



UNIVERSIDAD
Popular del cesar

Ingeniería de Sistemas



Ingeniería de Software

II

Unidad No. 1
Pruebas de Software

Objetivos de aprendizaje

- Conocer los principios básicos en los que se fundamentan las pruebas del software.
- Comprender la necesidad de las pruebas como parte esencial del desarrollo de un sistema de software.
- Conocer las diferentes técnicas de pruebas de software.
- Saber diferenciar los diferentes niveles de pruebas de software
- Realizar pruebas a un producto de software empleando herramientas y técnicas para detectar errores.

Introducción

La importancia de las pruebas en un proceso de desarrollo de software, ha sido valorada por los equipos de desarrollo, como parte integral del proceso de producción para alcanzar un **producto con calidad**.

Al construir software habitualmente se comenten errores. En la industria, la técnica para solucionar los problemas derivados de dichos errores, serán las **pruebas de software** ('testing'), que consistirán en una serie de pasos realizados **antes y después** de la construcción de este software.

Las **pruebas de software** se implementan para comprobar que el sistema cumple con los requerimientos del usuario y que su funcionamiento es correcto, es decir sin errores o defectos. Por esta razón, las pruebas son un elemento diferente dentro del proceso de desarrollo. Al contrario del resto de las actividades, **su éxito radica en la detección de errores**, tanto en el propio proceso, como en el software obtenido.

Introducción

- ¿Es posible construir software que **no falle**?
- Hoy en día estamos acostumbrados a que el software **falle**.
- Al construir software habitualmente se comenten **errores**.



1.1. Conceptos fundamentales

¿Qué son las pruebas de software?

Es todo proceso orientado a comprobar la calidad del software mediante la identificación de fallos en el mismo.

¿Qué es probar software?

Probar es el proceso ejecución de un programa con el fin de encontrar errores.

¿Por qué Probar Software?

Para demostrar que un programa tiene fallos, por tanto, hay la probabilidad de encontrar errores.

1.1. Conceptos fundamentales

• Verificación y Validación (V&V)

En el control de calidad de software se distinguen dos procesos de evaluación propios del proceso de desarrollo de software: la verificación y la validación.

V&V es un conjunto de procedimientos, actividades, técnicas y herramientas que se utilizan, paralelamente al desarrollo de software, para asegurar que un producto software resuelve el problema inicialmente planteado

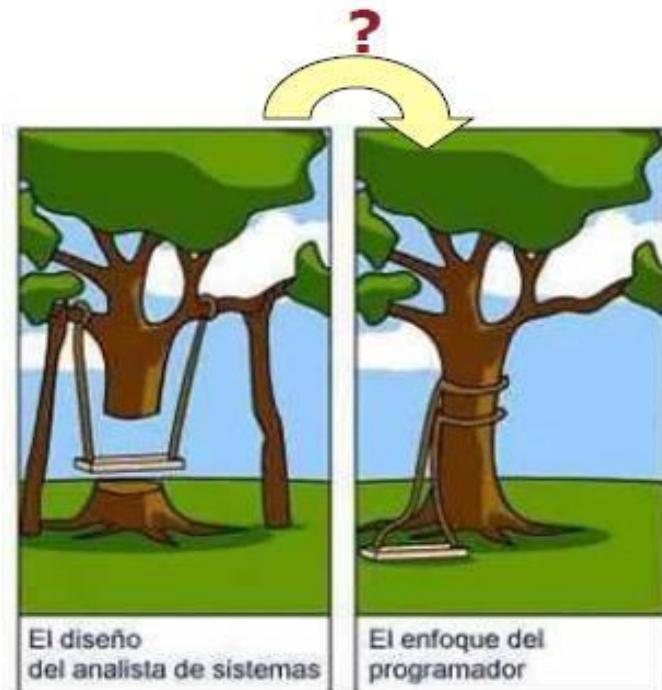


1.1. Conceptos fundamentales

• Verificación

Es el proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase.

- ¿Estamos construyendo el producto correctamente?
[Siguiendo las especificaciones]

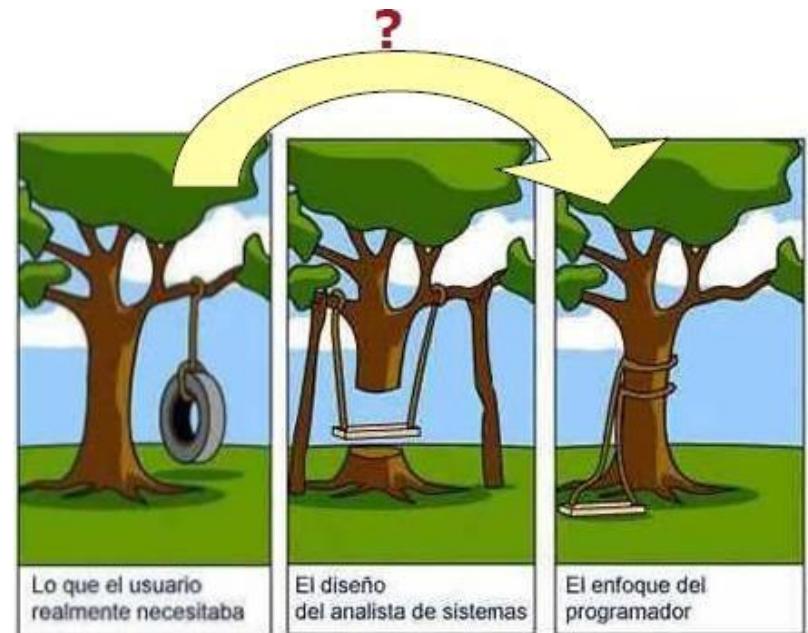


1.1. Conceptos fundamentales

• Validación

Es el proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario.

