INGENIERÍA DE APLICACIONES

Verificación y Validación de Software

Dra. María Luján Ganuza
mlg@cs.uns.edu.ar
DCIC - Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2019

Temario

Testing de Software

Proceso de Verificación y Validación de Software

Modelo de Testing Tradicional

Etapas de Testing de Software Comercial

- Pruebas de Desarrollo
- Pruebas de Lanzamiento
- Pruebas de Usuario

Testing de Software

El objetivo del testing es mostrar que un programa **hace lo que se pretende** y **descubrir los defectos** antes de que se ponga en uso.

Cuando se testea el software, se ejecuta un programa utilizando **datos artificiales** y se verifican los resultados de la ejecución de la prueba en busca de **errores**, **anomalías** o **información** sobre los atributos no funcionales del programa.

Testing de Software

Objetivos:

- Demostrar al desarrollador y al cliente que el software cumple con sus requisitos.

PRUEBA DE VALIDACIÓN

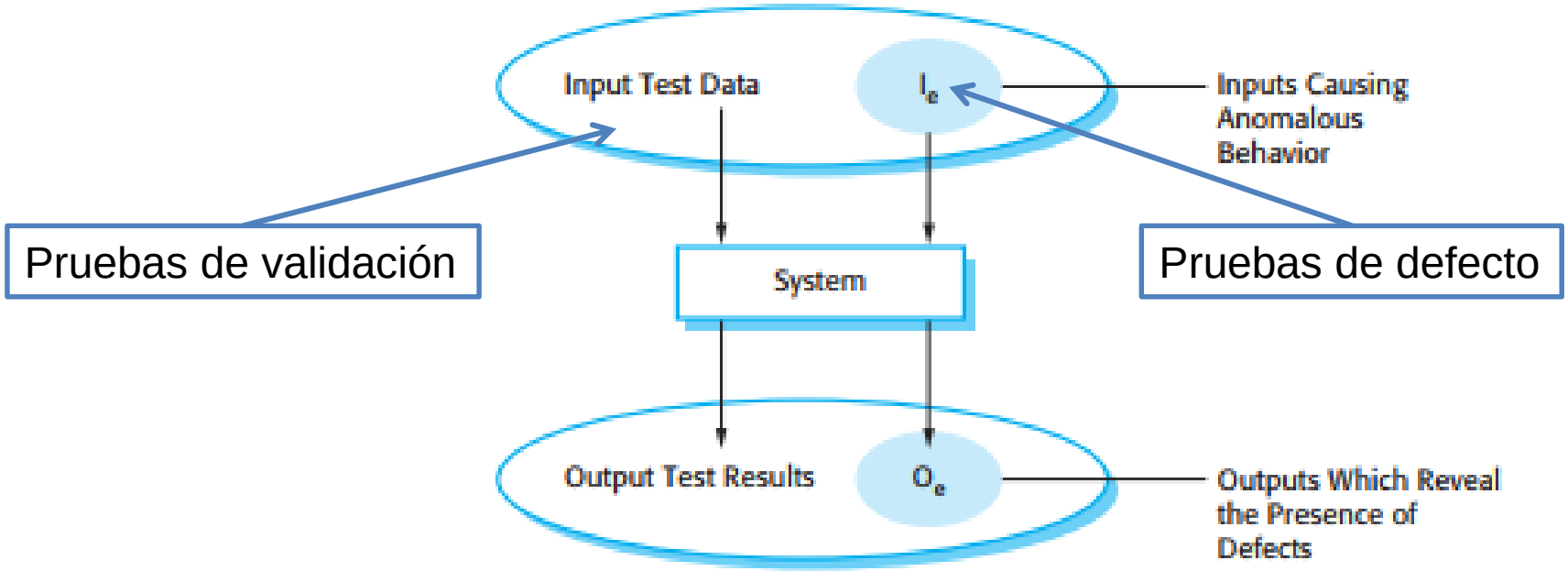
- Descubrir situaciones en las que el comportamiento del software sea incorrecto, indeseable o no se ajuste a su especificación.

