

Sesión S09:

Pruebas de aceptación (2)

Pruebas de aceptación

- Propiedades emergentes no funcionales
- Métricas
- Ejemplos de pruebas
 - ➔ Pruebas de carga
 - ➔ Pruebas de estrés
 - ➔ Pruebas estadísticas
- Pasos a seguir durante el proceso de pruebas
- Automatización de las pruebas: JMeter

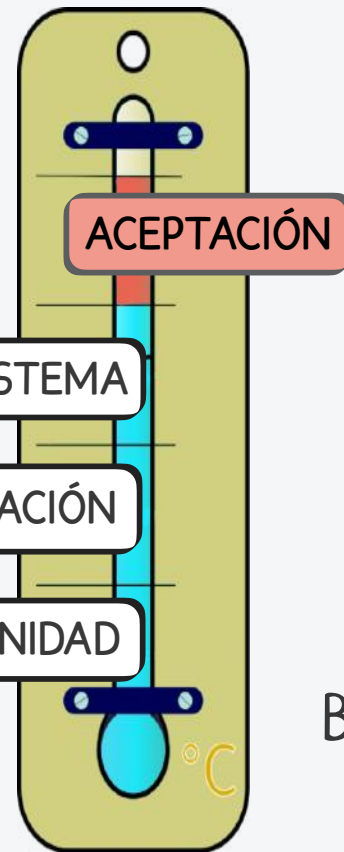
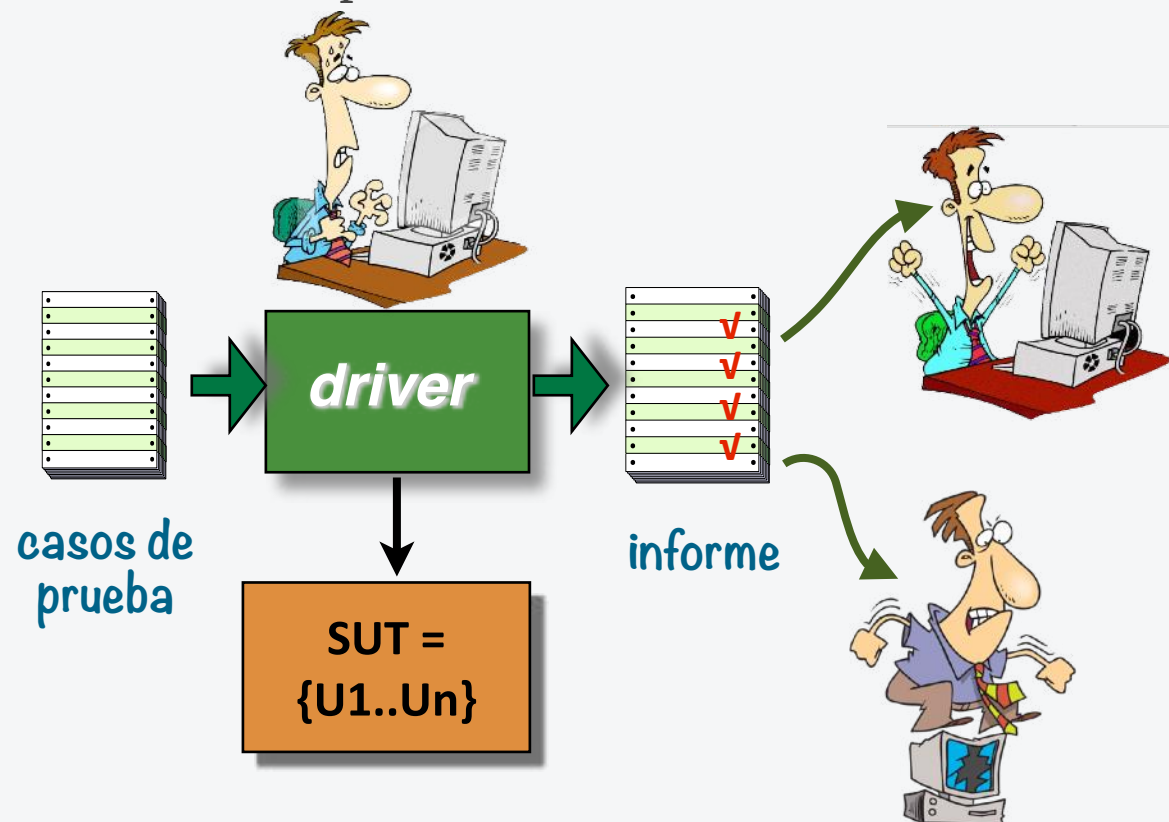
Vamos al laboratorio...