- ¿Estamos construyendo el producto correcto? [Siguiendo lo que el cliente quiere]



1.1. Conceptos fundamentales

• Actividades de VV

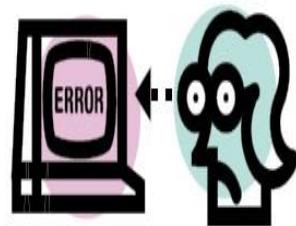
	VERIFICACIÓN	VALIDACIÓN
	¿Estamos construyendo correctamente el producto?	¿Estamos construyendo el producto correcto ?
El software debe	Estar conforme a su especificación	Hacer lo que el usuario realmente quiere
Objetivo	Demostrar la consistencia, compleción y corrección de los artefactos de las distintas fases (productos intermedios)	Determinar la corrección del producto final respecto a las necesidades del usuario
Técnica más utilizada	Revisiones	Pruebas

1.1. Conceptos fundamentales

TECNICAS DE V&V



♦ Inspecciones del software (Estático)



♦ Pruebas del software (Dinámico)

ESTÁTICAS

Buscan fallas en el sistema en reposo, analizando los distintos modelos que lo componen. Se aplican a requisitos, modelos de análisis y diseño, y al código

DINÁMICAS

Buscan fallas mediante entradas al sistema en funcionamiento. Se denominan **pruebas de software** o *testing*, y se aplican al código.

1.1. Conceptos fundamentales

Para terminar de entender las pruebas se deben diferenciar los términos **error**, **fallo** y **defecto**. Estos conceptos están relacionados entre si, pero tienen significados diferentes.

- **Error:** Una equivocación humana.
- **Defecto:** Se produce cuando una persona comete un error.
- **Fallo:** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificado.



1.1. Conceptos fundamentales

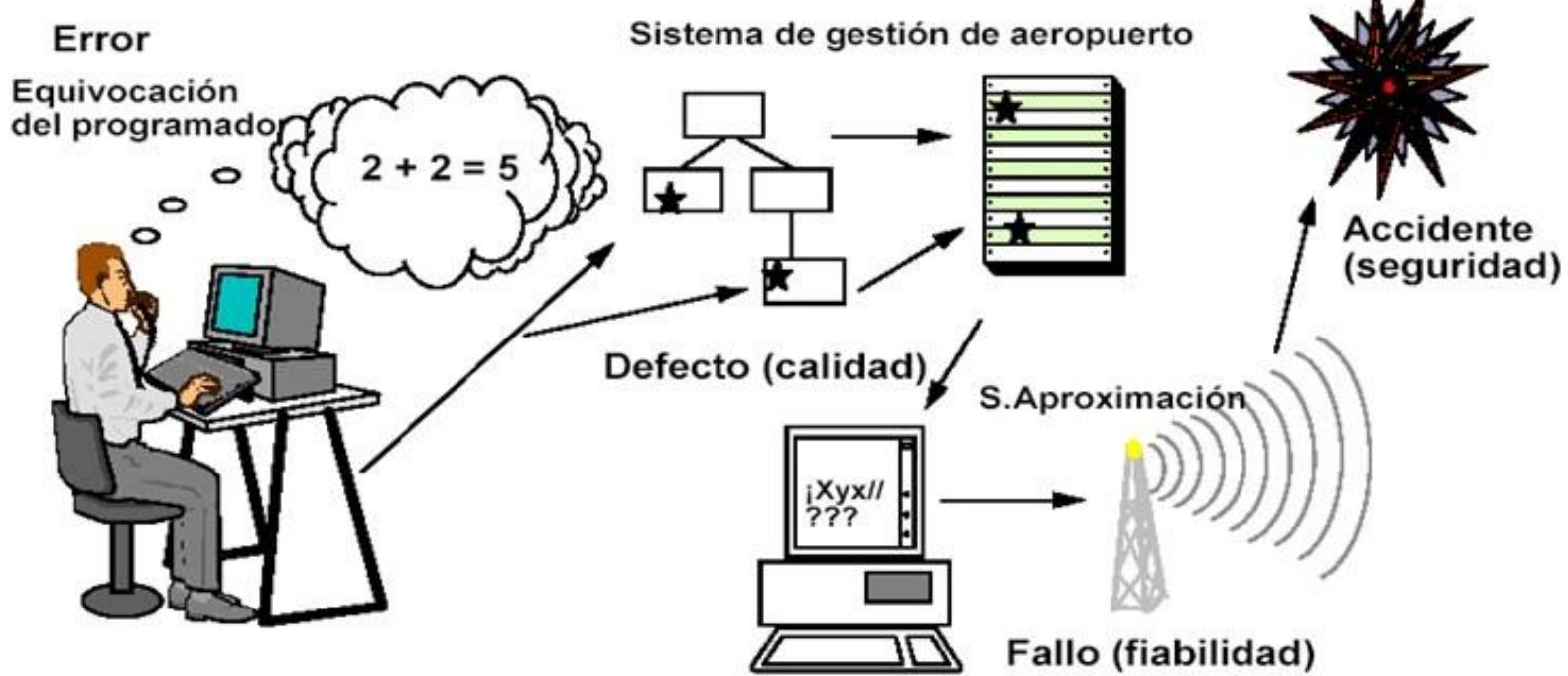
Relación entre error, defecto y fallo

- Una acción humana que conduce a un resultado incorrecto → **Error**
- Se comete un error → **Defecto**
- Un resultado incorrecto → **Fallo**



1.1. Conceptos fundamentales

Relación entre error, defecto y fallo



Pruebas de Software

1.1. Conceptos fundamentales

Objetivos de las pruebas de software

- El objetivo principal de las pruebas es aportar **calidad** al producto que se está desarrollando.
- **Encontrar el mayor número de defectos** en el código para que se resuelvan y eliminarlos.
- Asegurar que el producto **funciona tal y cómo se ha definido en los requisitos**
- Dar información al cliente de los **defectos que no se han podido eliminar** del producto final.
- Proporcionar al **producto final un grado mayor de calidad**.

