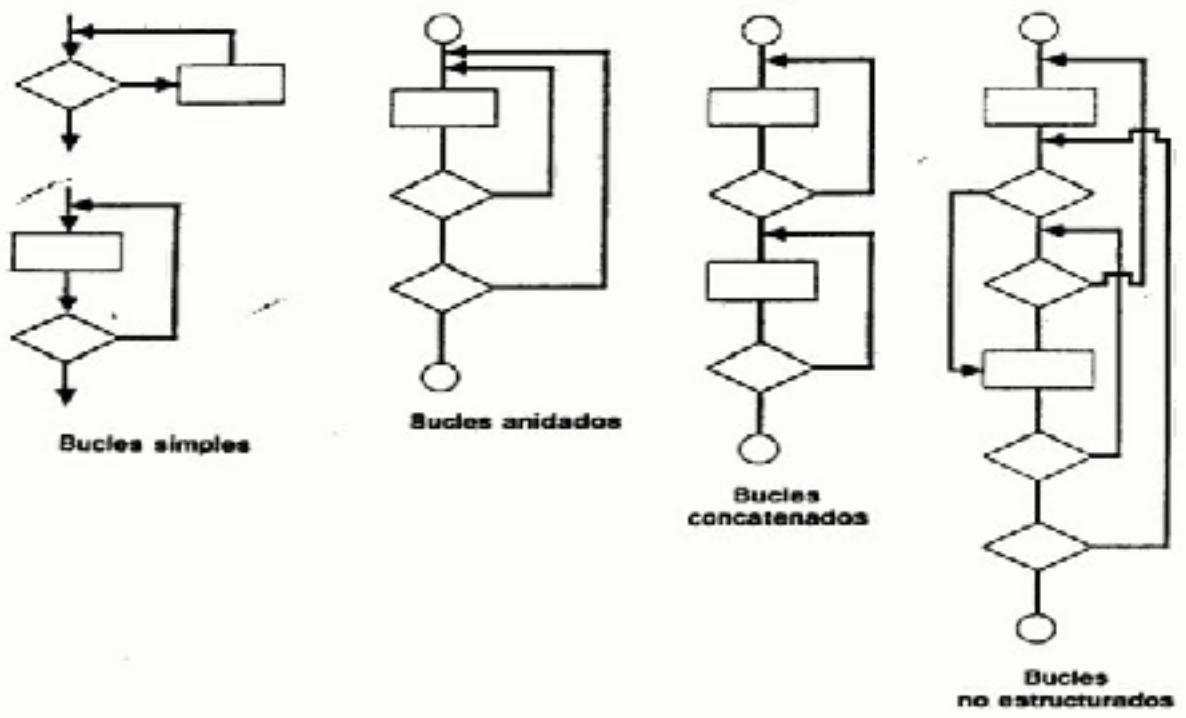
## PRUEBAS DE SOFTWARE

### PRUEBAS

#### 01 PRUEBAS DE CAJA BLANCA PRUEBA DE BUCLE

1 2 3 4

Para dicha prueba se requiere, en primer lugar, representar de forma gráfica los bucles, que pueden ser simples, anidados, concatenados y no estructurados, como lo muestra la siguiente figura.





## PRUEBAS DE SOFTWARE

### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE BUCLE

1 2 3 4

###### *Bucles Simples*

Con bucles simples, se requieren las siguientes iteraciones, siendo n el número máximo de pasos:

- Saltar por completo el bucle.
- Pasar una sola vez a través del bucle.
- Pasar dos veces a través del bucle.
- Pasar m veces a través del bucle, donde m < n.
- Hacer n-1, n y n+1 pasadas a través del bucle.



## PRUEBAS DE SOFTWARE



### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE BUCLE

1 2 3 4

###### *Bucles anidados*

En bucles anidados, se requieren los siguientes pasos:

- Comenzar con el bucle más interno.
- Realizar pruebas de bucle simple para el bucle más interno.
- Avanzar hacia afuera y realizar pruebas con el siguiente bucle.
- Continuar hasta que todos los bucles hayan sido probados.



## PRUEBAS DE SOFTWARE



### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE BUCLE

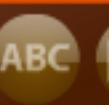
1 2 3 4

###### *Bucles concatenados*

Si los bucles concatenados son independientes se aplica pruebas de bucles simples, si no independientes se implementa el enfoque de los bucles anidados.

###### *Bucles no estructurados*

Los bucles no estructurados deben ser rediseñados porque comprometen la calidad del diseño. Posteriormente, se realizan pruebas de acuerdo al tipo de bucle resultante.



X

## PRUEBAS DE SOFTWARE

X

### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

Esta prueba evalúa las condiciones lógicas contenidas en un módulo del programa, las cuales pueden ser *simples* o *compuestas*.

Condición simple: es una variable lógica (TRUE o FALSE), o una expresión relacional de la forma: E1<operador relacional>E2, donde E1 Y E2 son expresiones aritmética, y el operador relacional es uno de los siguientes: <, >, <=, >=, =, !=.

Condición compuesta: se compone de dos o más condiciones simples y operadores lógicos de tipo NOT, AND, OR y paréntesis.





## PRUEBAS DE SOFTWARE



### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

1 **2** 3 4 5 6 7 8 9

En general, en la prueba de condición existen los siguientes tipos:

- De cobertura de decisión.
- De cobertura de condición.
- De cobertura de decisión/condición.

Ejemplo: Definir casos de prueba aplicando cobertura de decisión y de decisión/condición para el siguiente fragmento de código:



## PRUEBAS DE SOFTWARE



### PRUEBAS

#### PRUEBAS DE CAJA BLANCA 01 PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

```
public void comprobarhora (int h, int m, int s)
{
    If( (h>=0) && (h<=23) )
    {
        If( (m>=0) && (<=59) )
        {
            If( (s>=0) && (s<=50) )
            {
                System.out.println("La hora digitada es correcta");
            }
        }
    else
    {
        System.out.println("La hora digitada es incorrecta");
    }
}
```



## PRUEBAS DE SOFTWARE



### PRUEBAS

**01**

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

##### 1. Casos de prueba para cobertura de decisiones:

En el código hay tres decisiones.

D1 ( $h \geq 0$ ) y ( $h \leq 23$ )

D2 ( $m \geq 0$ ) y ( $m \leq 59$ )

D3 ( $s \geq 0$ ) y ( $s \leq 59$ )

##### 2. Datos concretos para los casos de prueba:

Cada decisión debe tomar al menos una vez el valor verdadero y otra el valor falso.

