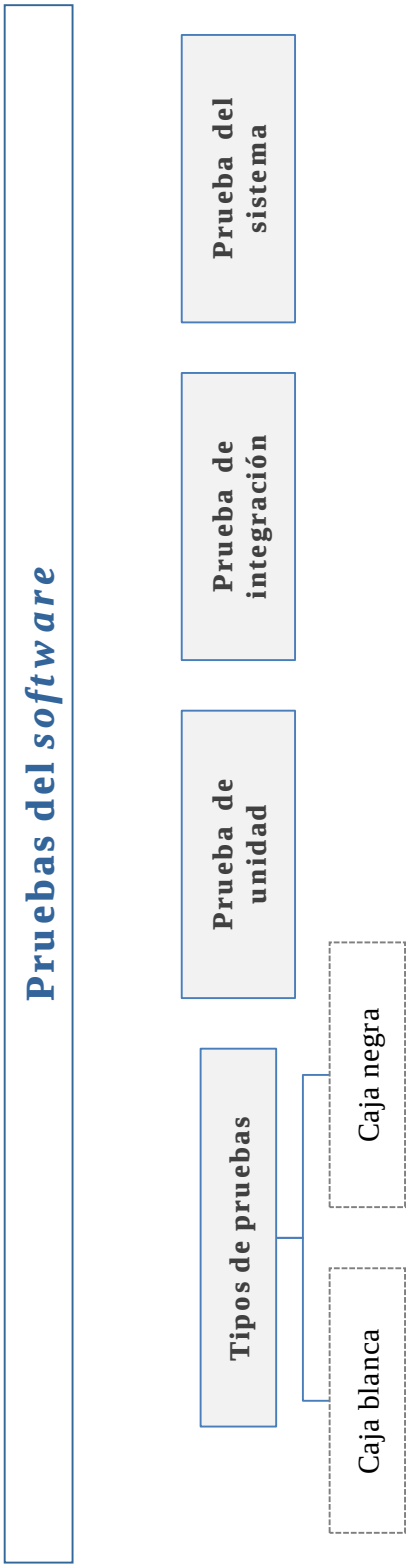
Pruebas de *software*

TEMA 12

Esquema



Ideas clave

12.1. ¿Cómo estudiar este tema?

Para estudiar este tema debes leer el **capítulo 18 (páginas 310-318) y el capítulo 17 (páginas 281-299)** del siguiente libro, disponible en el aula virtual bajo licencia CEDRO:

Pressman, R. S. (2002). *Ingeniería del software. Un enfoque práctico*. Madrid: McGraw Hill.

Para comprobar si has comprendido los conceptos puedes realizar el test de autoevaluación del tema.

En este tema vamos a ver los diferentes tipos de pruebas que se pueden realizar en el *software*.

12.2. Introducción a las pruebas del software

El desarrollo del *software* implica la realización de pruebas para **garantizar la calidad** del *software*. En la etapa de pruebas del *software* se diseñan casos de prueba, cuya finalidad es **descubrir errores**.

El diseño de casos de prueba debe centrarse en el **conjunto de casos de prueba** que tengan mayor probabilidad de descubrir errores en el *software*. Un buen caso de prueba será aquel que tiene mayor probabilidad de descubrir un nuevo error.

12.3. Tipos de pruebas

El *software* se puede probar de dos formas:

Conociendo el funcionamiento interno	Conociendo la tarea específica para la que fue diseñado
Pruebas de caja blanca	**Pruebas de caja negra**

Pruebas de caja blanca

Las pruebas de caja blanca o transparente se centran en la **estructura del código**. Su objetivo es comprobar los flujos dentro de cada unidad (función, clase, método, etc.).

En las pruebas de caja blanca se examinan:

Caminos independientes	Estructuras de decisión
Bucles	Estructuras de datos internas

Las principales pruebas de caja blanca son:

- **Prueba del camino básico**: se debe diseñar un caso de prueba por cada camino independiente.

