**Pruebas de Aplicaciones Convencionales**

**FUNDAMENTOS DE LAS PRUEBAS DEL SOFTWARE**

**Comprobabilidad.** James Bach1 proporciona la siguiente definición de comprobabilidad: “La

*comprobabilidad del software* significa simplemente saber con cuánta facilidad puede probarse

[un programa de cómputo].” Las siguientes características conducen a software comprobable.**.**

*Operatividad*. “Mientras mejor funcione, más eficientemente puede probarse.”

*Observabilidad*. “Lo que ve es lo que prueba.”.

*Controlabilidad*. “Mientras mejor pueda controlar el software, más podrá automatizar y optimizar

las pruebas.”

*Descomponibilidad*. “Al controlar el ámbito de las pruebas, es posible aislar más rápidamente

los problemas y realizar pruebas nuevas y más inteligentes.”

*Simplicidad*. “Mientras haya menos que probar, más rápidamente se le puede probar.”

*Estabilidad*. “Mientras menos cambios, menos perturbaciones para probar.”

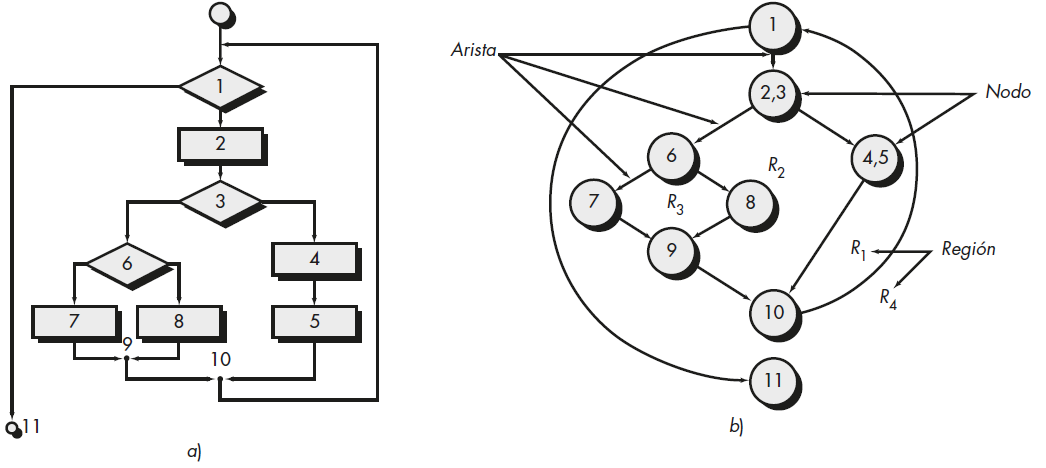
*Comprensibilidad*. “Mientras más información se tenga, se probará con más inteligencia.”

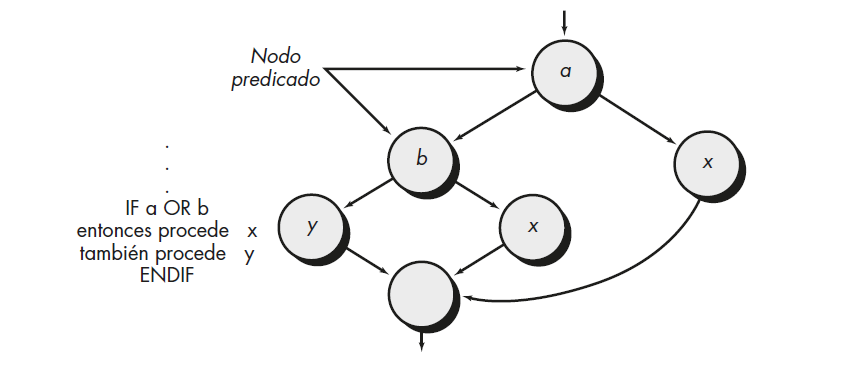
Pruebas de caja blanca son pruebas directamente al código en todo caso a cada componente del software

Pruebas de caja negra son en detalle pruebas al software en general específicamente ala parte visible para el usuario, a los requerimientos funcionales del software

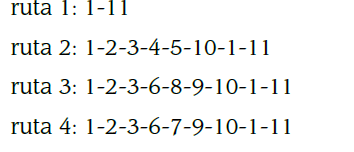
Pruebas de ruta básica son en pocas palabras contar las condiciones o ifs del código o parte del código después graficarlo en un gráfico de flujo y ver las diferentes trayectorias que hay.