Caso	Valor Verdadero	Valor Falso
D1	$h=8$	$h=25$
D2	$m=25$	$m=60$
D3	$s=59$	$s=75$

*Datos de prueba de decisión para ejemplo*



## PRUEBAS DE SOFTWARE



### PRUEBAS

**01**

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

##### 3. Casos de prueba para cubrir todas las decisiones:

Caso de prueba 1: D1 = Verdadero; D2= Verdadero; D3=Verdadero  
(h=8; m=25; s=59)

Caso de prueba 2: D1 = Verdadero; D2= Verdadero; D3=Falso  
(h=8; m=25; s=75)

Caso de prueba 3: D1 = Verdadero; D2= Falso  
(h=8; m=60)

Caso de prueba 4: D1 = Falso  
(h=25)



## PRUEBAS DE SOFTWARE



### PRUEBAS

#### PRUEBAS DE CAJA BLANCA

**01**

##### PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

1

2

3

4

5

6

7

8

9

#### 4. Casos de prueba para obtener una cobertura total de decisión/condición:

En el código hay tres decisiones y cada una contiene dos condiciones.

D1 ( $h \geq 0$ ) y ( $h \leq 23$ )

C1.1  $h \geq 0$

C1.2  $h \leq 23$

D2 ( $m \geq 0$ ) y ( $m \leq 59$ )

C2.1  $m \geq 0$

C2.2  $m \leq 59$

D3 ( $s \geq 0$ ) y ( $s \leq 59$ )

C3.1  $s \geq 0$

C3.2  $s \leq 59$



## PRUEBAS DE SOFTWARE

### PRUEBAS

**01**

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

**1 2 3 4 5 6 7 8 9**

5. Datos concretos para los casos de prueba:

Caso	Valor Verdadero	Valor Falso
C1.1	$h=10$	$h=-1$
C1.2	$h=10$	$h=24$
C2.1	$m=30$	$m=-1$
C2.2	$m=30$	$m=60$
C3.1	$s=50$	$s=-1$
C3.2	$s=50$	$s=70$

Datos de prueba de decisión/condición para ejemplo.



X

## PRUEBAS DE SOFTWARE



### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

###### Caso de prueba 1:

C1.1=Verdadero; C1.2=Verdadero;  
C2.1=Verdadero; C2.2=Verdadero;  
C3.1=Verdadero; C3.2=Verdadero;  
(h=10; m=30; s=50)

###### Caso de prueba 2:

C1.1=Verdadero; C1.2=Verdadero;  
C2.1=Verdadero; C2.2=Verdadero;  
C3.1=Verdadero; C3.2=Falso;  
(h=10; m=30; s=70)

###### Caso de prueba 3:

C1.1=Verdadero; C1.2=Verdadero;  
C2.1=Verdadero; C2.2=Verdadero;  
C3.1=Falso; C3.2=Verdadero;  
(h=10; m=30; s=-1)

###### Caso de prueba 4:

C1.1=Verdadero; C1.2=Verdadero;  
C2.1=Verdadero; C2.2=Falso;  
(h=10; m=30)

1 2 3 4 5 6 7 8 9



## PRUEBAS DE SOFTWARE

### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE CONDICIÓN

1 2 3 4 5 6 7 8 9

**Caso de prueba 5:**

C1.1=Verdadero; C1.2=Verdadero;  
C2.1=Falso; C2.2=Verdadero;  
(h=10; m=-1)

**Caso de prueba 6:**

C1.1=Verdadero; C1.2=Falso;  
(h=10)

**Caso de prueba 7:**

C1.1=Falso; C1.2=Verdadero;  
(h=-1)



## PRUEBAS DE SOFTWARE

### PRUEBAS

01

#### PRUEBAS DE CAJA BLANCA

##### PRUEBA DE ESTRUCTURA DE DATOS LOCALES

Estas pruebas aseguran la integridad de los datos durante todos los pasos de la ejecución del módulo. Se revisa que se cumpla:

- Integridad de datos.
- Inicialización de todas las variables utilizadas.
- Uso adecuado de los límites en los arreglos.
- Declaración correcta de variable.
- Comparaciones entre variables del mismo tipo.
- Control de errores como overflow, underflow, división por cero, etc.





## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- 01 PRUEBAS DE CAJA BLANCA
  - 02 PRUEBAS DE CAJA NEGRA
  - 03 ESTRATEGIA DE PRUEBAS
  - 04 PRUEBAS UNITARIAS
  - 05 PRUEBAS DE INTEGRACIÓN
  - 06 PRUEBAS DE VALIDACIÓN
  - 07 PRUEBAS DEL SISTEMA
  - 08 PRUEBAS DE ACEPTACIÓN



## PRUEBAS DE SOFTWARE



### PRUEBAS

02

#### PRUEBAS DE CAJA NEGRA

En este tipo de pruebas se considera el software como una caja negra, sin preocuparse por los detalles procedimentales de los programas, como lo ilustra la figura. De esta manera, los datos de entrada deben generar una salida coherente con las especificaciones; si no es así, es porque se ha encontrado un error el cual debe ser corregido para poder continuar con las pruebas.

Las dos técnicas más comunes son:

**PARTICIÓN DE EQUIVALENCIA**

**ANÁLISIS DE VALORES LÍMITE**

ENTRADAS



FUNCIONES



SALIDAS



## PRUEBAS DE SOFTWARE



### PRUEBAS

**02**

#### PRUEBAS DE CAJA NEGRA

##### PARTICIÓN DE EQUIVALENCIA

En este tipo de prueba, la entrada de un programa se divide en clases de datos de los cuales se puedan derivar casos de prueba, teniendo en cuenta las siguientes reglas:

- Si la entrada es un rango o un valor específico, se define una clase de equivalencia válida y dos inválidas.
- Si la entrada es un valor lógico, se define una clase de equivalencia válida y otra inválida.





## PRUEBAS DE SOFTWARE



### PRUEBAS

02

#### PRUEBAS DE CAJA NEGRA

En este tipo de pruebas se considera el software como una caja negra, sin preocuparse por los detalles procedimentales de los programas, como lo ilustra la figura. De esta manera, los datos de entrada deben generar una salida coherente con las especificaciones; si no es así, es porque se ha encontrado un error el cual debe ser corregido para poder continuar con las pruebas.

Las dos técnicas más comunes son:

**PARTICIÓN DE EQUIVALENCIA**

**ANÁLISIS DE VALORES LÍMITE**

ENTRADAS



FUNCIONES



SALIDAS



## PRUEBAS DE SOFTWARE



### PRUEBAS

#### PRUEBAS DE CAJA NEGRA

02

En este tipo de pruebas se considera el software como una caja negra, sin preocuparse por los detalles procedimentales de los programas, como lo ilustra la figura. De esta manera, los datos de entrada deben generar una salida coherente con las especificaciones; si no es así, es porque se ha encontrado un error el cual debe ser corregido para poder continuar con las pruebas.