NIVELES DE PRUEBAS

VERIFICACIÓN
¿el producto está implementado correctamente?
El objetivo general (intención) es **ENCONTRAR DEFECTOS**

Cada nivel tiene una granularidad y objetivos concretos diferentes. Hay un ORDEN temporal entre ellos.



VALIDACIÓN
¿el producto es el correcto?
Las pruebas de aceptación determinan en qué **GRADO** el sistema satisface los **CRITERIOS DE ACEPTACIÓN**

Business acceptance testing

User acceptance testing (α , β)

Propiedades emergentes

FUNCIONALES
Automatización de pruebas de aplicaciones Web

Selenium WebDriver

NO funcionales

JMeter

PROPIEDADES EMERGENTES NO FUNCIONALES

Solo tiene sentido aplicarlas al sistema como un TODO

○ Hay dos tipos de propiedades emergentes:

- **Funcionales**: describen lo **que** el sistema hace, o debería hacer (does view)
- **No funcionales**: hacen referencia a "how to well a system perform its functional requirements"

○ Muchas de las propiedades emergentes NO FUNCIONALES se categorizan como "-ilidades":

- **fiabilidad**: probabilidad de funcionamiento sin fallos durante un tiempo determinado en un entorno específico
- **disponibilidad**: tiempo durante el cual el sistema proporciona servicio al usuario. Suele expresarse como: hh/dd (p.ej 24/7: 24 horas al día, 7 días por semana)
- **mantenibilidad**: capacidad de un sistema para soportar cambios. Hay tres tipos de cambios: correctivos, adaptativos y perfectivos
 - * **correctivos**: son provocados por errores detectados en la aplicación
 - * **adaptativos**: son provocados por cambios en el hardware y/o software (sistema operativo) sobre los que se ejecuta nuestra aplicación
 - * **perfectivos**: debidos a que se quiere añadir/modificar las funcionalidades existentes para ampliar/mejorar el "negocio" que sustenta nuestra aplicación
- **escalabilidad**: hace referencia a la capacidad de mantener el tiempo de respuesta ante cambios en el número de usuarios que utilizan el sistema.

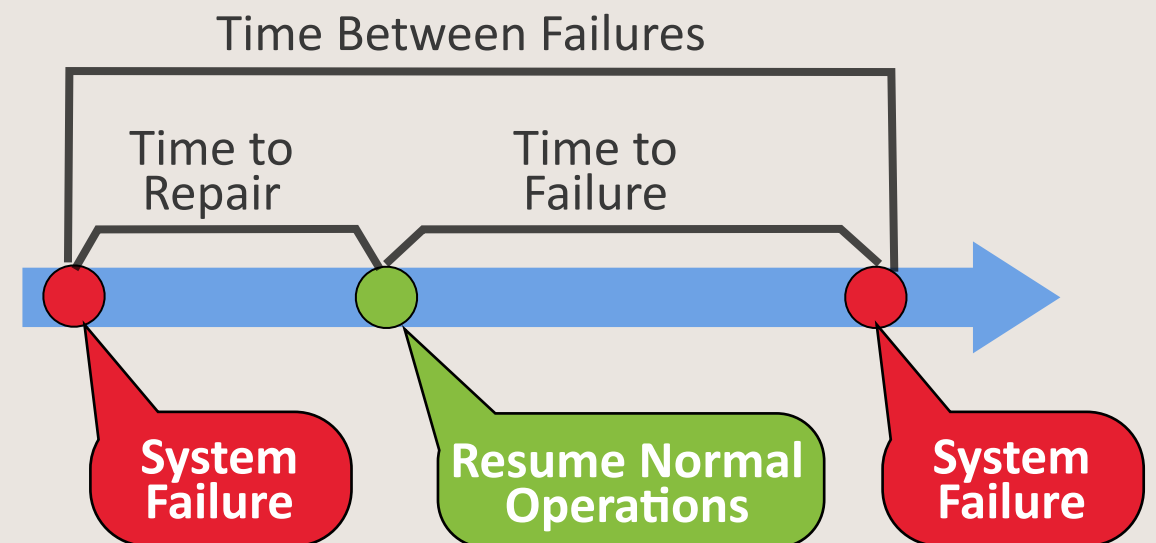
P

MÉTRICAS

Si no podemos "medir" No podremos validar!!