Pruebas de Software

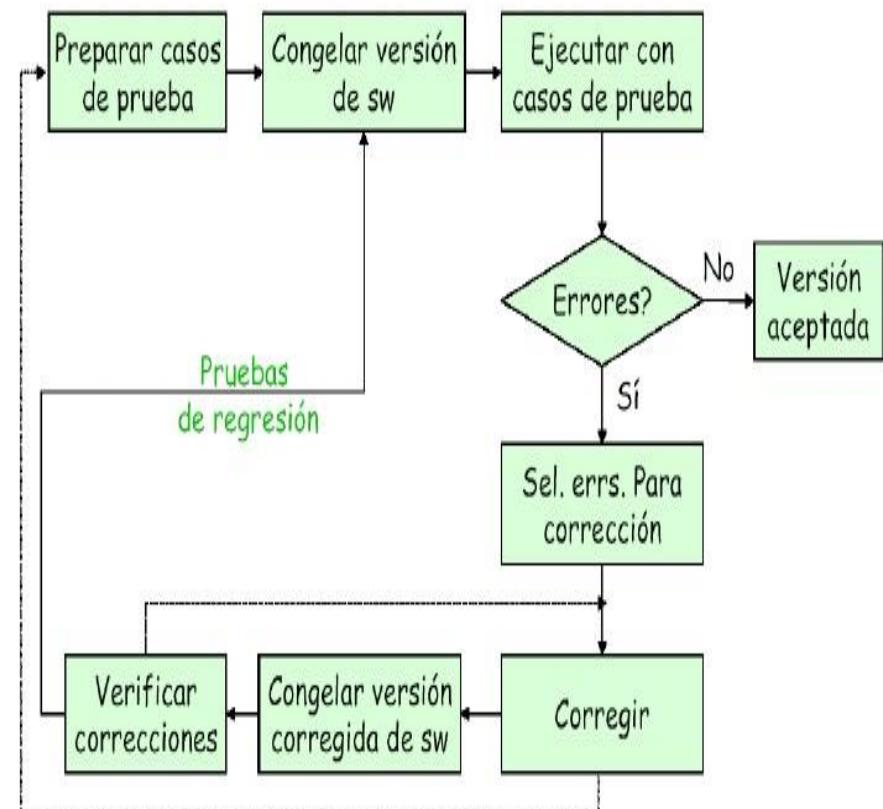
1.1. Conceptos fundamentales

Casos de prueba: Es un conjunto de entradas, condiciones de ejecución y resultados esperados, que son desarrollados para cumplir con un objetivo en particular.

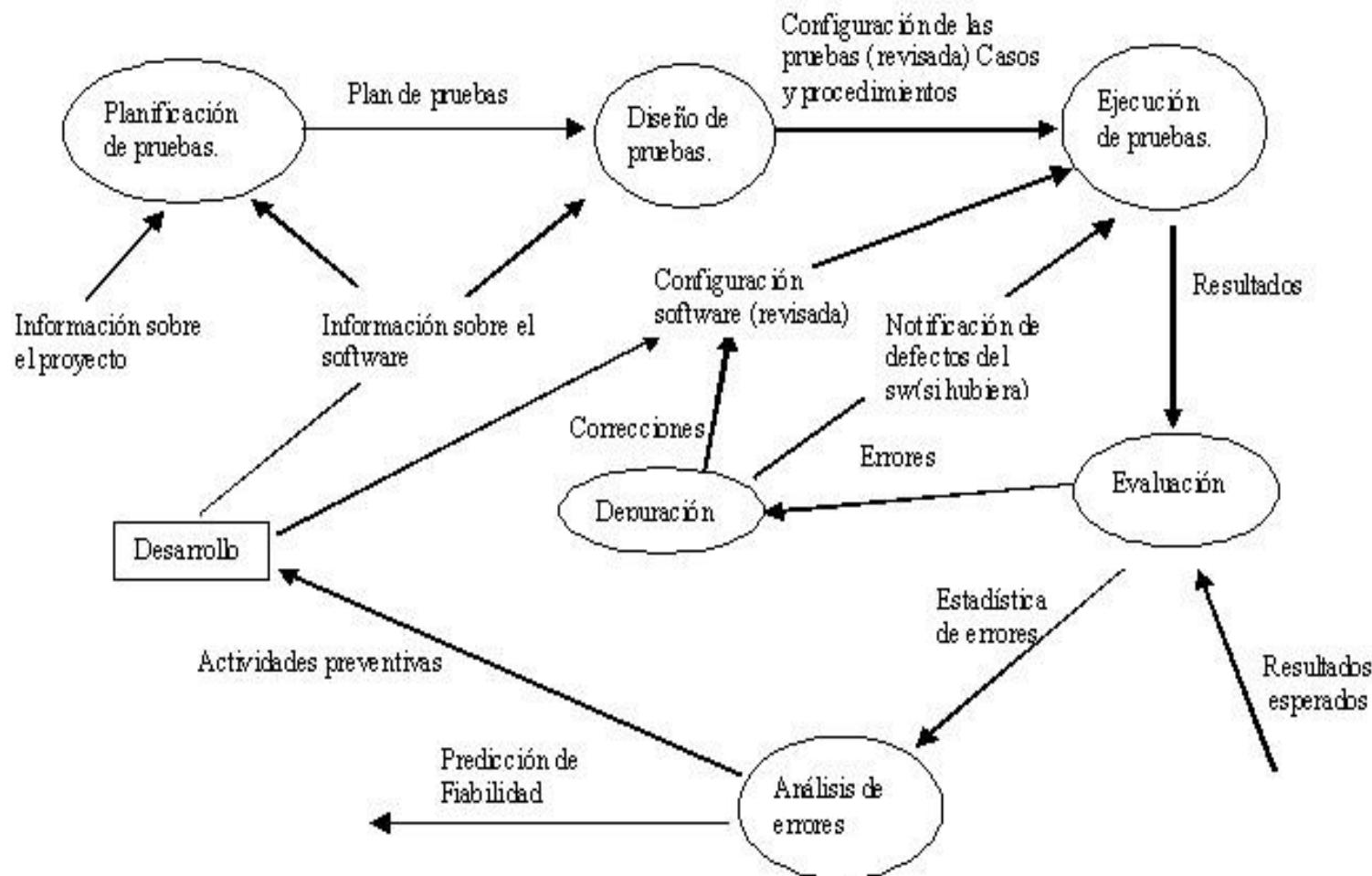
1.2. El proceso de las pruebas

Cómo se llevan a cabo las pruebas?

- Verificando el comportamiento del programa sobre un conjunto de **casos de prueba**.
- Los casos de prueba se generarán mediante **técnicas y estrategias** específicas de pruebas. Estas técnicas ayudan a conseguir la búsqueda de los errores de un programa.