PRUEBA DE DEFECTOS

Testing de Software

- **Pruebas de Validación:** Se espera que el sistema funcione correctamente utilizando un conjunto determinado de casos de prueba que reflejen el uso esperado del sistema.
- **Pruebas de Defectos:** los casos de prueba están diseñados para exponer defectos, pueden ser deliberadamente oscuros y no necesitan reflejar cómo se usa normalmente el sistema.

Testing de Software



Modelo de Entrada-Salida de un Programa de Testing

Testing de Software

Las pruebas solo pueden mostrar la presencia de errores, no su ausencia

(Dijkstra et al., 1972)

Las pruebas no pueden demostrar que el software está libre de defectos o que se comportará como se especifica en cada circunstancia.

Siempre es posible que una prueba que haya pasado por alto pueda descubrir más problemas con el sistema.

Verificación y Validación de Software

- El testing es parte de un proceso más amplio de **verificación y validación de software**.
- Los procesos de V&V se ocupan de verificar que el software que se está desarrollando cumpla con sus especificaciones y ofrezca la funcionalidad esperada.
- Estos procesos de verificación comienzan tan pronto como los requisitos estén disponibles y continúan en todas las etapas del proceso de desarrollo.

Verificación y Validación de Software

- **Validación:**
  ¿estamos construyendo el producto correcto?
  Objetivo: garantizar que el software cumpla con las expectativas del cliente.

- **Verificación:**
  ¿estamos construyendo el producto correctamente?
  Objetivo: verificar que el software cumpla con los requisitos funcionales y no funcionales establecidos.

Verificación y Validación de Software

La verificación y validación no puede hacerse en base a intuición, experiencia o suerte. Se requiere seguir principios y técnicas rigurosas y adecuadas.

Tanto el proceso de desarrollo de software, como los productos generados, deben ser verificados.

Se testea código, modelos, especificaciones, productos terminados… el mismo proceso de testeo debería ser testeado!

Verificación y Validación de Software

La actividad de testing debe realizarse en momentos diferentes con objetivos, técnicas y personal diferente.

Quién desarrolla el producto, no debería verificar su producto. El ego humano no permite ser objetivo, no queremos aceptar nuestras equivocaciones y queremos terminar pronto.

Verificación y Validación de Software

La presencia de defectos en el software complejo no puede evitarse completamente. A veces, algunos defectos pueden ser tolerados.

En la práctica, la correctitud es relativa.

Verificación y Validación de Software

El objetivo final de los procesos de V&V es establecer la confianza de que el sistema es "adecuado para el propósito". Esto depende de:

- El propósito del sistema.
- Las expectativas de los usuarios
- El entorno de comercialización

Verificación y Validación de Software

El objetivo final de los procesos de V&V es establecer la confianza de que el sistema es "adecuado para el propósito". Esto depende de:

- **El propósito del sistema:** Cuanto más crítico sea el software, más importante será su fiabilidad.

Verificación y Validación de Software

El objetivo final de los procesos de V&V es establecer la confianza de que el sistema es "adecuado para el propósito". Esto depende de:

- **Las expectativas de los usuarios:** Debido a experiencias con software defectuoso, muchos usuarios tienen pocas expectativas sobre la calidad del software.
- Cuando se instala un nuevo sistema, los usuarios pueden tolerar fallas (los beneficios del uso superan los costos).
- A medida que el software madura, los usuarios esperan que sea más confiable.

Verificación y Validación de Software

El objetivo final de los procesos de V&V es establecer la confianza de que el sistema es "adecuado para el propósito". Esto depende de:

- **El entorno de comercialización:** En un entorno competitivo, una empresa de sw puede decidir lanzar un programa antes de que haya sido completamente probado y depurado(quiere ser el primero en el mercado).
- Si un producto de software es muy barato, los usuarios pueden estar dispuestos a tolerar un nivel más bajo de confiabilidad.