Graficos de flujo contiene nodos aristas regiones y nodos predicado

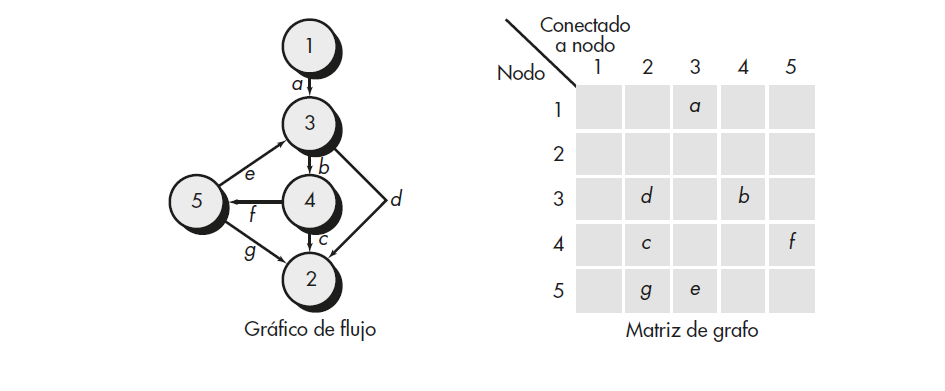




Rutas linealmente independientes



Matris de grafo representación tabular de un gráfico de flujo



Prueba de condición

Prueba de flujo de datos

La *prueba de bucle* es una técnica de prueba de caja blanca que se enfoca exclusivamente en

la validez de los constructos bucle. Pueden definirse cuatro clases diferentes de bucles [Bei90]:

simples, concatenados, anidados y no estructurados

Pruebas de caja negra

• ¿Cómo se prueba la validez funcional?

• ¿Cómo se prueban el comportamiento y el rendimiento del sistema?

• ¿Qué clases de entrada harán buenos casos de prueba?

• ¿El sistema es particularmente sensible a ciertos valores de entrada?

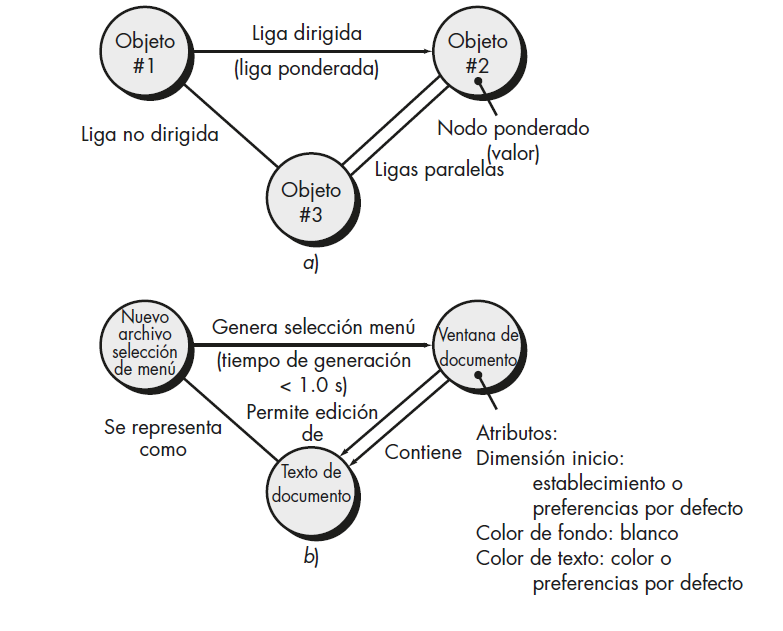
• ¿Cómo se aíslan las fronteras de una clase de datos?