1.2. El proceso de las pruebas



1.3. Técnicas de prueba

El objetivo de las pruebas es descubrir el máximo numero de fallos en el Software, para aportar **calidad** al producto que se está desarrollando.

Las **técnicas de prueba** son el medio que ayuda a conseguir los objetivos de las pruebas a través de su sistematización.

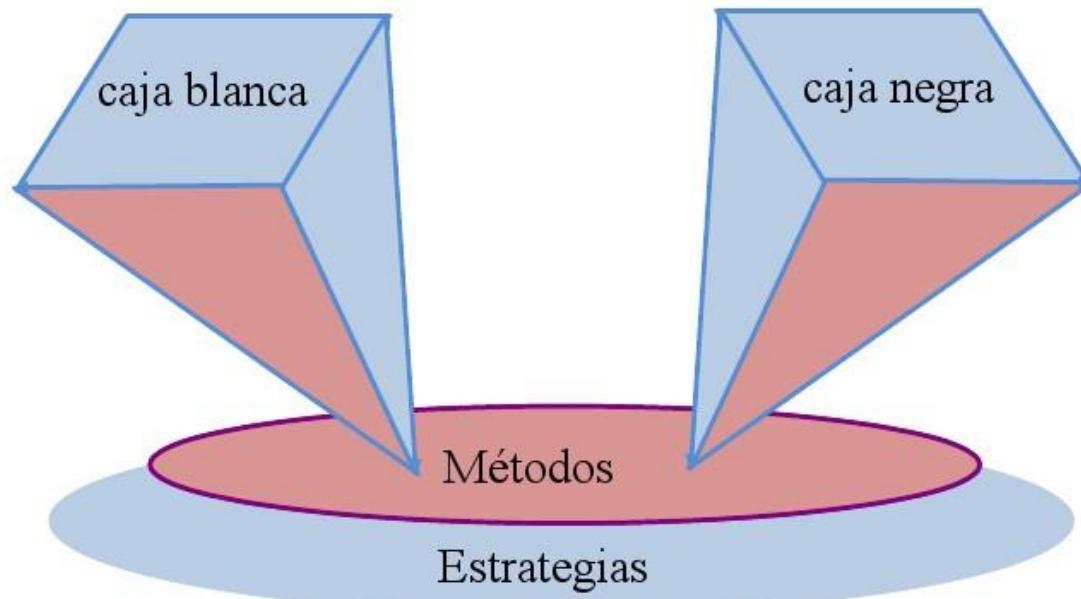
El objeto primordial de las técnicas a analizar es sistematizar el proceso de las pruebas.

Las técnicas de prueba se han clasificado en dos categorías: Las técnicas de **caja blanca** y las de **caja negra**.

1.3. Técnicas de prueba

Existen dos técnicas básicas para diseñar casos de prueba

- De **caja blanca** (o estructurales)
- De **caja negra** (o funcionales)



1.3. Técnicas de prueba

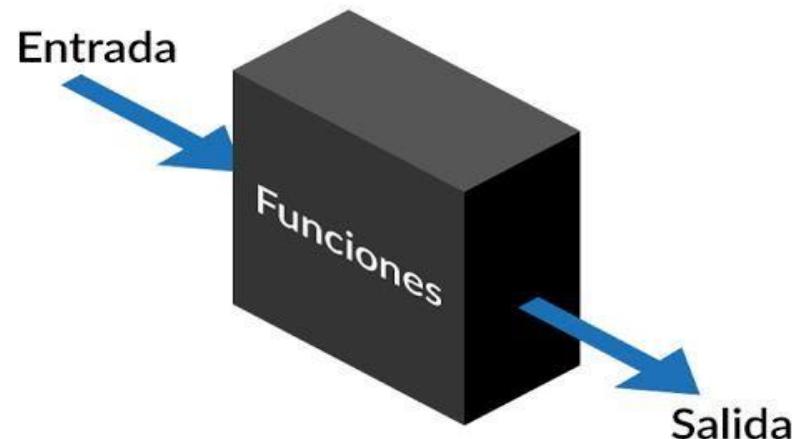
- Enfoque **estructural** o de **caja blanca**: Basadas en información sobre cómo el software ha sido diseñado o codificado.



- Enfoque **funcional** o de **caja negra**: Basada sólo en el comportamiento de entrada/salida.



1.3.1. Técnicas de pruebas de caja negra



Las técnicas de **caja negra o funcionales**, son las que utilizan el análisis de la especificación, tanto funcional como no funcional, sin tener en cuenta la estructura interna del programa para diseñar los casos de prueba.

1.3.1. Técnicas de pruebas de caja negra

Las pruebas de **Caja Negra** intentan encontrar errores de las siguientes categorías:

- Funciones Incorrectas o Ausentes.
- Errores de Interfaz.
- Errores en estructuras de datos o acceso a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.



1.3.1. Técnicas de pruebas de caja negra

Hay varios métodos que se pueden aplicar a la hora de escoger la técnica de caja negra como modelo para las pruebas. Las dos técnicas más comunes son:

- **Partición de equivalencia.**
- **Análisis de valores límite.**

1.3.1. Técnicas de pruebas de caja negra

❖Partición de Equivalencia

- Se basa en la **división del campo de entrada** en un conjunto de clases de datos denominadas **clases de equivalencia**.
- **Clase de equivalencia:** Son un conjunto de datos de entrada que definen estados válidos y no válidos del sistema.
 - **Clase válida:** Genera un valor esperado
 - **Clase no válida:** Genera un valor inesperado
- Se obtienen a partir de las **condiciones de entrada** descritas en las especificaciones.

1.3.1. Técnicas de pruebas de caja negra

❖Partición de Equivalencia

Pasos para identificar clases de equivalencia.

1. Identificación de las condiciones de entrada del programa.
2. Identificar las clases de equivalencia:
 - Datos válidos.
 - Datos no válidos.