Las pruebas de aceptación determinan en qué **GRADO** el sistema satisface los **CRITERIOS DE ACEPTACIÓN**

- Los criterios de aceptación deben incluir propiedades emergentes "cuantificables"
 - Hay que tener mucho cuidado con criterios de aceptación ambiguos, como "Las peticiones se tienen que servir en un tiempo razonable". Dichas sentencias son imposibles de cuantificar y por lo tanto, imposibles de probar (medir) con precisión
- Para juzgar en qué grado se satisfacen los criterios de aceptación se utilizan diferentes métricas:



- Para estimar la **fiabilidad** se utilizan pruebas aleatorias basándonos en un perfil operacional. Se utilizan las métricas MTTF (Mean Time To Failure), MTTR (Mean Time To Repair), y $MTBF = MTTF + MTTR$ (MTBF: Mean time between failures)
- Para estimar la **disponibilidad** se utiliza la métrica MTTR para medir el "downtime" del sistema. La idea es incluir medidas para minimizar el MTTR
- Para estimar la **mantenibilidad** se utiliza la métrica MTTR (que refleja el tiempo consumido en analizar un defecto correctivo, diseñar la modificación, implementar el cambio, probarlo y distribuirlo)
- La **escalabilidad** del sistema utiliza el número de transacciones (operaciones) por unidad de tiempo. Los sistemas suelen poder incrementar su escalabilidad siempre y cuando no sobrepasen limitaciones de almacenamiento de datos, ancho de banda o velocidad de procesador.

<https://blog.fosketts.net/2011/07/06/defining-failure-mttr-mttf-mtbf/>

P

EJEMPLOS DE PRUEBAS (PROCEDIMIENTOS)

Dependiendo de la propiedad a validar, se usan diferentes **métodos**

P

- Las **pruebas de carga** validan el **rendimiento** de un sistema en términos de "tratar un número específico de usuarios manteniendo un ratio de transacciones" (p.ej. "una petición del sistema se debe tratar en menos de 2 segundos cuando existen 10000 usuarios dentro del sistema")
- Las **pruebas de stress** consisten en "forzar" peticiones al sistema por encima del límite del diseño del software. Por ejemplo si el sistema se ha diseñado para permitir hasta 300 transacciones por segundo, comenzaremos por hacer pruebas con una **carga** de peticiones inferior a 300 e incrementaremos gradualmente la carga hasta sobrepasar los 300 y ver cuándo falla el sistema
 - Las pruebas de stress comprueban la **fiabilidad** y **robustez** del sistema cuando se supera la carga normal (robustez= capacidad de recuperación del sistema ante entradas erróneas u otros fallos)
- Para evaluar la **fiabilidad** de un sistema podemos utilizar lo que se denominan **pruebas estadísticas**, que consisten en:
 1. construir un "perfil operacional" (operational profile), que refleje el uso real del sistema (patrón de entradas). Como resultado se identifican las "clases" de entradas y la probabilidad de ocurrencia para cada clase, asumiendo un uso "normal" (diseño de los casos de prueba)
 2. generar un conjunto de datos de prueba que reflejen dicho perfil operacional
 3. probar dichos datos midiendo el número de fallos y el tiempo entre fallos, calculando la fiabilidad del sistema después de observar un número de fallos estadísticamente significativo

P

EJEMPLO DE GENERACIÓN DE PRUEBAS (DISEÑO)

P




Clase entrada	Distrib. Probab.	Intervalo
C1	50 %	1-49
C2	15 %	50-64
C3	15 %	65-79
C4	15 %	80-94
C5	5 %	95-99

Paso 2

Se generan números aleatorios entre 1 y 99, por ejemplo:
13-94-22-24-45-56-81-19-31-69-45-9-38-21-52-84-86-97-...

Se derivan casos de prueba según su distribución de probabilidad:
C1-C4-C1-C1-C1-C2-C4-C1-C1-C3-C1-C1-C1-C1-C2-C4-C4-C5-...

 El perfil operacional es la base para el diseño de pruebas emergentes no funcionales

Paso 3

... a continuación deberíamos ejecutar las pruebas midiendo el número de fallos y el tiempo entre fallos

RESUMEN DEL PROCESO DE PRUEBAS NO FUNCIONALES

- Escalabilidad, fiabilidad, carga... son ejemplos de propiedades emergentes no funcionales. Todas ellas influyen en el **rendimiento** del sistema.
 - En general, las propiedades emergentes no funcionales determinan el **RENDIMIENTO** de nuestra aplicación
- Para evaluar el **RENDIMIENTO**, necesitamos realizar las siguientes actividades:
 1. **Identificar** los criterios de **aceptación**: identificar y cuantificar las propiedades emergentes no funcionales que determinan cuál es el rendimiento aceptable para nuestra aplicación (p.e. tiempos de respuesta, fiabilidad, utilización de recursos...)
 2. **Diseñar** los tests: deberemos conocer el patrón de uso de la aplicación (perfil operacional) para que nuestros casos de prueba estén basados en ESCENARIOS reales de nuestra aplicación
 3. **Preparar** el entorno de pruebas: es importante que el entorno de pruebas sea lo más realista posible
 4. **Automatizar** las pruebas: utilizando alguna herramienta software, "grabaremos" los escenarios de prueba, y ejecutaremos los tests
 5. **Analizaremos** los resultados y realizaremos los cambios oportunos para conseguir nuestro objetivo