Las dos técnicas más comunes son:

**PARTICIÓN DE EQUIVALENCIA**

**ANÁLISIS DE VALORES LÍMITE**

ENTRADAS



FUNCIONES



SALIDAS



## PRUEBAS DE SOFTWARE

### PRUEBAS

02

#### PRUEBAS DE CAJA NEGRA

##### ANÁLISIS DE VALORES LÍMITE

Después de probar todos los casos en la técnica de participación de equivalencia, se prueban los valores fronterizos de cada clase (valores límite).





## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |           |                               |           |                               |
|-----------|-------------------------------|-----------|-------------------------------|
| <b>01</b> | <b>PRUEBAS DE CAJA BLANCA</b> | <b>05</b> | <b>PRUEBAS DE INTEGRACIÓN</b> |
| <b>02</b> | <b>PRUEBAS DE CAJA NEGRA</b>  | <b>06</b> | <b>PRUEBAS DE VALIDACIÓN</b>  |
| <b>03</b> | <b>ESTRATEGIA DE PRUEBAS</b>  | <b>07</b> | <b>PRUEBAS DEL SISTEMA</b>    |
| <b>04</b> | <b>PRUEBAS UNITARIAS</b>      | <b>08</b> | <b>PRUEBAS DE ACEPTACIÓN</b>  |





## PRUEBAS DE SOFTWARE

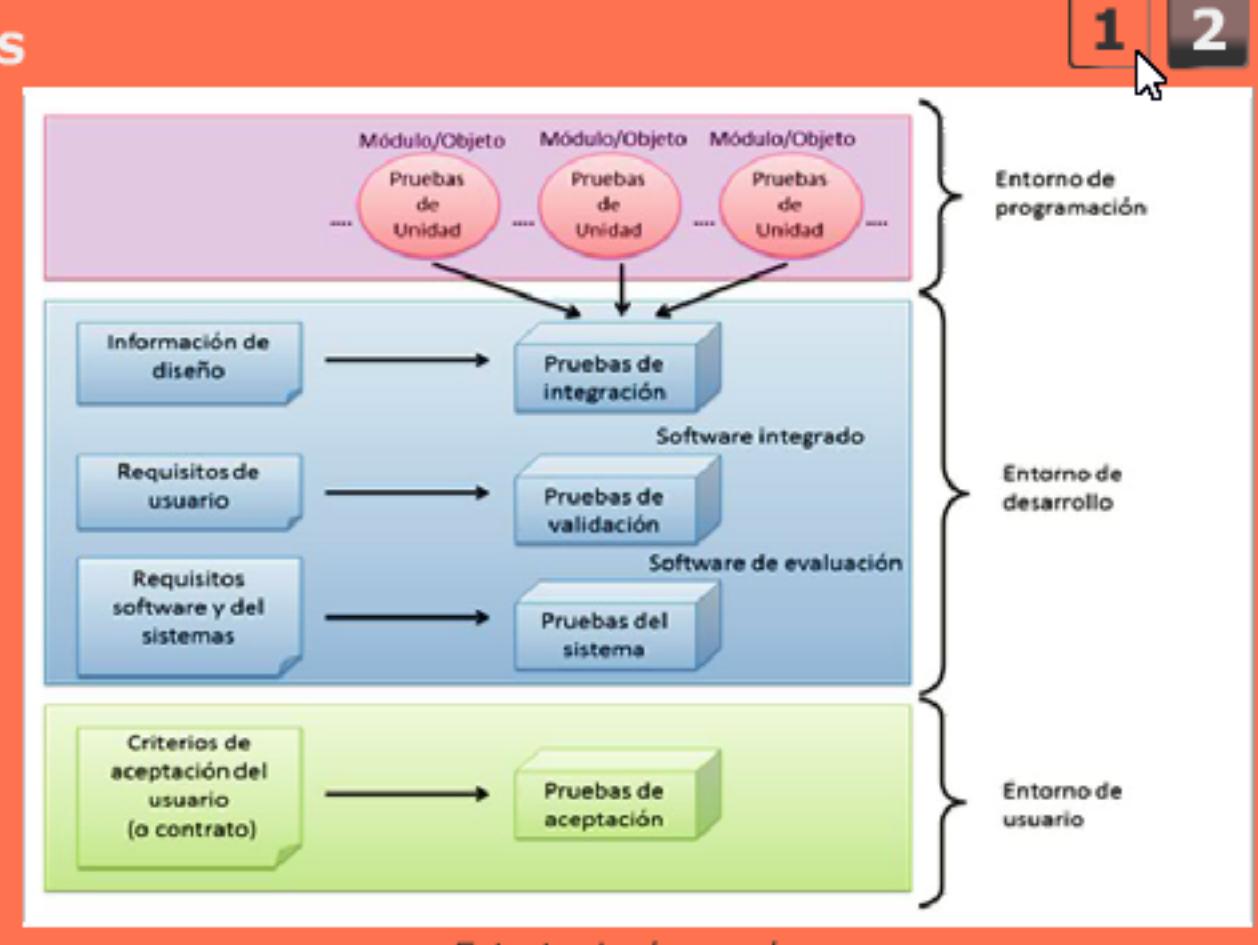


### PRUEBAS

#### ESTRATEGIA DE PRUEBAS

03

La prueba de software se realiza desde dentro hacia fuera del sistema, como lo ilustra la siguiente figura.



Estrategia de pruebas



## PRUEBAS DE SOFTWARE



### PRUEBAS

03

#### ESTRATEGIA DE PRUEBAS

1

2

- **Pruebas unitarias o de unidad:** se comprueba la lógica, funcionalidad y especificación de cada unidad (componente, clase u objeto) del sistema de manera aislada con respecto al resto de unidades.
- **Pruebas de integración:** se centra en el diseño y la construcción de arquitectura del software, analizando el flujo de información entre unidades a través de las interfaces.
- **Pruebas de validación:** se comprueba el cumplimiento de los requisitos.
- **Pruebas del sistema:** se prueba el software y otros elementos del sistema como un todo.
- **Pruebas de aceptación:** el usuario valida que el producto se ajusta a sus requerimientos.



## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- 01 PRUEBAS DE CAJA BLANCA
  - 02 PRUEBAS DE CAJA NEGRA
  - 03 ESTRATEGIA DE PRUEBAS
  - 04 PRUEBAS UNITARIAS
  - 05 PRUEBAS DE INTEGRACIÓN
  - 06 PRUEBAS DE VALIDACIÓN
  - 07 PRUEBAS DEL SISTEMA
  - 08 PRUEBAS DE ACEPTACIÓN





## PRUEBAS DE SOFTWARE

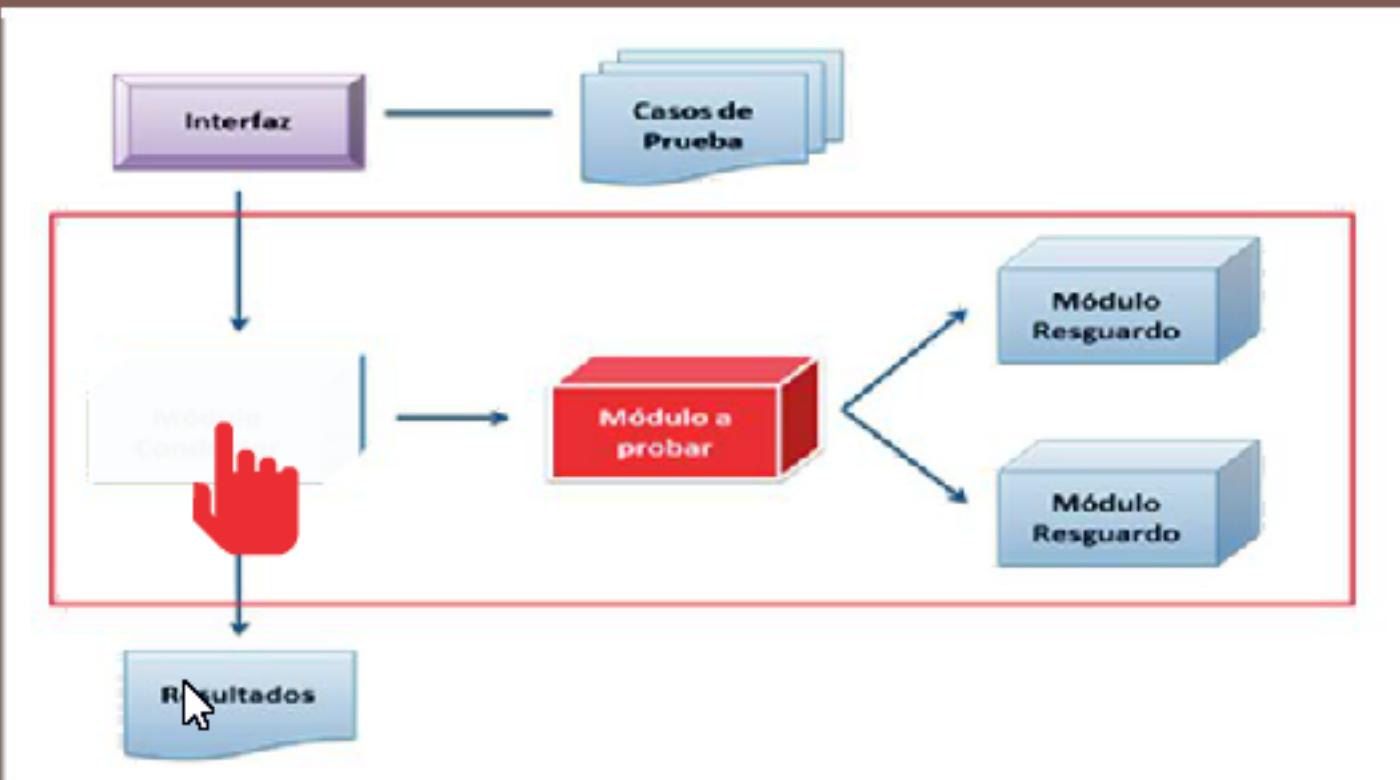


### PRUEBAS

#### 04

#### PRUEBAS UNITARIAS

Corresponden a la evaluación de cada uno de los bloques más pequeños con identidad propia en el sistema, y es realizada por el programador en su entorno de desarrollo. Las pruebas unitarias usan técnicas de caja blanca, para lo cual se crean módulos conductores y módulos resguardo.



Entorno de prueba unitaria.



## PRUEBAS DE SOFTWARE

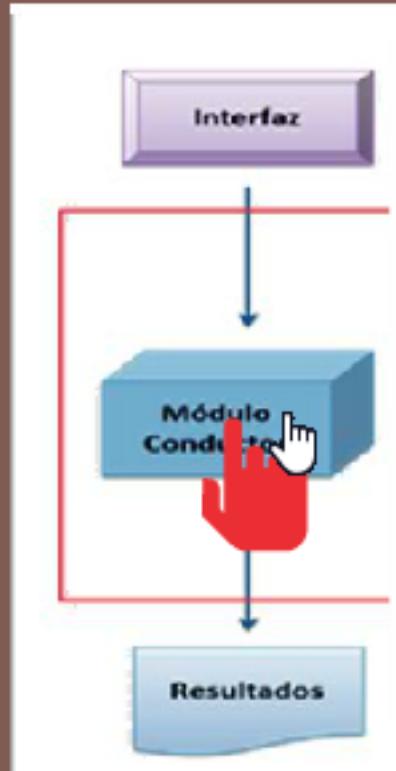


### PRUEBAS

#### 04

#### PRUEBAS UNITARIAS

Corresponden a la evaluación de cada uno de los bloques más pequeños con identidad propia en el sistema, y es realizada por el programador en su entorno de desarrollo. Las pruebas unitarias usan técnicas de caja blanca, para lo cual se crean módulos conductores y módulos resguardo.



Un módulo conductor o controlador es un *"programa principal"* que recibe a través de la interfaz datos de caso de prueba, y los pasa al componente que se desea probar. Los módulos resguardo (en inglés stubs) o *"subprogramas tontos"*, se utilizan para sustituir componentes que son subordinados o llamados desde el componente que se desea probar. Finalmente se generan reportes con los resultados obtenidos.

Entorno de prueba unitaria.



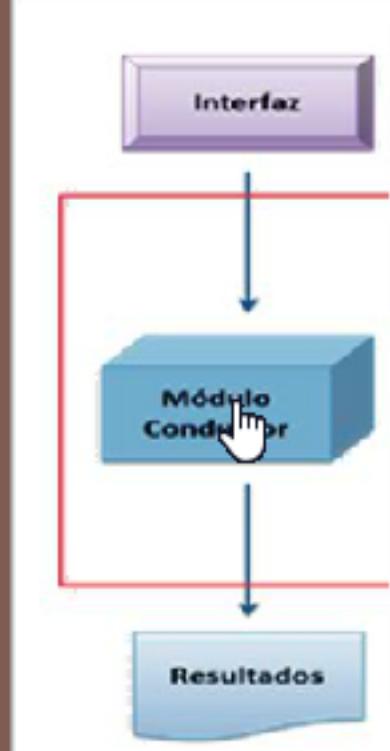
## PRUEBAS DE SOFTWARE

### PRUEBAS

#### 04

Corresponden a la evaluación de cada uno de los bloques más pequeños con identidad propia en el sistema, y es realizada por el programador en su entorno de desarrollo. Las pruebas unitarias usan técnicas de caja blanca, para lo cual se crean módulos conductores y módulos resguardo.