Condiciones de Entrada	Clases de equivalencia válida	Clases de equivalencia no válida

1.3.1. Técnicas de pruebas de caja negra

Criterios de identificación de las clases de equivalencias

Condiciones de Entrada	Número de clases de equivalencia válida	Número de clases de equivalencia no válida
1. Rango de valores	1 clase que contemple los valores del rango	2 clases fuera del rango, una por encima y otra por debajo de éste
2. Valor específico	1 clase que contemple dicho valor	2 clases que representen un valor por encima y otro por debajo
3. Elementos de un conjunto tratados de forma diferente por el programa	1 clase de equivalencia por cada elemento	1 clase que represente un elemento fuera del conjunto
4. Condición lógica	1 clase que cumpla la condición	1 clase que no cumpla la condición

1.3.1. Técnicas de pruebas de caja negra

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

Condición de Entrada	Clases Válidas	Clases Inválidas
Código de Área # de 3 dígitos que no empieza con 0 ni 1:	1) $200 \leq \text{código} \leq 999$	2) Código < 200. 3) Código > 999. 4) No es número.
Nombre Para identificar la operación	5) Seis caracteres.	6) Menos de 6 caracteres. 7) Más de 6 caracteres.
Orden Una de las Siguientes	8) "Cheque" 9) "Depósito" 10) "Pago factura" 11) "Retiro de fondos"	12) Ninguna orden válida

1.3.1. Técnicas de pruebas de caja negra

❖Partición de Equivalencia

Pasos para identificar casos de prueba:

1. Asignar un número único para cada clase de equivalencia.
2. Escribir casos de pruebas para todas las clases válidas.
3. Escribir casos de pruebas para todas las clases no válidas.

Dato entrada	Valor	Escenario	Resultado esperado
		Correcto / incorrecto	

1.3.1. Técnicas de pruebas de caja negra

❖ Análisis de valores límite

- Un análisis de las condiciones límites de las clases de equivalencia aumenta la eficiencia de la prueba.
- **Condiciones límites:** valores justo por encima y por debajo de los márgenes de la clase de equivalencia.
- Se basa en la evidencia experimental de que **los errores** suelen aparecer con mayor probabilidad en los **extremos de los campos de entrada**.

1.3.1. Técnicas de pruebas de caja negra

❖Análisis de valores límite

Las reglas para identificar las clases son:

1. Si una condición de entrada especifica un rango que deben generar casos para los extremos.
2. Si la condición de entrada especifica un número finito y consecutivo de valores, escribir casos para los números máximo, mínimo, uno más del máximo y uno menos del mínimo de valores

1.3.1. Técnicas de pruebas de caja negra

❖ Análisis de valores límite

Las reglas para identificar las clases son:

Condiciones de la especificación	Obtención de los casos de prueba
1. Rango de valores como condición de entrada	1 caso que ejercite el valor máximo del rango
	1 caso que ejercite el valor mínimo del rango
	1 caso que ejercite el valor justo por encima del máximo del rango
	1 caso que ejercite el valor justo por debajo del mínimo del rango
2. Valor numérico específico como condición de entrada	1 caso que ejercite el valor numérico específico
	1 caso que ejercite el valor justo por encima del valor numérico específico
	1 caso que ejercite el valor justo por debajo de valor numérico específico

1.3.1. Técnicas de pruebas de caja negra

❖ Análisis de valores límite

Las reglas para identificar las clases son:

Condición	Descripción de los casos de prueba
Entre 5 y 15 caracteres	1 caso con n° de caracteres identificados=15
	1 caso con n° de caracteres identificados=5
	1 caso con n° de caracteres identificados=16
	1 caso con n° de caracteres identificados=4

1.3.2. Técnicas de pruebas de caja blanca

Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento.

Estos casos deben garantizar:

- Que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
- Que se ejercent todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Que se ejecuten todos los bucles en sus límites operacionales.
- Que se ejercent las estructuras internas de datos para asegurar su validez.

1.3.2. Técnicas de pruebas de caja blanca

Para validar el software en su estructura interna se han definido distintos criterios de cobertura lógica, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba. Estos criterios son:

- **Cobertura de Sentencias:** Que cada sentencia se ejecute al menos una vez
- **Cobertura de decisiones:** Que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso.
- **Cobertura de Condiciones:** Que cada condición de cada decisión adopte el valor verdadero al menos una vez y el falso al menos una vez.
- **Criterios de decisión/Condición:** Que se cumplan a la vez el criterio de condiciones y el de decisiones.
- **Criterio de Condición Múltiple:** La evaluación de las condiciones de cada decisión no se realiza de forma simultánea.

- **Criterio de Cobertura de Caminos:** Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa.

1.3.2. Técnicas de pruebas de caja blanca

Una de las técnicas empleadas para aplicar las pruebas de Caja Blanca es la **Prueba del Camino Básico**.

Esta técnica se basa en obtener una medida de la complejidad del diseño procedural de un programa o de la lógica del programa.

Esta técnica fue propuesta por Tom McCabe y consiste en definir un conjunto básico de caminos usando la medida de complejidad llamada complejidad ciclomática.

Esta medida es la complejidad ciclomática de McCabe, y representa un límite superior para el número de casos de prueba que se deben realizar para asegurar que se ejecuta cada camino del programa.



1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

Los pasos en las pruebas de caminos básicos son:

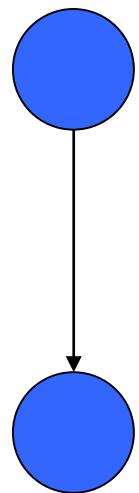
1. Dibujar grafo de flujo.
2. Determinar la complejidad ciclomática del grafo.
3. Determinar los caminos linealmente independientes.
4. Diseñas los casos de prueba.