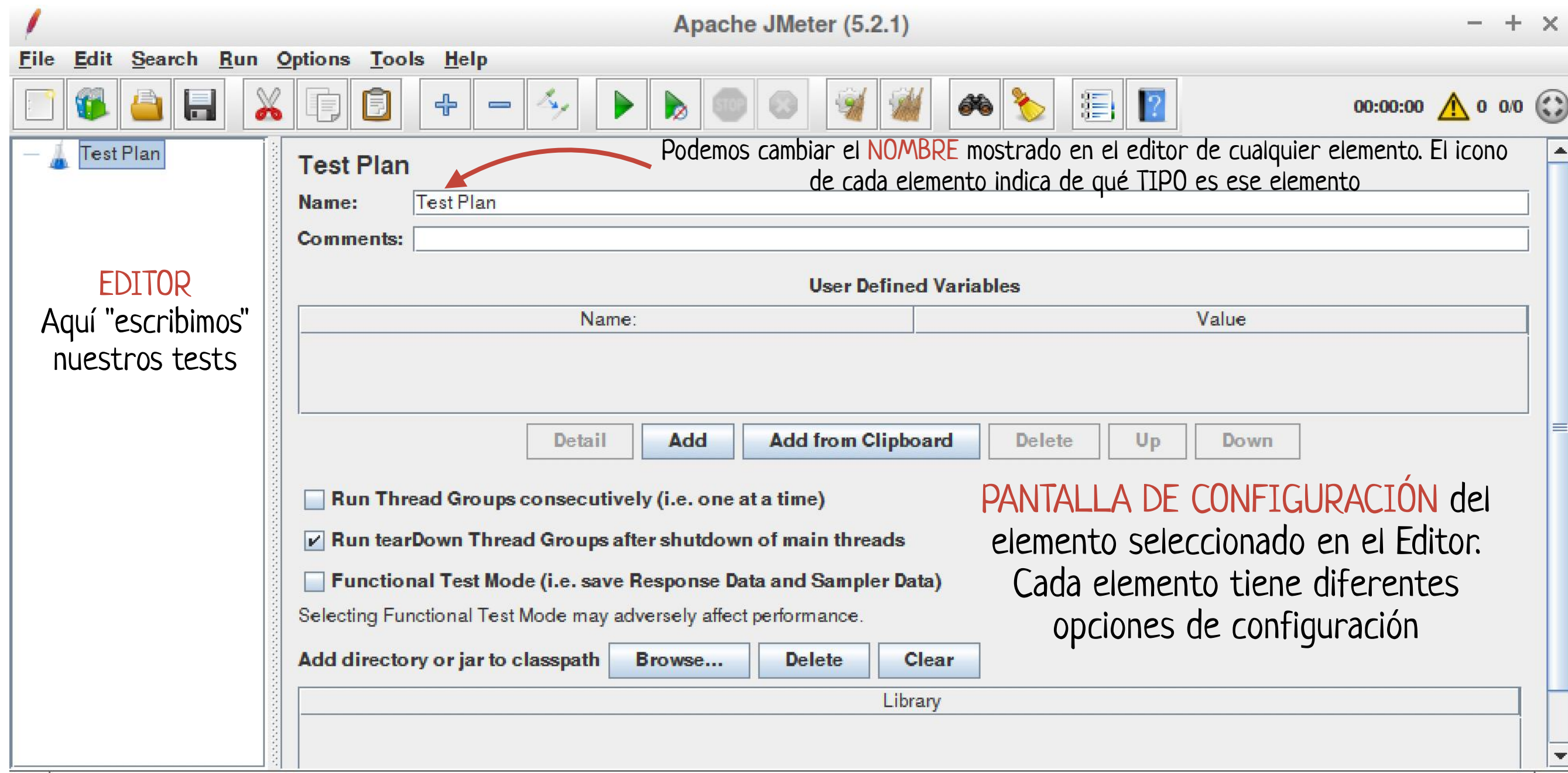


CONSIDERACIONES SOBRE EL RENDIMIENTO



- **!!! No hay que dejar las pruebas de rendimiento para el final del proyecto !!!**
- Posibles fallos relacionados con el rendimiento pueden conllevar el retocar mucho código
 - Las propiedades emergentes NO funcionales están condicionadas fundamentalmente por la ARQUITECTURA del sistema
 - Normalmente la arquitectura del sistema se determina en las primeras fases del desarrollo!! Eso significa que cambiar la arquitectura puede implicar cambiar "todo".
 - Recuerda que el coste de reparar un error siempre es proporcional al intervalo de tiempo transcurrido desde que se produjo el error, hasta que éste es detectado.
- Minimizaremos los problemas derivados de la validación de las propiedades emergentes no funcionales mediante una buena **estrategia** de pruebas combinada con una arquitectura software que considere el rendimiento desde el inicio del desarrollo