Verificación y Validación de Software

INSPECCIONES Y REVISIONES

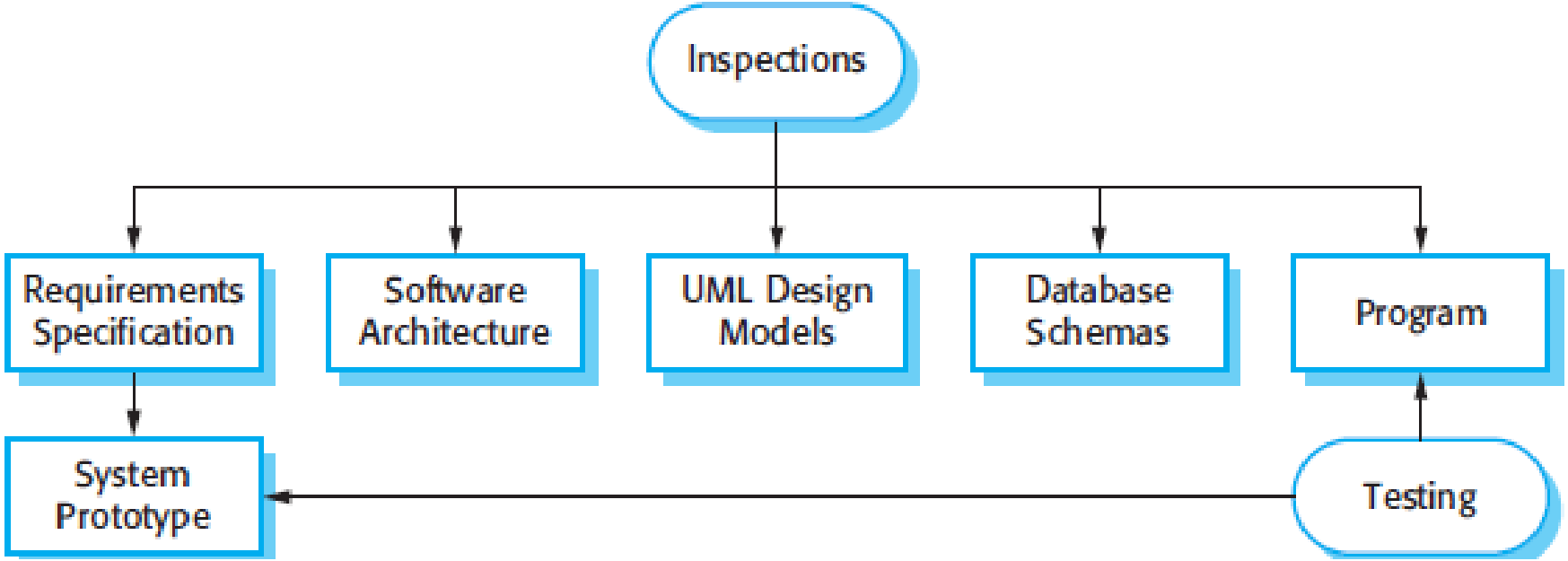
Además del testing, el proceso de V&V puede incluir **inspecciones y revisiones** de software.

Las inspecciones y revisiones analizan y verifican los **requisitos del sistema**, **los modelos de diseño, el código fuente del programa** e incluso **las pruebas de sistema propuestas**.

Estas son las llamadas **técnicas de V&V estáticas** en las que no es necesario ejecutar el software.

Verificación y Validación de Software

TESTING E INSPECCIONES



Ventajas de la inspección sobre el testing

- Durante el testeo, los errores pueden enmascarar (ocultar) otros errores. La inspección es estática, no tiene que preocuparse por las interacciones entre los errores.
- Las versiones incompletas de un sistema se pueden inspeccionar sin costos adicionales.
- Una inspección puede considerar, además de la búsqueda de errores, atributos de calidad más amplios (puede buscar ineficiencias, algoritmos inapropiados y un estilo de programación deficiente).