1.3.2. Técnicas de pruebas de caja blanca

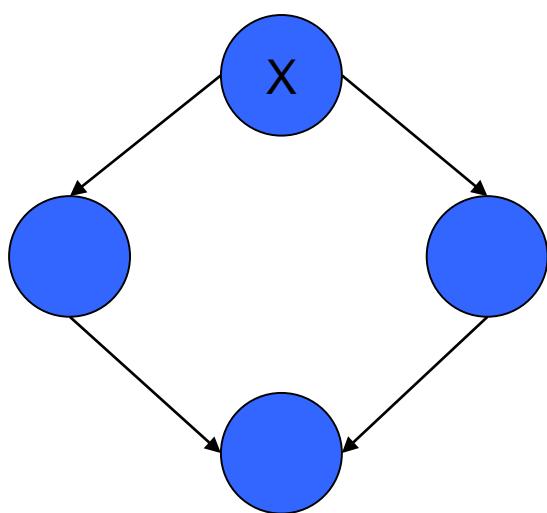
1. Dibujar Grafo de Flujo de las Estructuras Básicas de programa

La notación que se utiliza para construir el grafo es la siguiente:

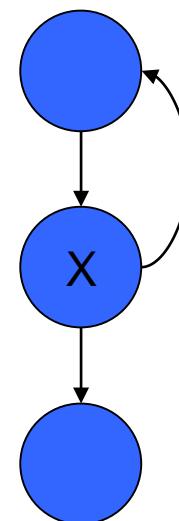


Secuencia

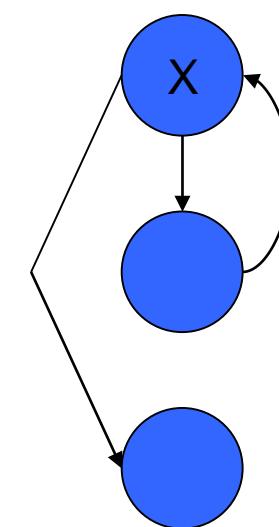
Si x Entonces ...
(If x Then ... Else...)



Hacer ... hasta x
(Do ... Until x)
Repetir



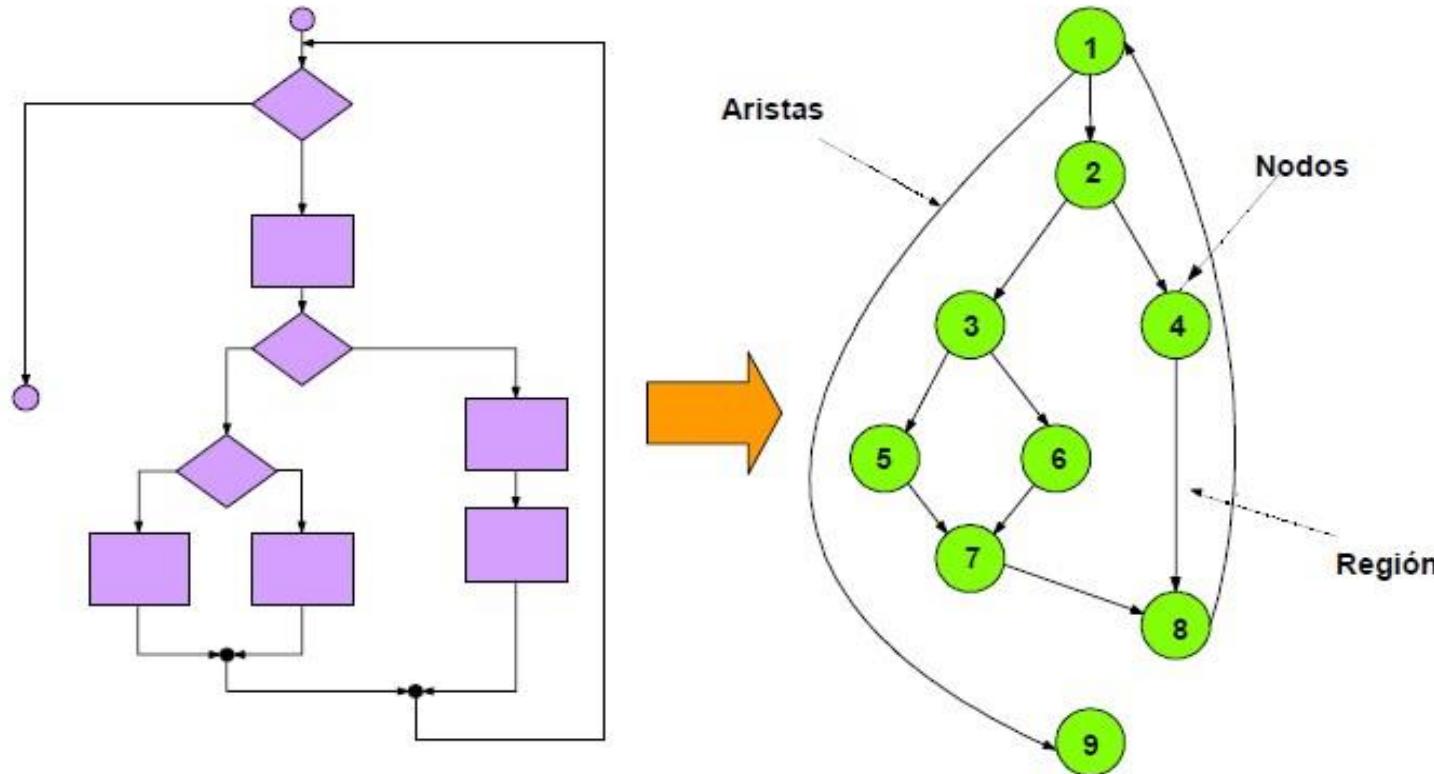
Mientras x Hacer ...
(While x Do ...)



1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

Ejemplo de un flujo de programa con la notación de grafo de flujo.



1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

2. Determinar la complejidad ciclomática del grafo.

La complejidad ciclomática determina el número de caminos a probar, mediante la siguiente fórmula:

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

$$V(G) = \# \text{ de regiones cerradas del grafo.}$$

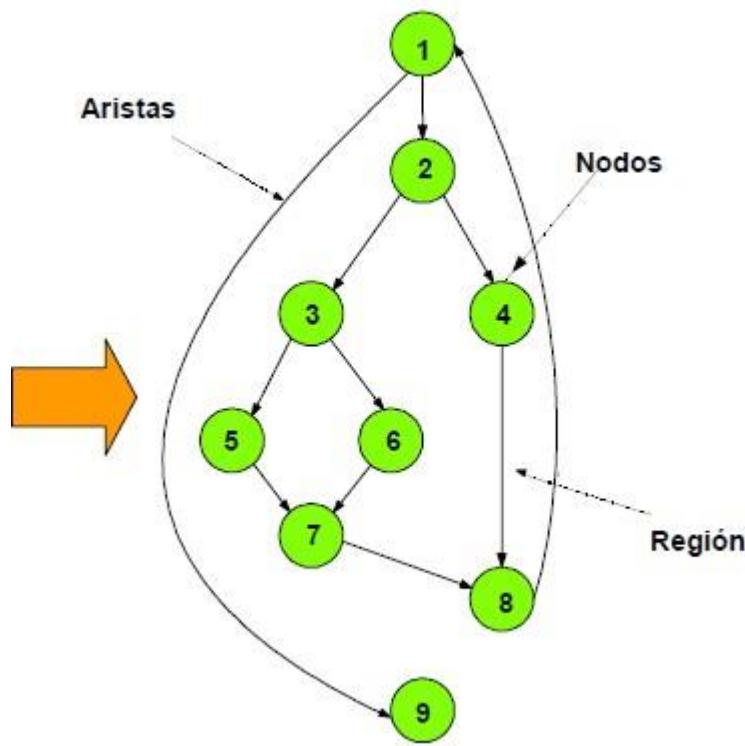
$$V(G) = \# \text{ de nodos de condición} + 1$$