#### PRUEBAS UNITARIAS



Un módulo conductor o controlador es un “*programa principal*” que recibe a través de la interfaz datos de caso de prueba, y los pasa al componente que se desea probar. Los módulos resguardo (en inglés stubs) o “*subprogramas tontos*”, se utilizan para sustituir componentes que son subordinados o llamados desde el componente que se desea probar. Finalmente se generan reportes con los resultados obtenidos.

Entorno de prueba unitaria.



## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |           |                               |           |                               |
|-----------|-------------------------------|-----------|-------------------------------|
| <b>01</b> | <b>PRUEBAS DE CAJA BLANCA</b> | <b>05</b> | <b>PRUEBAS DE INTEGRACIÓN</b> |
| <b>02</b> | <b>PRUEBAS DE CAJA NEGRA</b>  | <b>06</b> | <b>PRUEBAS DE VALIDACIÓN</b>  |
| <b>03</b> | <b>ESTRATEGIA DE PRUEBAS</b>  | <b>07</b> | <b>PRUEBAS DEL SISTEMA</b>    |
| <b>04</b> | <b>PRUEBAS UNITARIAS</b>      | <b>08</b> | <b>PRUEBAS DE ACEPTACIÓN</b>  |



## PRUEBAS DE SOFTWARE



### PRUEBAS

05

#### PRUEBAS DE INTEGRACIÓN

No es suficiente con probar el funcionamiento correcto de cada módulo, se requiere además del funcionamiento en conjunto. Por tal razón, se hacen necesarias las pruebas de integración, las cuales generalmente implementan técnicas de caja negra.

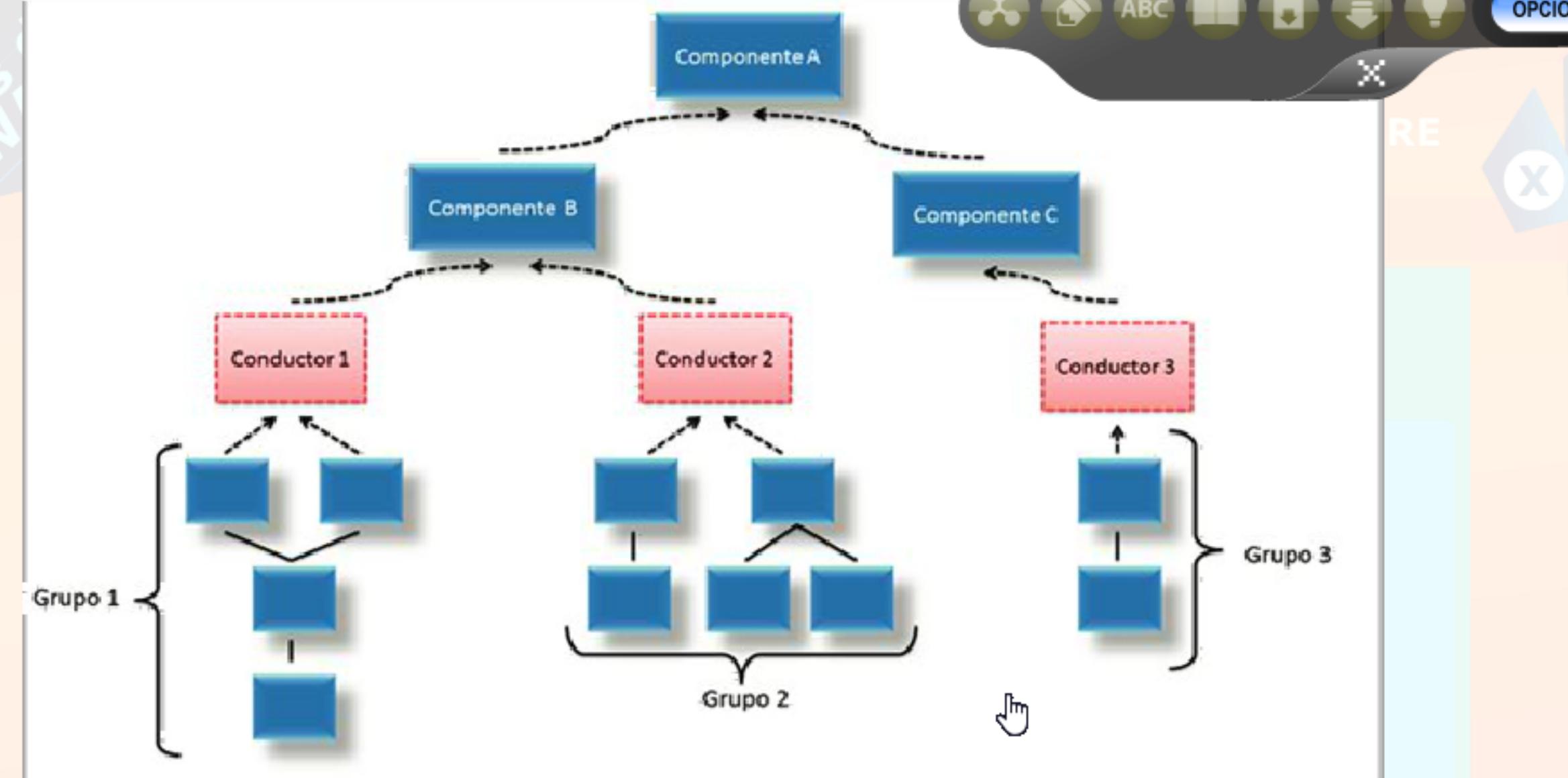
Existen dos estrategias básicas para realizar la combinación de módulos en las pruebas de integración: integración ascendente e integración descendente.

**Pruebas de integración ascendente**

**Pruebas de integración descendente**

**Pruebas de integración sandwich**

**Pruebas de regresión**



hacia arriba según la estructura del

En la Figura se ilustra la integración ascendente, en donde componentes de nivel inferior se combinan para formar los grupos 1, 2 y 3. El grupo 1 se prueba usando el conductor 1, el grupo 2 usando el conductor 2 y el grupo 3 con el conductor 3. Los conductores 1 y 2 se remueven para que los grupos se pongan en interfaz directa con el componente B. De manera similar, se hace para el grupo 3.

Finalmente los componentes B y C se integran con el componente A, y así sucesivamente según la arquitectura del programa.