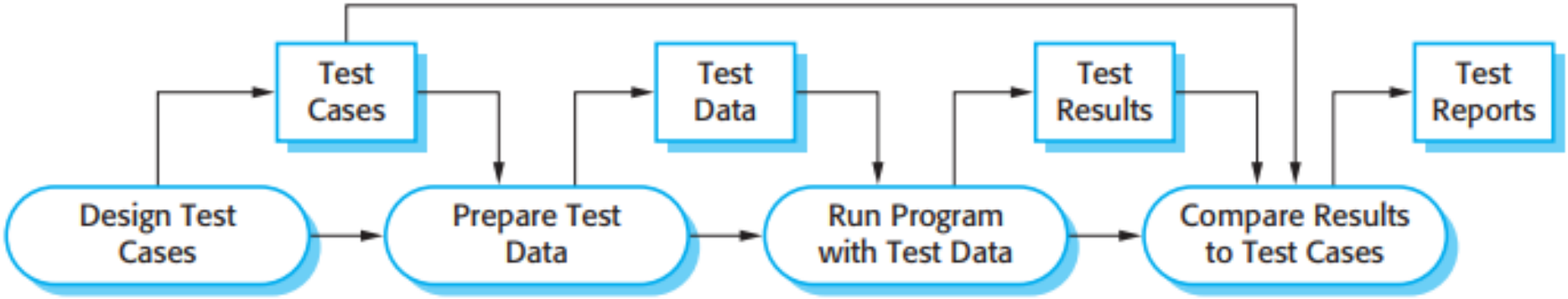
Inspección vs. Testing

- Sin embargo, las inspecciones no pueden reemplazar el testing de software.

- Las inspecciones no son buenas para descubrir defectos que surgen debido a interacciones inesperadas entre diferentes partes de un programa, problemas de tiempo o problemas con el rendimiento del sistema.

- Además, puede ser difícil y costoso formar un equipo de inspección por separado.

Modelo de Proceso de Testing Tradicional



- *Los **casos de prueba** son especificaciones de las entradas a la prueba y el resultado esperado del sistema (los resultados de la prueba).*
- *Los **datos de prueba** son las entradas que se han diseñado para probar un sistema.*
- *Los resultados esperados se comparan automáticamente con los resultados previstos, por lo que no es necesario que una persona busque errores y anomalías en la ejecución de la prueba.*

Etapas de Prueba de un Sistema Comercial

1. **Pruebas de desarrollo**, donde el sistema se prueba durante el desarrollo para descubrir fallas y defectos.

2. **Prueba de lanzamiento**, donde una versión completa del sistema se prueba antes de que se lance a los usuarios. El objetivo de las pruebas de lanzamiento es verificar que el sistema cumpla con los requisitos de las partes interesadas del sistema.

3. **Prueba de usuario**, donde los usuarios o usuarios potenciales prueban el sistema en su propio entorno.

Testing de Software

En la práctica, el proceso de testing generalmente involucra una mezcla de **pruebas manuales** y **automáticas**.

- En las **pruebas manuales**, un tester ejecuta el programa con algunos datos de prueba y compara los resultados con sus expectativas.
- En las **pruebas automatizadas**, las pruebas se codifican en un programa que se ejecuta cada vez que se testea el sistema en desarrollo. Por lo general, es más rápido que las pruebas manuales.

Pruebas de Desarrollo

Las pruebas de desarrollo incluyen todas las actividades de prueba que lleva a cabo el equipo que desarrolla el sistema.

- El tester suele ser el programador que desarrolló ese software.
- En **sistemas críticos**, el proceso involucra un grupo de testing separado del equipo de desarrollo. Ellos son responsables de desarrollar pruebas y mantener registros detallados de los resultados de las pruebas.

Pruebas de Desarrollo

Durante el desarrollo, las pruebas se pueden llevar a cabo en tres niveles de granularidad:

- **Pruebas unitarias**, donde se prueban unidades de programas individuales o clases de objetos.
- **Pruebas de componentes**, donde se integran varias unidades individuales para crear componentes compuestos.
- **Pruebas del sistema**, donde el sistema se prueba como un todo.