- **Prueba de la estructura de control**: se deben diseñar casos de prueba que evalúen las condiciones del *software* a cierto/falso y casos de prueba en los que se ejecuten los bucles N veces.

Pruebas de caja negra

Las pruebas de caja negra o transparente se centran en la **funcionalidad** y la **interfaz** del *software*, sin tener en cuenta el funcionamiento interno y su estructura.

En las pruebas de caja negra se examinan:

Entradas de datos	Salida o respuesta
Integridad de la información	Operatividad de las funciones

Las principales pruebas de caja negra son:

- **Prueba de partición equivalente**: los datos de entrada y los de salida se agrupan en clases diferentes, de forma que el software se comporta de la misma forma para todos los miembros de la clase.

- **Prueba de análisis de valores límites**: complementa la prueba de partición equivalente. Se eligen valores de los datos de entrada en los extremos.

12.4. Prueba de unidad

El objetivo de la prueba de unidad es **verificar el correcto funcionamiento de un módulo de código**. También son llamadas:

- **Pruebas modulares**, ya que determinan si el funcionamiento de un módulo es correcto.
- **Pruebas de caja blanca**, ya que utilizan métodos de caja blanca.

En este tipo de pruebas es necesario determinar si el tamaño del módulo permitirá realizar las pruebas de forma sencilla. Si el módulo es muy grande, se puede recomendar separar en módulos más sencillos.

Se prueba:

Interfaz	Estructura de datos
Condiciones límite	Caminos independientes
Caminos de manejo de errores	

12.5. Prueba de integración

El objetivo de la prueba de integración es **detectar errores de interacción** entre los distintos módulos.

Se debe utilizar una integración incremental, en la que el programa se construye y se prueba en pequeños segmentos.

Las principales estrategias de integración incremental son:

Integración descendente	**Integración ascendente**

Integración incremental ascendente

- Los módulos de más bajo nivel se juntan en grupos que realizan alguna función específica.
- Mediante un *driver* se realizan llamadas a los módulos, obteniendo unos resultados a partir de los datos de prueba introducidos.
- Una vez probado el grupo de módulos, el *driver* es sustituido por los módulos de nivel superior, hasta tener el programa completo.

Las **ventajas** de este tipo de integración son:

- La generación de datos de entrada para las pruebas es más fácil.
- Facilita la comprobación de los resultados de las pruebas.
- Permite resolver primero los errores de los módulos de nivel inferior.

Las **desventajas** son:

- Requiere la programación de los *drivers*.
- Hasta que no se añade el último bulto, no se prueba el programa completo.

Integración incremental descendente

- Comienza con el módulo de mayor nivel, escribiendo módulos ficticios que simulan a los subordinados.
- Cada vez que se incorpora un módulo hay que probar todo el conjunto.
- Cuando se finaliza una prueba, se sustituye el módulo ficticio por el real.
- Cuando se incorpora el último módulo, todos los módulos que lo invocan deben estar probados.

Ventajas:

- Los errores en módulos de nivel superior se detectan pronto.
- Estructura completa del programa desde el principio.
- Define primero las interfaces.

Desventajas:

- Necesidad de escribir módulos ficticios.
- Dificultad para introducir los datos de entrada hasta que se incorporan los módulos de entrada de datos.
- Retrasa la prueba de ciertos módulos.

12.6. Prueba de validación

El objetivo de la prueba de validación es **verificar que el *software* funciona de acuerdo a los requisitos** establecidos.

En las pruebas de validación se utilizan **métodos de caja negra**.

Las pruebas de validación que habitualmente se realizan son:

Pruebas de aceptación	Pruebas alfa	Pruebas beta
Permiten que el usuario final valide el *software*.	Permiten que un usuario final valide el *software* en un entorno controlado.	Permiten que los usuarios finales validen el *software* en un entorno no controlado.