## PRUEBAS DE SOFTWARE

### PRUEBAS

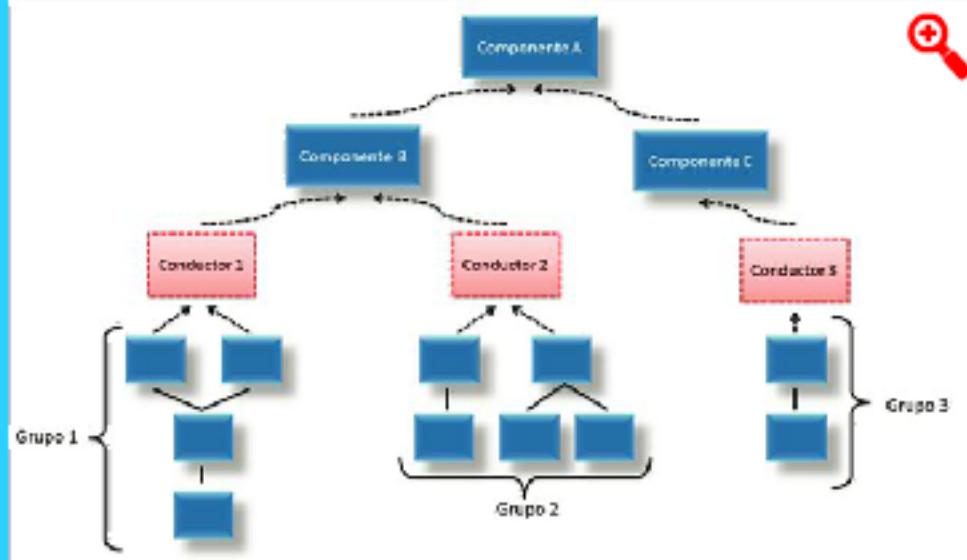
#### PRUEBAS DE INTEGRACIÓN

**05**

##### Pruebas de integración ascendente

En estas pruebas se implementan las siguientes fases:

- Combinación de los componentes del nivel más bajo en grupos.
- Construcción de un módulo conductor para coordinar la E y S de los casos de prueba.
- Probar el grupo.
- Se realizan pruebas de regresión.
- Eliminación de conductores y combinación de grupos en movimiento hacia arriba según la estructura del programa.



Integración ascendente



## PRUEBAS DE SOFTWARE



### PRUEBAS

05

#### PRUEBAS DE INTEGRACIÓN

No es suficiente con probar el funcionamiento correcto de cada módulo, se requiere además del funcionamiento en conjunto. Por tal razón, se hacen necesarias las pruebas de integración, las cuales generalmente implementan técnicas de caja negra.

Existen dos estrategias básicas para realizar la combinación de módulos en las pruebas de integración: integración ascendente e integración descendente.

**Pruebas de integración ascendente**

**Pruebas de integración descendente**

**Pruebas de integración sandwich**

**Pruebas de regresión**



## PRUEBAS DE SOFTWARE

### PRUEBAS

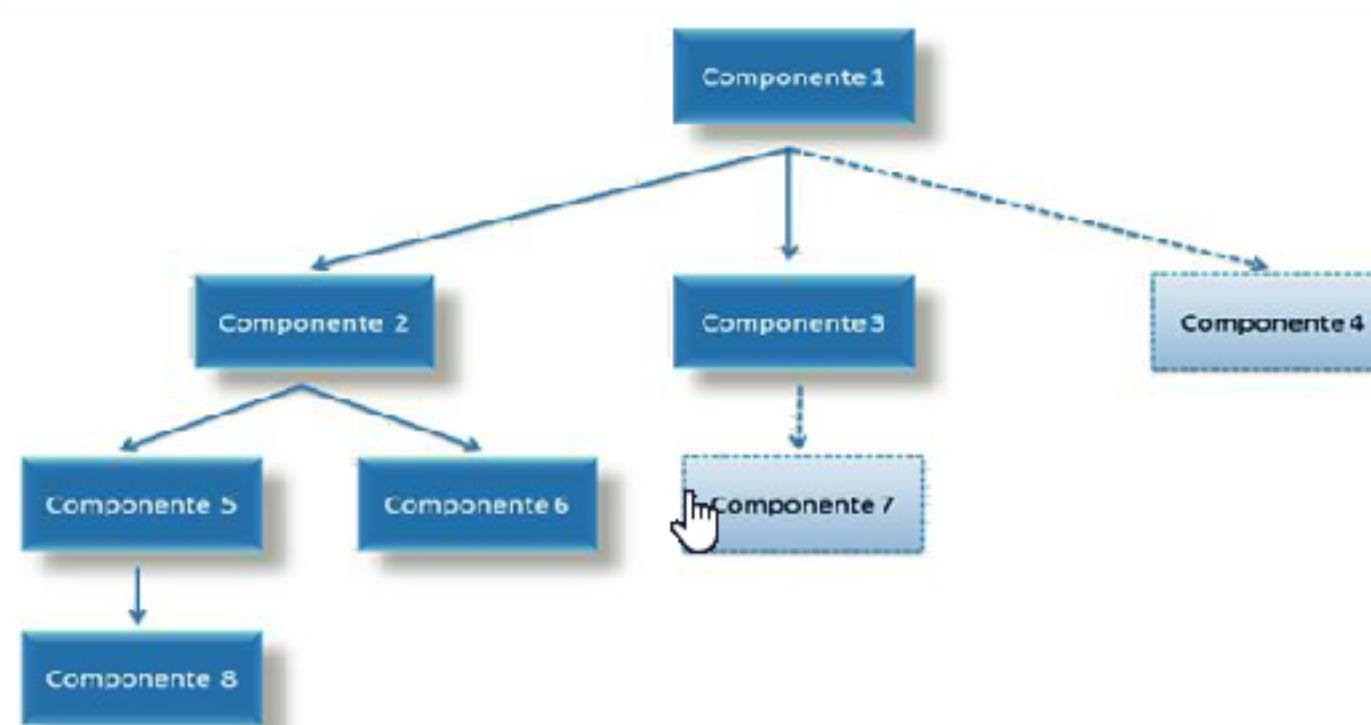
#### PRUEBAS DE INTEGRACIÓN

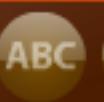
05

##### Pruebas de integración descendente

En dichas pruebas se comienza por el módulo superior y se avanza hacia los módulos inferiores, siguiendo las siguientes fases:

- El módulo de control principal actúa como conductor de pruebas y los módulos inmediatamente superiores son reemplazados por resguardos.
- Los resguardos se sustituyen por módulos reales.
- Cada vez que se integra un módulo nuevo, se realiza una prueba.
- Se realizan pruebas de regresión.





## PRUEBAS DE SOFTWARE



### PRUEBAS

05

#### PRUEBAS DE INTEGRACIÓN

No es suficiente con probar el funcionamiento correcto de cada módulo, se requiere además del funcionamiento en conjunto. Por tal razón, se hacen necesarias las pruebas de integración, las cuales generalmente implementan técnicas de caja negra.

Existen dos estrategias básicas para realizar la combinación de módulos en las pruebas de integración: integración ascendente e integración descendente.