Pruebas de Desarrollo Unitarias

Las **pruebas unitarias** prueban unidades de programas individuales o clases de objetos.

Se enfocan en probar la funcionalidad de objetos o métodos.

Las **funciones** o **métodos individuales** son el tipo más simple de componente. Las pruebas consisten en llamadas a estas rutinas con diferentes parámetros de entrada.

Pruebas de Desarrollo Unitarias

Al testear clases de objetos, las pruebas deben proporcionar cobertura de todas las características del objeto:

- Debe testear todas las operaciones asociadas con el objeto;
- Debe establecer y verificar el valor de todos los atributos asociados con el objeto;
- Debe poner el objeto en todos los estados posibles. Esto significa que debe simular todos los eventos que causan un cambio de estado.

Pruebas Unitarias Automatizadas

- Siempre que sea posible, las pruebas unitarias deben automatizarse.
- Se utiliza un marco de automatización de pruebas (como JUnit) para escribir y ejecutar las pruebas.
- Los marcos de prueba de unidades proporcionan clases de prueba genéricas mediante las cuales se crean casos de prueba específicos.
- Luego se ejecutan todas las pruebas y se informan, a menudo a través de alguna GUI, sobre el éxito o el fracaso de las pruebas.

Pruebas Unitarias Automatizadas

Una prueba automatizada tiene tres partes:

- **Configuración**: donde inicializa el sistema con el caso de prueba, es decir, las entradas y las salidas esperadas.
- **Llamada**: donde llama al objeto o método a probar.
- **Aserción:** en la que compara el resultado de la llamada con el resultado esperado. Si la afirmación se evalúa como verdadera, la prueba ha sido exitosa; si es falso, entonces ha fallado.

Pruebas Unitarias Automatizadas: JUnit

```
public class MyTests {

    @Test
    public void multiplicationOfZeroIntegersShouldReturnZero() {
        MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
        assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");
        assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");
        assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");
    }
}
```

Casos de Prueba Unitarios

Es importante que elija casos de prueba unitarios **efectivos.**

Efectividad, en este contexto, significa dos cosas:

1. Los casos de prueba deben mostrar que, **cuando se usa como se esperaba**, el componente que se está probando **hace lo que se supone que debe hacer**.
2. **Si hay defectos** en el componente, **estos deben ser revelados** por los casos de prueba.

Casos de Prueba Unitarios

Por lo tanto, se deben escribir **dos tipos de casos de prueba**.

1. Uno debe reflejar el funcionamiento normal y mostrar que el componente funciona.
2. El otro debe usar entradas anormales para verificar que se procesen correctamente y no bloqueen el componente.

Casos de Prueba Unitarios

Estrategias para elegir casos de prueba:

1. **Pruebas de partición**, donde se identifican grupos de entradas que tienen características comunes y deben procesarse de la misma manera.

2. **Pruebas basadas en guías**, donde usa pautas de prueba para elegir casos de prueba. Estas pautas reflejan la experiencia previa de los tipos de errores que los programadores a menudo cometen al desarrollar componentes.

Casos de Prueba Unitarios

Pruebas de partición

Los datos de entrada y los resultados de salida de un programa a menudo se dividen en varias clases diferentes con **características comunes**.

Los programas normalmente se **comportan de manera comparable para todos los miembros de una clase**.