 - Si usamos una metodología de desarrollo iterativa las iteraciones iniciales del proyecto son para construir prototipos exploratorios y comprobar todos los requisitos no funcionales.

- Apache JMeter (<http://jmeter.apache.org/>) es una herramienta de escritorio 100% Java diseñada para medir el rendimiento mediante pruebas de carga
- Permite múltiples hilos de ejecución concurrente, procesando diversos y diferentes patrones de petición
 - JMeter permite trabajar con muchos tipos distintos de aplicaciones. Para cada una de ellas proporciona un **Sampler** o Muestreador, para hacer las correspondientes peticiones

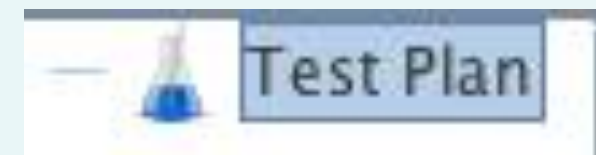


Podemos cambiar el **NOMBRE** mostrado en el editor de cualquier elemento. El icono de cada elemento indica de qué TIPO es ese elemento

EDITOR
Aquí "escribimos" nuestros tests

PANTALLA DE CONFIGURACIÓN del elemento seleccionado en el Editor. Cada elemento tiene diferentes opciones de configuración

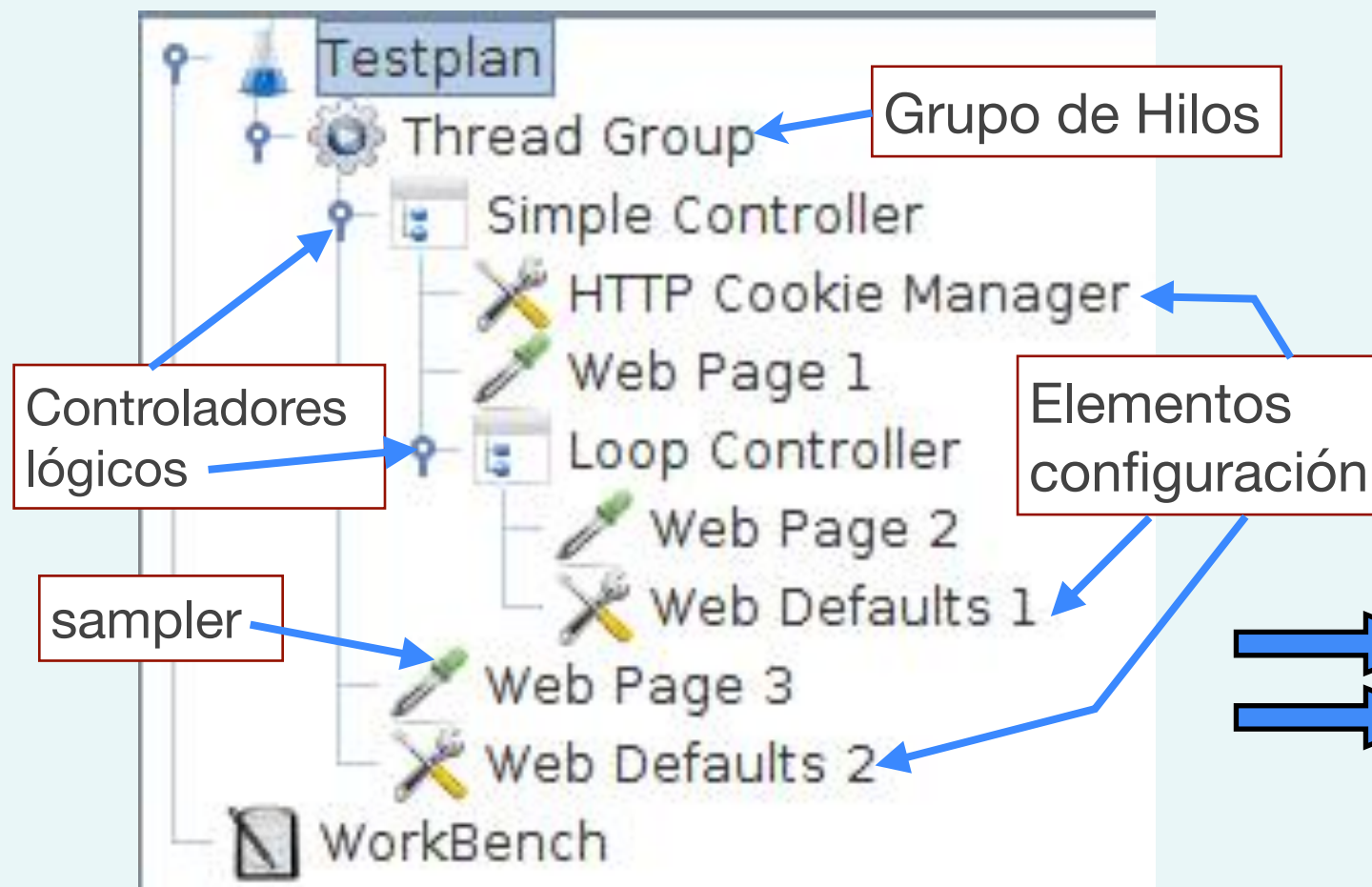
P JMeter: PLAN DE PRUEBAS (I)