1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

3. Determinar los caminos linealmente independientes.



Así por ejemplo para la el grafo, se determina la complejidad ciclomática, que determinara los caminos a probar

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

$$V(G) = 11 - 9 + 2$$

$$V(G) = 4$$

Los cuatro posibles **caminos independientes** generados serían:

Camino 1: 1-9

Camino 2: 1-2-4-8-1-9

Camino 3: 1-2-3-5-7-8-1-9

Camino 4: 1-2-3-6-7-8-1-9

1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

4. Diseñar los casos de pruebas que sigan cada camino

El último paso es construir los casos de prueba que fuerzan la ejecución de cada camino. Una forma de representar el conjunto de casos de prueba es como se muestra en la Tabla siguiente:

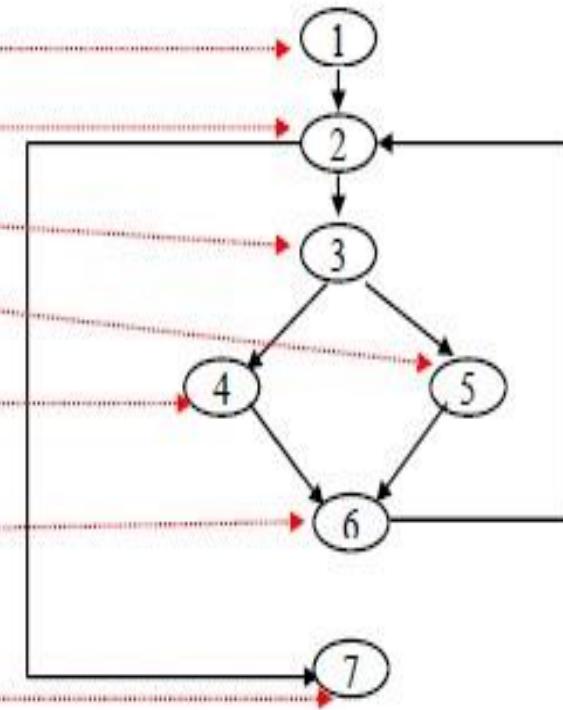
Numero del camino	Caso de prueba	Resultado esperado

1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

Grafo de Flujo de un Programa (Pseudocódigo)

```
Abrir archivo;  
WHILE (haya registros) DO  
    IF (cliente) THEN  
        Mostrar cliente;  
    ELSE  
        Mostrar registro;  
    ENDIF;  
    Incrementar registro;  
ENDWHILE;  
Cerrar archivos.
```

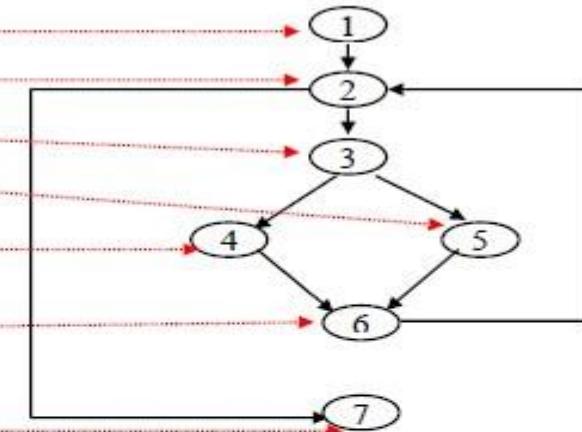


1.3.2. Técnicas de pruebas de caja blanca

Prueba del Camino Básico

Grafo de Flujo de un Programa (Pseudocódigo)

```
Abrir archivo;  
WHILE (haya registros) DO  
    IF (cliente) THEN  
        Mostrar cliente;  
    ELSE  
        Mostrar registro;  
    ENDIF;  
    Incrementar registro;  
ENDWHILE;  
Cerrar archivos.
```



Complejidad ciclomática:

- Indicador del número de caminos independientes de un grafo
- Límite mínimo de número de casos de prueba para un programa