**Pruebas de integración ascendente**

**Pruebas de integración descendente**

**Pruebas de integración sandwich**

**Pruebas de regresión**



## PRUEBAS DE SOFTWARE

### PRUEBAS

05

#### PRUEBAS DE INTEGRACIÓN

##### Pruebas de integración sandwich

En este tipo de pruebas se aplica la integración ascendente en los módulos inferiores, y de manera paralela se realiza integración descendente en los componentes superiores. La integración termina cuando ambas técnicas se encuentran en un punto intermedio de la estructura de componentes.



## PRUEBAS DE SOFTWARE



### PRUEBAS

**05**

#### PRUEBAS DE INTEGRACIÓN

No es suficiente con probar el funcionamiento correcto de cada módulo, se requiere además del funcionamiento en conjunto. Por tal razón, se hacen necesarias las pruebas de integración, las cuales generalmente implementan técnicas de caja negra.

Existen dos estrategias básicas para realizar la combinación de módulos en las pruebas de integración: integración ascendente e integración descendente.

**Pruebas de integración ascendente**

**Pruebas de integración descendente**

**Pruebas de integración sandwich**

**Pruebas de regresión**





## PRUEBAS DE SOFTWARE



### PRUEBAS

05

#### PRUEBAS DE INTEGRACIÓN

##### Pruebas de regresión

Cada vez que se agrega un nuevo componente o módulo en las pruebas de integración, el software cambia y se generan nuevas rutas en el flujo de datos. De esta manera, se requieren prueba de regresión en las cuales se ejecute algún tipo de pruebas ya realizadas, sean de ascendentes, descendentes o sandwich.





## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |           |                               |           |                               |
|-----------|-------------------------------|-----------|-------------------------------|
| <b>01</b> | <b>PRUEBAS DE CAJA BLANCA</b> | <b>05</b> | <b>PRUEBAS DE INTEGRACIÓN</b> |
| <b>02</b> | <b>PRUEBAS DE CAJA NEGRA</b>  | <b>06</b> | <b>PRUEBAS DE VALIDACIÓN</b>  |
| <b>03</b> | <b>ESTRATEGIA DE PRUEBAS</b>  | <b>07</b> | <b>PRUEBAS DEL SISTEMA</b>    |
| <b>04</b> | <b>PRUEBAS UNITARIAS</b>      | <b>08</b> | <b>PRUEBAS DE ACEPTACIÓN</b>  |



## PRUEBAS DE SOFTWARE

### PRUEBAS

06

#### PRUEBAS DE VALIDACIÓN

En la pruebas de validación se verifica el cumplimiento de los requisitos de usuario, con participación del desarrollador y el usuario. Se utiliza la técnica de caja negra, dividida en dos partes:

- Validación por parte del usuario de los resultados.
- Validación de utilidad, facilidad de uso y ergonomía de la interfaz de usuario.



En las pruebas de validación se encuentran las pruebas alfa y las pruebas beta.

- **Pruebas alfa:** son realizadas por el cliente en el lugar de desarrollo, y se prueba el sistema aunque no estén terminadas todas las funcionalidades.
- **Pruebas beta:** Son realizadas por los potenciales consumidores en su entorno, sin presencia del desarrollador.



## PRUEBAS DE SOFTWARE



### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |                                     |                                     |
|-------------------------------------|-------------------------------------|
| <b>01</b><br>PRUEBAS DE CAJA BLANCA | <b>05</b><br>PRUEBAS DE INTEGRACIÓN |
| <b>02</b><br>PRUEBAS DE CAJA NEGRA  | <b>06</b><br>PRUEBAS DE VALIDACIÓN  |
| <b>03</b><br>ESTRATEGIA DE PRUEBAS  | <b>07</b><br>PRUEBAS DEL SISTEMA    |
| <b>04</b><br>PRUEBAS UNITARIAS      | <b>08</b><br>PRUEBAS DE ACEPTACIÓN  |



## PRUEBAS DE SOFTWARE

### PRUEBAS

07

#### PRUEBAS DEL SISTEMA

En este tipo de pruebas se verifica el cumplimiento de los requisitos especificados, probando el sistema integrado en su entorno de hardware y software.





## PRUEBAS DE SOFTWARE

### PRUEBAS

Las pruebas de software, se clasifican principalmente en dos categorías: *pruebas de caja blanca* y *pruebas de caja negra*.

- 
- |    |                        |    |                        |
|----|------------------------|----|------------------------|
| 01 | PRUEBAS DE CAJA BLANCA | 05 | PRUEBAS DE INTEGRACIÓN |
| 02 | PRUEBAS DE CAJA NEGRA  | 06 | PRUEBAS DE VALIDACIÓN  |
| 03 | ESTRATEGIA DE PRUEBAS  | 07 | PRUEBAS DEL SISTEMA    |
| 04 | PRUEBAS UNITARIAS      | 08 | PRUEBAS DE ACEPTACIÓN  |



## PRUEBAS DE SOFTWARE



### PRUEBAS

08

#### PRUEBAS DE ACEPTACIÓN

Estas pruebas son realizadas en el entorno del usuario, para validar la aceptación por parte del cliente, comprobando que el sistema está listo para ser implantado. El usuario puede definir los casos de prueba.





## PRUEBAS DE SOFTWARE

De lo anterior, se puede decir que las pruebas de software son estrictamente necesarias, para determinar de manera dinámica la calidad del software; de esta manera, se garantiza que se ha construido el sistema correcto y de la forma correcta.

**ORIENTACIONES GENERALES**

**ACTIVIDADES**

**PRUEBAS**

**DISEÑO DE CASOS DE PRUEBA**



**DEPURACIÓN DE ERRORES**

**DOCUMENTACIÓN DE PRUEBAS**





## PRUEBAS DE SOFTWARE



### DISEÑO DE CASOS DE PRUEBA

Se busca seleccionar un conjunto de casos de prueba que tengan la mayor probabilidad de detectar el mayor número posible de errores y fallos, minimizando los recursos empleados para ello.

Se siguen los siguientes pasos:

- Aplicar análisis de valor límite.
- Diseñar casos con la técnica de participación de equivalencia.
- Utilizar la técnica de conjetura de error, combinado entradas y usando casos típicos de error.
- Utilizar técnicas de caja blanca

Se pueden utilizar diversos formatos para el diseño de los casos de prueba, los cuales deben incluir entre otros:

- Identificación del caso de prueba
- Caso de uso a probar
- Datos de entrada
- Salidas esperadas
- Secuencia de pasos a seguir
- Estado
- Fecha y hora de realización
- Responsable

Se proporciona como ejemplo un formato para la especificación de caso de prueba en los archivos:  
[Plantilla\\_Caso\\_de\\_prueba.xls](#) y  
[Plantilla\\_caso\\_prueba.doc](#)



## PRUEBAS DE SOFTWARE

De lo anterior, se puede decir que las pruebas de software son estrictamente necesarias, para determinar de manera dinámica la calidad del software; de esta manera, se garantiza que se ha construido el sistema correcto y de la forma correcta.