https://jmeter.apache.org/usermanual/test_plan.html

- Un plan de pruebas JMeter describe una serie de "pasos" (acciones) que JMeter realizará cuando se ejecute el plan
- Un **plan de pruebas** está formado por:
 - Uno o más grupos de hilos (**Thread Groups**)
 - Controladores lógicos (**Logic Controllers**)
 - **Samplers, Listeners, Timers, Assertions, Pre-Processors, Post-Processors** y
 - Elementos de Configuración (**Configuration Elements**)

Nos centraremos en los elementos que hemos marcado con una flecha azul



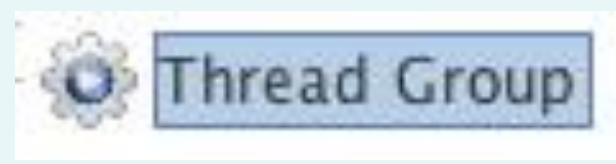
El orden de ejecución de los elementos de un plan es el siguiente:

1. Configuration elements ←
2. Pre-Processors
3. Timers ←
4. Sampler ←
5. Post-Processors (unless SampleResult is null)
6. Assertions (unless SampleResult is null) ←
7. Listeners (unless SampleResult is null) ←

P

JMETER: HILOS DE EJECUCIÓN

S



Un hilo de ejecución es el punto de partida de cualquier plan de pruebas

P

Thread Group

Name:

Comments:

Action to be taken after a Sampler error

☒ Continue ☐ Start Next Thread Loop ☐ Stop Thread ☐ Stop Test ☐ Stop Test Now

Thread Properties

Number of Threads (users):

Ramp-Up Period (in seconds):

Loop Count: ☐ Forever

☐ Delay Thread creation until needed

☒ Scheduler

Scheduler Configuration

Duration (seconds)

Startup delay (seconds)


CADA hilo ejecuta COMPLETAMENTE el plan de forma independiente de otros hilos

Podemos ejecutar el grupo de hilos un cierto número de veces o de forma indefinida (bucle "infinito")

RAMP-UP (periodo de subida): sirve para que los hilos se creen de forma gradual. Esto permite comprobar cómo rinde el servidor conforme crece la carga.

Por ejemplo, si el periodo de subida es de 100 segundos y el número de hilos es 50, significa que el servidor tardará 100 segundos en crear los 50 hilos, es decir, un nuevo hilo cada 2 segundos

Cada hilo representa a un usuario



P

JMETER: SAMPLERS



P

- Los Samplers (muestreadores) envían peticiones a un servidor. Ejemplos de samplers: HTTP request, FTP request, JDBC Request,... Se ejecutan en el orden en el que aparecen en el plan

Podemos usar cualquier **NOMBRE** pero ten en cuenta que dicho nombre será el que se mostrará en el plan. Deberíamos elegir nombres representativos de las acciones asociadas a cada elemento

HTTP Request

Name: HTTP Request

Comments:

Basic

Advanced

Web Server

Protocol [http]: Server Name or IP: Port Number:

HTTP Request

Method: GET Path: Content encoding:

☐ Redirect Automatically

☒ Follow Redirects

☒ Use KeepAlive

☐ Use multipart/form-data

☐ Browser-compatible headers

Parameters

Body Data

Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
<div><div>Detail</div><div>Add</div><div>Add from Clipboard</div><div>Delete</div><div>Up</div><div>Down</div></div>				

P

JMETER: CONTROLADORES LÓGICOS



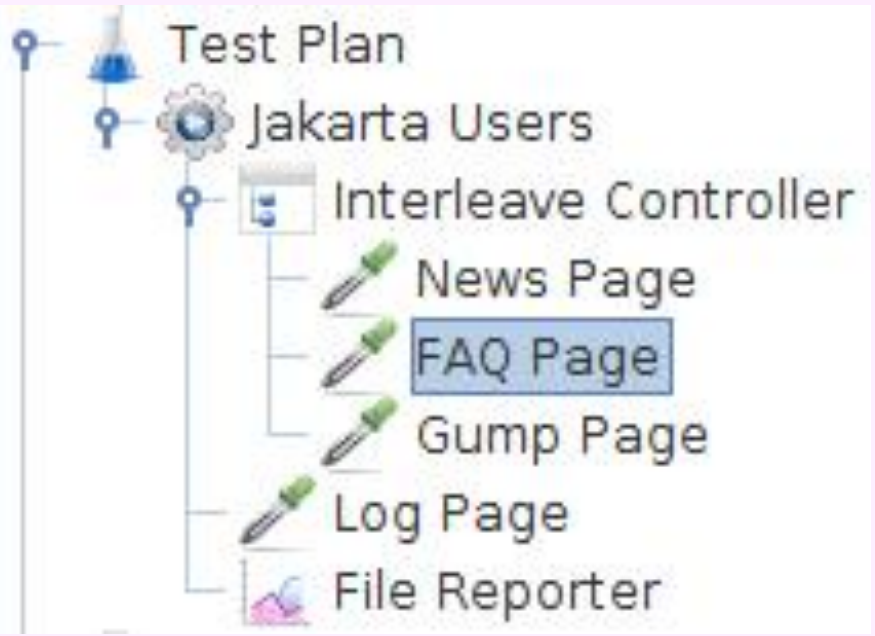
http://jmeter.apache.org/usermanual/component_reference.html#Interleave_Controller

P

- Determinan la lógica que JMeter utiliza para decidir cuándo enviar las peticiones (orquestan el flujo de control). Actúan sobre sus elementos hijo
- Ejemplos de controladores
 - **Simple controller**: No tiene efecto sobre cómo procesa JMeter los elementos hijos de dicho controlador. Simplemente sirve para "agrupar" dichos elementos.
 - **Loop controller**: Itera sobre sus elementos hijos un cierto número de veces
 - **Only once controller**: Indica a JMeter que sus elementos hijos deben ser procesados UNA ÚNICA vez en el plan de pruebas
 - **Interleave controller**: ejecutará uno de sus subcontroladores o samplers en cada iteración del bucle de pruebas, alterándose secuencialmente a lo largo de la lista

Ejemplo: supongamos que el grupo de hilos está formado por 2 hilos, y 5 iteraciones

Sesión 9: Pruebas de aceptación 2



Loop Iteration	Each JMeter Thread Sends These HTTP Requests
1	News Page
1	Log Page
2	FAQ Page
2	Log Page
3	Gump Page
3	Log Page
4	Because there are no more requests in the controller, JMeter starts over and sends the first HTTP Request, which is the News
4	Log Page
5	FAQ Page
5	Log Page