Estas clases a veces se llaman **particiones de equivalencia** o dominios

Casos de Prueba Unitarios

Pruebas de partición

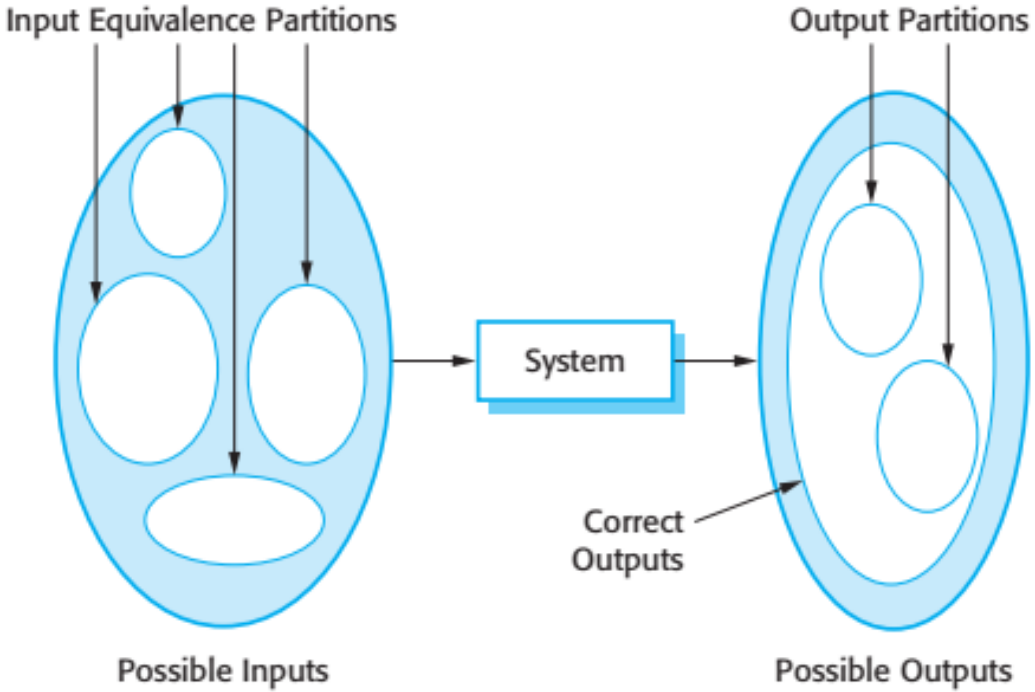
Un enfoque sistemático para el diseño de casos de prueba se basa en **la identificación de todas las particiones** de entrada y salida para un sistema o componente.

Los casos de prueba están diseñados para que las entradas o salidas se encuentren dentro de estas particiones.

Las pruebas de partición se pueden usar para diseñar casos de prueba para sistemas y componentes.

Casos de Prueba Unitarios

Pruebas de partición



Casos de Prueba Unitarios

Pruebas de partición

- Una vez identificado un conjunto de particiones, se diseñan casos de prueba de cada una de estas particiones.
- Una buena regla general es elegir casos de prueba en los límites de las particiones, y casos cercanos al punto medio de la partición.

Casos de Prueba Unitarios

Pruebas basadas en guías

También puede usar pautas de prueba para ayudar a elegir casos de prueba.

Las directrices encapsulan el conocimiento de qué tipos de casos de prueba son efectivos para descubrir errores.

Casos de Prueba Unitarios

Pruebas basadas en guías

Por ejemplo, cuando prueba programas con secuencias, matrices o listas, las pautas incluyen:

- Pruebas con secuencias que tienen un solo valor.
- Usa diferentes secuencias de diferentes tamaños en diferentes pruebas.
- Obtenga pruebas para acceder a los primeros, medios y últimos elementos de la secuencia.

Casos de Prueba Unitarios

Pruebas basadas en guías

Algunas de las **pautas más generales**:

- Elija entradas que obliguen a generar todos los mensajes de error;
- Entradas que causen el desbordamiento de los buffers de entrada;
- Repetir la misma entrada o serie de entradas numerosas veces;
- Forzar salidas inválidas;
- Forzar los resultados de cálculo para que sean demasiado grandes o demasiado pequeños.