• ¿Qué tasas y volumen de datos puede tolerar el sistema?

• ¿Qué efecto tendrán sobre la operación del sistema algunas combinaciones específicas

de datos?

Métodos de prueba basados en graficos



Prueba de arreglo ortogonal= esta prueba se puede hacer cuando la función o el procedimiento tiene datos que se pueden contar datos limitados o pocos

Prueba basada en modelo

La *prueba basada en modelo* (PBM) es una técnica de prueba de caja negra que usa la información

contenida en el modelo de requerimientos como la base para la generación de casos de

prueba.

**Pruebas de interfaces gráficas de usuario**

**Prueba de arquitecturas cliente-servidor**

**Pruebas de función de aplicación.** La funcionalidad de las aplicaciones cliente se

prueba usando los métodos analizados anteriormente en este capítulo y en los capítulos

19 y 20. En esencia, la aplicación se prueba en forma independiente con la intención de

descubrir errores en su operación.

• **Pruebas de servidor.** Se prueban las funciones de coordinación y gestión de datos del

servidor. También se considera el rendimiento del servidor (tiempo de respuesta global y

cantidad de datos transmitidos).

• **Pruebas de base de datos.** Se prueban la precisión y la integridad de los datos almacenados

por el servidor. Se examinan las transacciones colocadas por las aplicaciones cliente para asegurar que los datos se almacenen, actualicen y recuperen de manera

adecuada. También se prueba la forma de archivar.

• **Pruebas de transacción.** Se crea una serie de pruebas para garantizar que cada clase

de transacciones se procese de acuerdo con los requerimientos. Las pruebas se enfocan

en comprobar lo correcto del procesamiento y también en los conflictos de rendimiento

(por ejemplo, tiempos de procesamiento de transacción y volumen de transacción).

• **Pruebas de comunicación de red.** Estas pruebas verifican que la comunicación entre

los nodos de la red ocurren de manera correcta y que el mensaje que pasa, las transacciones

y el tráfico de red relacionado ocurren sin errores. Como parte de estas pruebas,

también pueden realizarse pruebas de seguridad de red.

**Prueba para sistemas de tiempo real**

**Prueba de tareas.** El primer paso en la prueba del software en tiempo real es probar

cada tarea de manera independiente.

**Prueba de comportamiento.** Con modelos de sistema creados con herramientas automatizadas,

es posible simular el comportamiento de un sistema en tiempo real y

examinar su comportamiento como consecuencia de eventos externos.

**Prueba intertarea.** Una vez aislados los errores en las tareas individuales y en el

comportamiento del sistema, las pruebas se cambian a los errores relacionados con el

tiempo.

**Prueba de sistema.** Al integrar software y hardware, se lleva a cabo un amplio rango

de pruebas del sistema con la intención de descubrir errores en la interfaz softwarehardware.

*Nombre del patrón:* **PairTesting**

*Resumen:* Patrón orientado a proceso, **PairTesting** describe una técnica que es análoga a la programación

por parejas (capítulo 3) en la que dos examinadores trabajan en conjunto para diseñar y

ejecutar una serie de pruebas que pueden aplicarse a actividades de prueba de unidad, integración o

validación.

*Nombre del patrón:* **SeparateTestInterface**

*Resumen:* Hay necesidad de probar cada clase en un sistema orientado a objetos, incluidas “clases

internas” (es decir, clases que no exponen alguna interfaz afuera del componente que los usa). El

patrón **SeparateTestInterface** describe cómo crear “una interfaz de prueba que puede usarse para

describir pruebas específicas sobre clases que son visibles solamente de manera interna en un componente”

[Lan01].

*Nombre del patrón:* **ScenarioTesting**

*Resumen:* Una vez realizadas las pruebas de unidad e integración, hay necesidad de determinar si

el software se desempeñará en forma que satisfaga a los usuarios. El patrón **ScenarioTesting** describe

una técnica para revisar el software desde el punto de vista del usuario. Un fallo en este nivel

indica que el software fracasó para satisfacer un requisito visible del usuario [Kan01].