**ORIENTACIONES GENERALES**

**ACTIVIDADES**

**PRUEBAS**

**DISEÑO DE CASOS DE PRUEBA**

**DEPURACIÓN DE ERRORES**

**DOCUMENTACIÓN DE PRUEBAS**



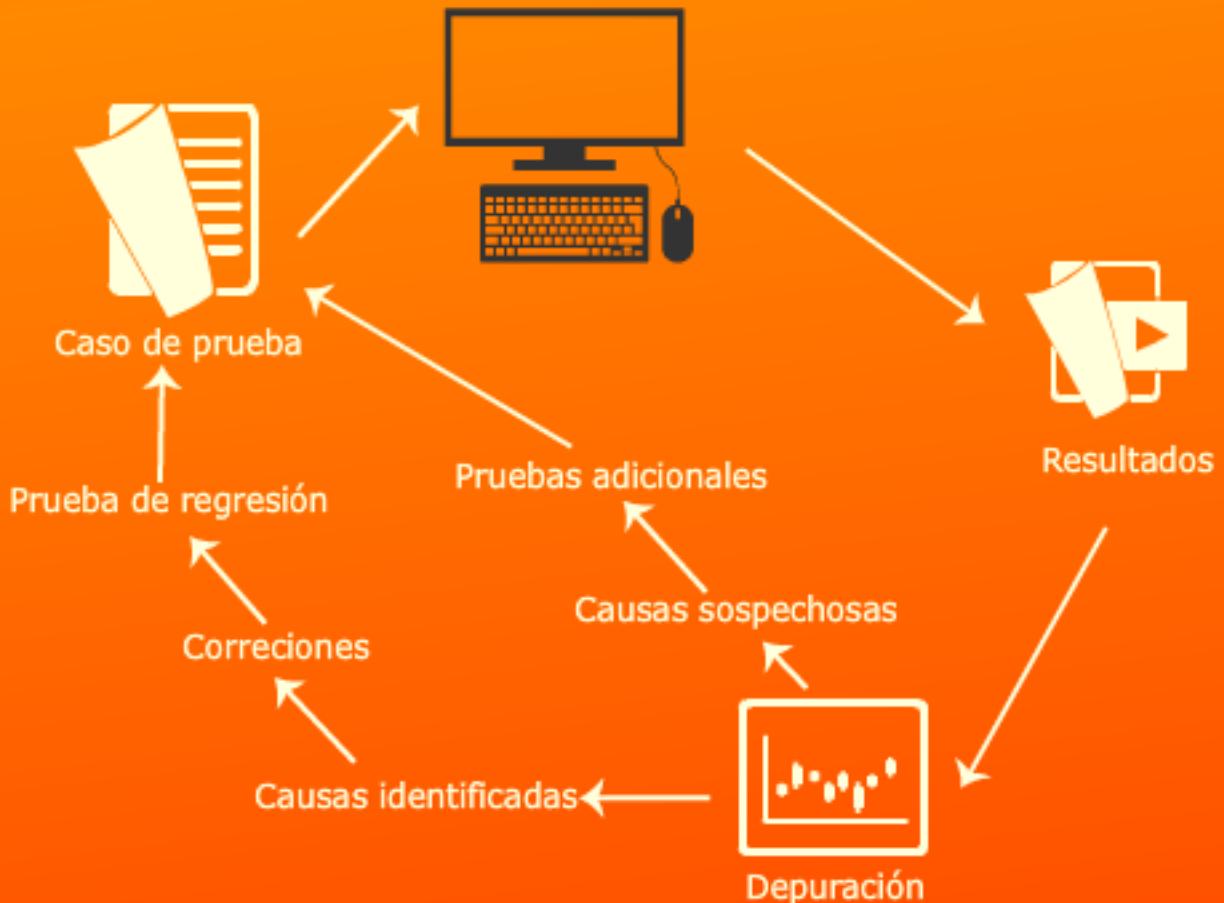


## PRUEBAS DE SOFTWARE

### DEPURACIÓN DE ERRORES

La depuración de errores se realiza en una dinámica cíclica como lo muestra la siguiente figura

El proceso de depuración inicia con la ejecución de un caso de prueba, valorando los resultados para encontrar la falta de correspondencia entre el valor real y el esperado. Se busca encontrar la causa de la falta y así corregir el error, pero si no se encuentra, se puede tener una causa sospechosa, caso en el cual se realizan pruebas auxiliares para la corrección del error en forma iterativa.





## PRUEBAS DE SOFTWARE

De lo anterior, se puede decir que las pruebas de software son estrictamente necesarias, para determinar de manera dinámica la calidad del software; de esta manera, se garantiza que se ha construido el sistema correcto y de la forma correcta.

**ORIENTACIONES GENERALES**

**ACTIVIDADES**

**PRUEBAS**

**DISEÑO DE CASOS DE PRUEBA**

**DEPURACIÓN DE ERRORES**

**DOCUMENTACIÓN DE PRUEBAS**





## PRUEBAS DE SOFTWARE



### DOCUMENTACIÓN DE PRUEBAS

Según el estándar IEEE 829-1983, las pruebas deben generar los siguientes documentos mínimos:

- Plan de pruebas.
- Especificación de requerimientos para el diseño de casos de prueba.
- Caso de prueba con descripción del procedimiento y del ítem a probar.
- Reporte de incidentes de prueba.
- Resumen de pruebas.

### DISEÑO DEL PLAN DE PRUEBAS





## PRUEBAS DE SOFTWARE

### DOCUMENTACIÓN DE PRUEBAS

#### DISEÑO DEL PLAN DE PRUEBAS



Aunque el plan de pruebas se utiliza en la fase de pruebas, se debe definir en la etapa de diseño, que es cuando se especifica la arquitectura del sistema.

Los principales elementos que se deben especificar en un plan de pruebas de acuerdo con el estándar IEEE 829-2008 son:

- El alcance y los riegos asociados
- Los objetivos y el criterio de finalización de las pruebas (condiciones de suspensión y reanudación)
- El tipo de técnicas a aplicar.
- Método o estrategia de pruebas y tiempo disponible
- Los recursos requeridos para realizar las pruebas: personal, entorno de pruebas, presupuesto
- Priorización de los casos de prueba
- Responsabilidades
- Programación de los casos de prueba

Se proporciona como ejemplo el archivo [Plantilla\\_Plan\\_de\\_pruebas.doc](#), basado en este estándar.