Pruebas de Componentes

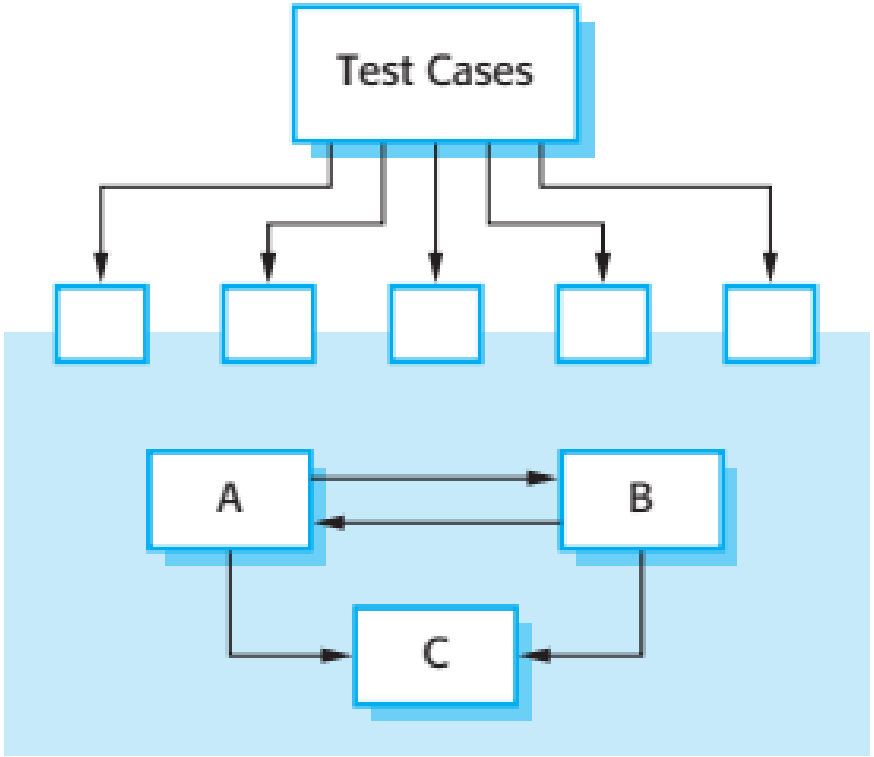
Los **componentes de software** a menudo son componentes compuestos que se componen de varios objetos que interactúan.

Se accede a la funcionalidad a través de **la interfaz del componente**.

Las **pruebas de componentes** se enfocan en mostrar que la interfaz del componente se comporta de acuerdo con su especificación.

Puede suponer que las pruebas unitarias en los objetos individuales dentro del componente se han completado.

Pruebas de Componentes



Pruebas de Componentes

Tipos de Interfaces de Componentes:

- Interfaces de parámetros
- Interfaces de memoria compartida
- Interfaces de procedimiento
- Interfaces de envío de mensajes

Pruebas de Componentes

Tipos de Interfaces de Componentes:

- **Interfaces de parámetros:** Interfaces en las que los datos o, a veces, las referencias de funciones se pasan de un componente a otro. Los métodos en un objeto tienen una interfaz de parámetros.
- Interfaces de memoria compartida
- Interfaces de procedimiento
- Interfaces de envío de mensajes

Pruebas de Componentes

Tipos de Interfaces de Componentes:

- Interfaces de parámetros
- **Interfaces de memoria compartida:** Interfaces en las que un bloque de memoria se comparte entre los componentes. Los datos son colocados en la memoria por un subsistema y recuperados desde allí por otros subsistemas.
- Interfaces de procedimiento
- Interfaces de envío de mensajes

Pruebas de Componentes

Tipos de Interfaces de Componentes:

- Interfaces de parámetros
- Interfaces de memoria compartida
- **Interfaces de procedimiento**: Interfaces en las que un componente encapsula un conjunto de procedimientos que otros componentes pueden invocar.
- Interfaces de envío de mensajes.

Pruebas de Componentes

Tipos de Interfaces de Componentes:

- Interfaces de parámetros
- Interfaces de memoria compartida
- Interfaces de procedimiento
- **Interfaces de envío de mensajes:** Interfaces en las que un componente solicita un servicio de otro componente pasándole un mensaje.