12.7. Prueba del sistema

El objetivo de la prueba del sistema es **verificar el sistema como un todo**, teniendo en cuenta los requerimientos generales y todas las parte del sistema.

Las principales pruebas del sistema son:

Prueba de recuperación	Prueba de seguridad
Verifica la recuperación del *software* ante un fallo.	Verifica los mecanismos de protección del *software*.
Prueba de resistencia	**Prueba de rendimiento**
Verifica el comportamiento del *software* ante situaciones anormales.	Verifica el rendimiento del *software* en tiempo de ejecución.

Lo + recomendado

No dejes de leer...

Pruebas unitarias en Java

Tutorial sobre `JUnit`.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://felinfo.blogspot.com.es/2013/06/junit-en-eclipse-pruebas-unitarias-en.html

Unit test

Página de Microsoft en la que se profundiza sobre la prueba unitaria.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://msdn.microsoft.com/es-es/library/jj130731.aspx

Pruebas del sistema

Seminario sobre las pruebas unitarias.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://es.slideshare.net/juandiaz89/estrategias-de-aplicacin-de-pruebas-del-sistema?qid=185f7038-c8da-4051-a05f-7ef95d173155&v=default&b=&from_search=4

+ Información

A fondo

Automatización de pruebas

Página de Microsoft sobre la automatización de pruebas.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://msdn.microsoft.com/es-es/library/ff472576.aspx

Pruebas unitarias

Seminario sobre las pruebas unitarias.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://es.slideshare.net/microgestion/workshop-pruebas-unitarias

Introducción a `JUnit`

Tutorial sobre `JUnit`.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://es.slideshare.net/ikercanarias/introduccin-a-junit?related=1

Pruebas del sistema y pruebas de aceptación

Seminario sobre pruebas del sistema y pruebas de aceptación.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://es.slideshare.net/ikercanarias/introduccin-a-junit?related=1

Test

1. Las pruebas de caja blanca:

A. No requieren un conocimiento interno del *software*.

B. Comprueban la entrada y salida de datos.

C. Se centran en la estructura del código.

D. Verifican el funcionamiento del sistema en su totalidad.

2. La prueba de partición equivalente:

A. Es una prueba de caja negra.

B. Es una prueba de caja blanca.

C. Es una prueba de caja gris.

D. Se utiliza para verificar los bucles del código.

3. Las pruebas de unidad:

A. Detectan errores de interacción entre los módulos.

B. Verifican que se cumplen los requisitos funcionales del sistema.

C. Suelen utilizar pruebas de caja blanca.

D. Suelen utilizar pruebas de caja negra.

4. Las estructuras de datos se examinan en:

A. Las pruebas de validación.

B. Las pruebas de unidad.

C. Las pruebas de integración.

D. Las pruebas del sistema.

5. En la integración ascendente:

A. Se combinan los módulos de alto nivel en grupos.

B. Se combinan los módulos de bajo nivel en grupos.

C. Se prueba cada módulo de forma independiente.

D. Los módulos se integran de arriba hacia abajo.

6. Las pruebas alfa y beta se utilizan en:

A. Pruebas de integración.

B. Pruebas del sistema.

C. Pruebas de unidad.

D. Pruebas de validación.

7. En las pruebas de validación se utilizan

A. Pruebas de caja negra.

B. Pruebas de caja blanca.

C. Pruebas de seguridad.

D. Pruebas de recuperación.

8. La prueba de rendimiento se realiza en:

A. Pruebas de unidad.

B. Pruebas de validación.

C. Pruebas del sistema.

D. Pruebas de integridad.

9. La integridad incremental:

A. Prueba todo el programa en su conjunto.

B. Prueba pequeños segmentos del programa.

C. Dificulta la corrección de errores.

D. Utiliza el enfoque del *big bang*.

10. Las pruebas de resistencia:

A. Verifica los mecanismos de protección.

B. Verifica la recuperación del *software*.

C. Prueban el rendimiento del *software*.

D. Enfrentan a los programas a situaciones anormales.