

# REPASO SESIONES S06..S11

## VERIFICACIÓN

El producto está implementado  
CORRECTAMENTE??

## PRUEBAS UNITARIAS

SESIONES S01..S05  
SESIONES P01..P05

### (S07) PRUEBAS DEL SISTEMA

OBJETIVO: **detectar defectos** en el comportamiento de la aplicación como un todo (siempre desde un PUNTO DE VISTA técnico).

Métodos de **DISEÑO** (caja negra): (1) basado en casos de uso; (2) método de transición de estados

Los casos de prueba resultantes contienen una secuencia de entradas-resultado esperado-entradas-resultado esperado ...

Para **implementar** las pruebas podemos usar Webdriver (aplicaciones web).

Los tests contienen varios "assert" (distribuidos en función de los resultados esperados "parciales" de cada caso de prueba).

La **ejecución** de las pruebas puede incluirse en el proceso de construcción de maven

### (S06) PRUEBAS DE INTEGRACIÓN

OBJETIVO: **detectar defectos** derivados de la interacción de las unidades.

Es FUNDAMENTAL definir una estrategia de integración (orden)

La selección de comportamientos a probar (**DISEÑO**) depende del tipo de interfaz.

Para **implementar** las pruebas usaremos DBUnit para integrar nuestra aplicación con una Base de Datos

La **ejecución** de las pruebas puede incluirse en el proceso de construcción de maven

#### PRUEBAS DE REGRESIÓN

Aluden al hecho de repetir las pruebas cada vez que se integran nuevos componentes (unidades)

# REPASO SESIONES S06..S11

El producto es el CORRECTO??

Las pruebas se realizan desde el punto de vista del USUARIO!!!

## VALIDACIÓN

(S07, S08, S09)

### PRUEBAS DE ACEPTACIÓN

OBJETIVO: ver en que **grado** se satisfacen las EXPECTATIVAS del cliente (**criterios de aceptación**)

### PROPIEDADES EMERGENTES FUNCIONALES

TODOS los criterios de aceptación deben ser CUANTIFICABLES!!!

Métodos de **DISEÑO** (caja negra): (1) basado en requerimientos; (2) basado en escenarios

Para **implementar** las pruebas podemos usar (para aplicaciones WEB):

Selenium IDE (S07)

API (comandos Selenese) para interaccionar con diferentes navegadores.

Los scripts se programan desde una extensión del navegador

La **ejecución** de las pruebas se realiza desde una extensión del navegador

Webdriver (S08)

API (Java) para interaccionar con diferentes navegadores.

La **ejecución** de las pruebas puede incluirse en el proceso de construcción de maven

### PROPIEDADES EMERGENTES NO FUNCIONALES

Los métodos de **DISEÑO** dependen de la propiedad a validar. P.ej.

(1) Pruebas de carga

(2) Pruebas de stress

(3) Pruebas estadísticas (fiabilidad)

(S09)

En general para validar propiedades que afectan al **RENDIMIENTO** se siguen unos pasos, entre los cuales es fundamental generar los datos de prueba usando un **PERFIL OPERACIONAL**.

JMeter es una herramienta usada para validar propiedades que afectan al rendimiento (carga, fiabilidad, escalabilidad...)

Permite **implementar** y **ejecutar** las pruebas de aceptación

# REPASO SESIONES S06..S11

## (S10) ANÁLISIS DE COBERTURA

La cobertura es una MÉTRICA que mide la EXTENSIÓN de nuestras pruebas (no la calidad de las mismas ni la del código probado).

Que nuestras pruebas tengan una determinada extensión no garantiza la efectividad de las mismas. Pero es evidente que si tenemos código no probado es imposible que dichas pruebas sean efectivas.

Podemos establecer diferentes NIVELES de cobertura:

Nivel 1: Cobertura de líneas

Nivel 2: Cobertura de decisiones

Nivel 3: Cobertura de condiciones

Nivel 4: Cobertura de decisiones+condiciones

Nivel 5: Cobertura de condiciones múltiples

Nivel 6: Cobertura de bucles

Nivel 7: Cobertura de caminos

## (S11) PLANIFICACIÓN

El plan de pruebas debe contemplar tanto "tareas" de verificación como de validación. Hay una relación de precedencia entre ellas (que es independiente del modelo de proceso).

P.ej. pr. unidad preceden a pr. de integración, y éstas preceden a pr. del sistema, etc. Además las actividades de diseño preceden a las de automatización.

Y hay una relación de precedencia entre las tareas de testing y otras tareas de desarrollo (que es independiente del modelo de proceso).

P.ej. el diseño de pruebas de aceptación depende de la especificación de requerimientos, etc.

Las actividades de pruebas pueden llegar a dirigir el desarrollo, y/o ser una parte fundamental del mismo. P.ej. las prácticas ágiles TDD, BDD e integraciones continuas.