Pruebas de Componentes

Estos errores de interfaz se dividen en tres clases:

- **Uso indebido de la interfaz** Común en las interfaces de parámetros (parámetros de tipo incorrecto, orden incorrecto, o número incorrecto de parámetros).
- **Malentendido de la interfaz** Un componente no entiende la especificación del componente invocado y hace suposiciones sobre su comportamiento.
- **Errores de temporización** Se producen en sistemas en tiempo real que utilizan una memoria compartida o una interfaz de paso de mensajes. El proveedor de datos y el consumidor pueden operar a diferentes velocidades.

Pruebas de Componentes

Algunas **pautas generales** para las pruebas de interfaz son:

- Examinar el código y enumerar explícitamente cada llamada a un componente externo. Diseñar un conjunto de pruebas con valores extremos.
- Cuando se pasan punteros, siempre pruebe la interfaz con punteros nulos.
- Cuando se invoca un componente, provocar deliberadamente su falla.
- Use pruebas de estrés en sistemas de envío de mensajes. Generar muchos más mensajes de los que es probable que ocurran en la práctica. Esto revela problemas de tiempo.

Pruebas de Sistema

Las **pruebas del sistema** durante el desarrollo implican la integración de componentes y luego testear el sistema integrado.

Las pruebas del sistema comprueban que los **componentes** son **compatibles**, **interactúan correctamente** y **transfieren los datos correctos en el momento correcto** a través de sus interfaces.

Pruebas de Sistema

- Las pruebas del sistema deben centrarse en probar las **interacciones entre los componentes y los objetos** que componen un sistema.
- También pueden probar **componentes o sistemas reutilizables** para verificar que funcionen como se espera cuando se integran con componentes nuevos.
- Estas pruebas descubren errores que solo se revelan cuando un componente es utilizado por otros componentes en el sistema.
- También ayudan a encontrar **malentendido**s sobre el comportamiento de otros componentes en el sistema.

Pruebas de Lanzamiento

La **prueba de lanzamiento** prueba una versión particular de un sistema que está destinado a ser utilizado fuera del equipo de desarrollo.

Las pruebas del sistema se enfocan en descubrir errores en el sistema **(prueba de defectos)**. Las **pruebas de lanzamiento** se enfocan en verificar que el sistema cumpla con sus requisitos y que sea lo suficientemente bueno para uso externo **(pruebas de validación).**

Pruebas de Lanzamiento

- Deben mostrar que el sistema brinda su **funcionalidad**, **rendimiento** y **confiabilidad** especificados, y que **no falla** durante el uso normal.
- Debe tener en cuenta todos los **requisitos** del sistema, no solo los requisitos de los usuarios finales del sistema.
- Es generalmente un proceso de prueba de **caja negra** donde las pruebas se derivan de la especificación del sistema.

Pruebas de Usuario

La **prueba del usuario** es una etapa en el proceso de prueba en el cual los usuarios o clientes proporcionan información y asesoramiento sobre las pruebas del sistema.

Esto puede implicar **probar formalmente** un sistema, o **probar informalmente** un nuevo producto de software para ver si les gusta y si hace lo que necesitan.

Las pruebas de usuario son esenciales, incluso cuando se han llevado a cabo pruebas exhaustivas del sistema.

Pruebas de Usuario

Tipos de Pruebas de Usuario:

1. **Pruebas alfa**, donde los usuarios del software trabajan con el equipo de desarrollo para probar el software en el sitio del desarrollador.
2. **Pruebas beta**, donde una versión del software se pone a disposición de los usuarios para permitirles experimentar y plantear problemas que descubren.
3. **Pruebas de aceptación**, donde los clientes prueban un sistema para decidir si está listo o no para ser aceptado y desplegado en el entorno del cliente

Material Bibliográfico

- Ian Sommerville. 2010. Software Engineering (9th ed.). Addison-Wesley Publishing Company, USA.

- Cadle, J., & Yeates, D. (Eds.). 2004. *Project management for information systems*. Pearson education.

- Epstein, D., & Maltzman, R. 2013. *Project workflow management: a business process approach*. J. Ross Publishing.