Cálculo de complejidad ciclomática

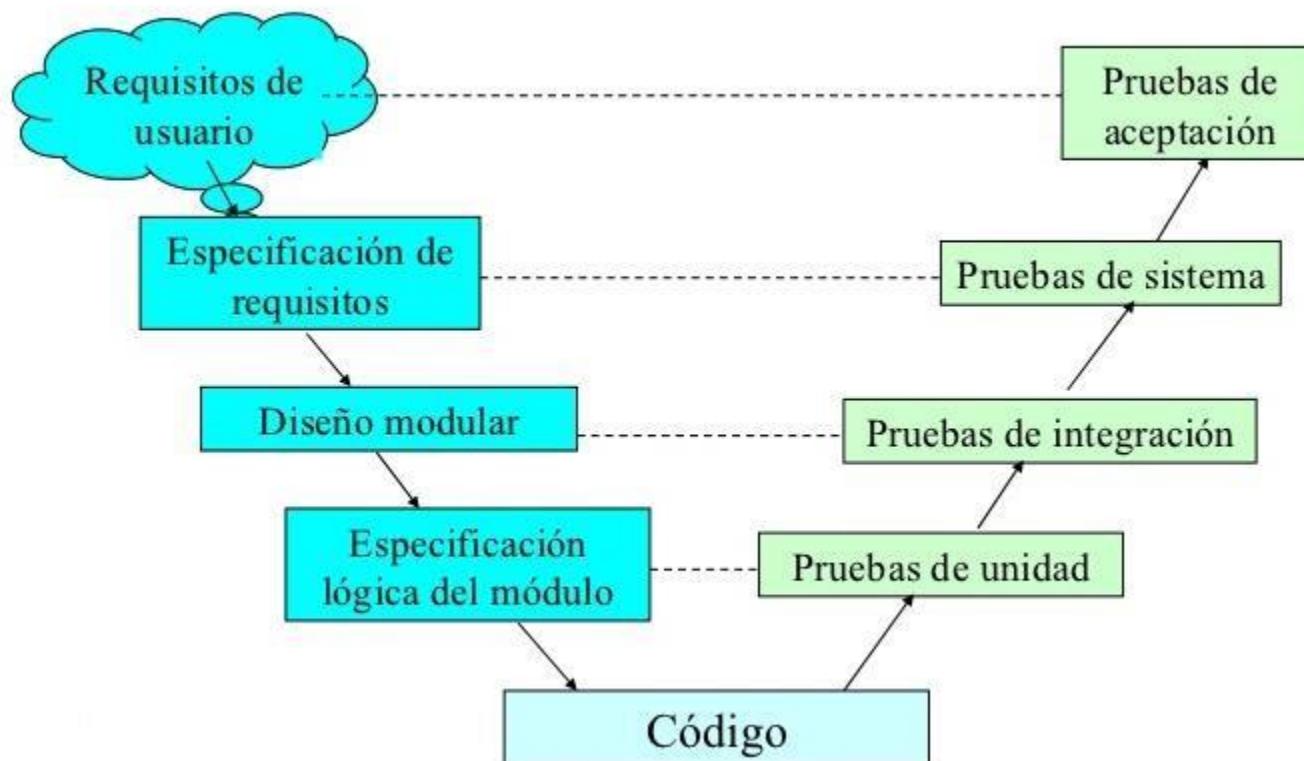
- $V(G) = \text{arcos} - \text{nodos} + 2$
- $V(G) = \text{número de regiones cerradas}$
- $V(G) = \text{número de nodos condición} + 1$

Possible conjunto de caminos

- 1 - 2 - 7
- 1 - 2 - 3 - 4 - 6
- 1 - 2 - 3 - 5 - 6

1.4. Niveles de pruebas

Para cada nivel de desarrollo se define un nivel de prueba. Dentro de cada nivel de prueba el probador debe asegurarse de que los resultados cumplen con la verificación y la validación del software.



1.4.1. Pruebas unitarias

El primer nivel de las pruebas es el de la prueba unitaria o de componente, que consiste en probar el correcto funcionamiento de una unidad de código.

En este nivel se comprueba la **lógica**, la funcionalidad y especificación de cada unidad (componente, clase u objeto) del sistema de manera **aislada** con respecto al resto de unidades. Se hace 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 y caja negra. Tanto las pruebas de caja blanca como las de caja negra se han de aplicar para probar de la manera más completa posible un módulo.



1.4.1. Pruebas unitarias

El propósito de las pruebas unitarias es encontrar errores en la lógica, datos o algoritmos en componentes o subsistemas individuales. Los casos de prueba que se generan deben:

- Tratar de detectar errores en los algoritmos y la lógica
- Tratar de detectar errores en la manipulación de las estructuras de datos
- Tratar de detectar errores en el llamado a otros módulos
- Identificar todos los caminos posibles del módulo y tratar de hacer casos de prueba que los cubran
- Tratar de detectar errores usando datos límites

1.4.2. Pruebas de integración

Aún cuando los módulos de un programa funcionen bien por separado es necesario probarlos conjuntamente. Un módulo o método puede tener un efecto adverso o inadvertido sobre otro módulo. Por lo tanto, es necesario probar el software ensamblando todos los módulos probados previamente. Realizado por los desarrolladores de los módulos que serán integrados.

En este nivel se utilizan técnicas que verifican el correcto manejo de las entradas y salidas del software (pruebas funcionales). Generalmente implementan técnicas de Caja negra, basadas en las especificaciones de las interfaces.

1.4.2. Pruebas de integración

El propósito de las pruebas de integración es encontrar errores en las interfaces entre los módulos. Los casos de prueba que se generan deben:

- Tratar de detectar errores en los formatos de intercambio de datos
- Tratar de detectar errores en el orden en que interactúan los módulos, la sincronización y los tiempos de respuesta

1.4.3. Pruebas de sistema

Una vez que se han probado los componentes y la integración de los mismos, entramos en el tercer nivel de prueba, donde se va a comprobar si el producto cumple con los requisitos especificados.

Las pruebas de sistema deben de verificar los requisitos funcionales y no funcionales del sistema y las características de calidad. Para ello se aplican técnicas de prueba de caja negra.

Su propósito es encontrar errores en el comportamiento del sistema de acuerdo con la especificación de requerimientos. Realizadas por un grupo diferente al de desarrollo.



1.4.3. Pruebas de sistema

Este tipo de pruebas tiene como propósito revisar el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema entre el hardware y el software, además, que realizan las funciones adecuadas. Concretamente se debe comprobar que:

- Se cumplen los requisitos funcionales establecidos.
- El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- La adecuación de la documentación de usuario.
- Rendimiento y respuesta en condiciones límite y de sobrecarga.

1.4.4. 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. Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.
- Están enfocadas a probar los requisitos de usuario, a demostrar que se cumplen los requisitos, los criterios de aceptación o el contrato establecido.
- Está considerada como la fase final del proceso para crear una confianza en que el producto es el apropiado para su uso en explotación.

Referencias

- Roger, P. (2010). Ingeniería del Software: Un Enfoque Práctico. 7^a. Edicion. McGraw Hill.
- Ian S. (2011). Ingeniería del Software. 9^a. Edición. Pearson Educación, México.
- Salvador, S. Miguel, S. y Daniel, Rodríguez. (2012). Ingeniería del software. Un enfoque desde la guía SWEBOK. Alfaomega.
- José, S. (2015). Pruebas de Software. Fundamentos y Técnicas. Universidad Politécnica de Madrid, España.
- Beatriz F. (2013). Técnicas de Pruebas de Software. Universidad del Valle, Colombia.



UNIVERSIDAD
Popular del cesar