# REPASO SESIONES P06..P10

Las prácticas de estas sesiones son bastante guiadas, y usando herramientas diferentes.

En FUNDAMENTAL tener claro lo que estamos haciendo en cada momento.

**MAVEN** es la herramienta con la que automatizamos la construcción de nuestros proyectos.

Es indispensable tener claro como debemos **configurar el pom** para cada tipo de pruebas. Así cómo dónde ubicar el código del proyecto y qué **comandos maven** usar para ejecutar cada uno de los tipos de pruebas.



No es posible usar correctamente ninguna herramienta (práctica) sin entender para qué sirve (teoría)

## (P06A) PROYECTOS MULTIMÓDULO

Maven permite construir proyectos formados por varios sub-proyectos maven a los que llamamos **módulos**

Los proyectos multimódulos pueden usar relaciones de **agregación** y **herencia**, para evitar duplicaciones a la hora de configurar el pom de cada módulo.

Además maven usa un "mecanismo **REACTOR**" para determinar el orden de construcción de los módulos en función de las dependencias existentes entre ellos.

## (P06B) INTEGRACIÓN CON UNA BD

**DBUNIT** es un API java usado para poder realizar pruebas de integración con una base de datos.

El api DBUnit nos permite **CONTROLAR** el estado de la BD antes y después de invocar a la SUT durante las pruebas.

Los tests de integración (con o sin DBUnit) son ejecutados por el plugin **failsafe**, y tienen lugar después de las pruebas unitarias. (en la fase "integration-test", y después de la fase "package")

# REPASO SESIONES P06..P10

Pruebas de aceptación de propiedades emergentes FUNCIONALES

## P07 PRUEBAS CON SELENIUM IDE

Selenium IDE no es un lenguaje de programación, por lo carece de algunas características como por ejemplo la posibilidad de parametrizar, gestión de errores e integración en una herramienta de construcción de proyectos.

Su uso está muy extendido debido a que permite generar muchas líneas de código en muy poco tiempo gracias a su capacidad para "grabar" todas las interacciones de usuario, generando comandos selenese de forma automática

Todos los comandos selenese tiene dos parámetros, uno de los cuales (target) indica la localización de un elemento en la página html

## P08 PRUEBAS CON SELENIUM WEBDRIVER

Selenium Webdriver es un API Java (también se dispone del mismo API para diferentes lenguajes de programación)

Usaremos el patron **PAGE OBJECT** para independizar los tests del código html

También podemos usar la clase **PageFactory** junto con anotaciones **@FindBy** para localizar los elementos de las páginas html de forma "lazy"

Una PO representa una página web y proporciona los mismos servicios que las páginas correspondientes.

Es fundamental tener claro cómo localizar los diferentes elementos de las páginas html y cómo interaccionar con ellos.

Sólo para aplicaciones WEB!!

Selenium IDE y Selenium Webdriver también pueden usarse para automatizar las pruebas del SISTEMA

## P09 PRUEBAS CON JMETER

JMeter es una aplicación de escritorio para realizar pruebas de aceptación de propiedades emergentes NO funcionales.

No es un lenguaje de programación. El test (plan de pruebas) está formado por diferentes elementos organizados en forma de árbol.

Cada tipo de elemento tiene asociada su propia configuración.

La ejecución del plan de pruebas NO depende del orden en el que "escribamos" elementos en el plan. Se ejecutan siguiendo un determinado orden dependiendo del tipo de elemento.

Un plan siempre debe tener como mínimo: un grupo de hilos, aserciones, samplers y listeners

Los datos recopilados por los listeners SIEMPRE tienen que analizarse después de ejecutar el plan de pruebas para poder saber en qué grado se satisface o no la propiedad no funcional correspondiente.

Usaremos Jmeter para evaluar las expectativas del cliente que tengan que ver con el RENDIMIENTO

## P10 ANÁLISIS DE COBERTURA

JaCoCo nos permite analizar la cobertura de nuestras pruebas a nivel de:

- Instrucciones byte code cubiertas
- Branches ejercitadas (JaCoCo llama branch tanto a una condición como a una decisión)
- Líneas de código
- Métodos
- Clases
- Adicionalmente calcula el valor de CC

JaCoCo usa un mecanismo para instrumentar las clases denominado "Java Agent", y que será necesario tener en cuenta en el pom.

JaCoCo permite realizar en análisis a nivel de pruebas unitarias y de pruebas de integración.

Adicionalmente permite la posibilidad de incluir en un informe de un proyecto, el informe de sus dependencias (informe agregado)

JaCoCo también incluye una goal para "chequear" los niveles de cobertura (y detener la construcción del proyecto si éste no alcanza los niveles de cobertura requeridos).