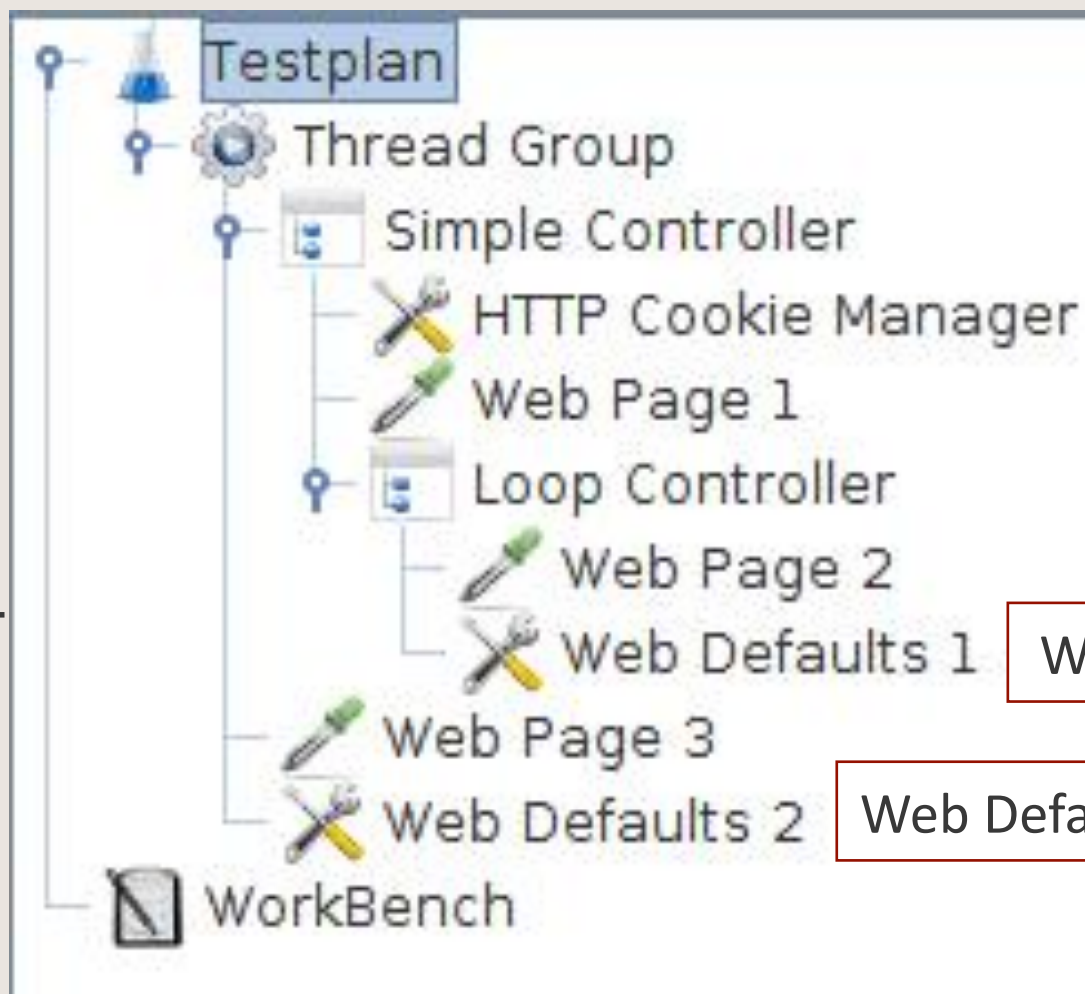
Cada hilo realiza 10 peticiones

JMETER: ELEMENTOS DE CONFIGURACIÓN



http://jmeter.apache.org/usermanual/test_plan.html

- "Trabajan" conjuntamente con un sampler. Si bien no realizan peticiones (excepto HTTP Proxy Server: es un elemento solamente disponible para el banco de trabajo), pueden modificar las mismas a través de los atributos correspondientes
- Un elemento de configuración es accesible sólo dentro de la rama del árbol (y sub-ramas) en la que se sitúa el elemento
- Un elemento de configuración dentro de una rama del árbol tiene mayor preferencia que el mismo elemento (mismo tipo de elemento) en una rama padre



Cookie Manager es accesible por Web Page 1 y Web Page 2

Web Defaults 1 sólo es accesible por Web Page 2

Web Defaults 2 es accesible por Web Page 1, Web Page 2 y Web Page 3

P

TIMERS



P

- Por defecto JMeter envía las peticiones sin realizar ninguna pausa entre las mismas. En este caso nuestro test podría "saturar" el servidor como consecuencia de enviar demasiadas peticiones en intervalos cortos de tiempo!!

- Los temporizadores permiten introducir pausas **antes** de realizar **cada** una de las peticiones de **cada** hilo



- Podemos utilizar varios tipos de temporizadores:

- ☐ **Constant timer:** Retrasa cada petición de usuario la misma cantidad de tiempo
- ☐ **Uniform random timer:** Introduce pausas aleatorias
- ☐ **Gaussian random timer:** Introduce pausas según una determinada distribución

Siempre se procesan
ANTES de
cada muestra



Uniform Random Timer

Name: Uniform Random Timer

Comments:

Thread Delay Properties

Random Delay Maximum (in milliseconds): 100.0

Constant Delay Offset (in milliseconds): 0

P

JMETER: ASERCIONES

S



- Las aserciones permiten hacer afirmaciones sobre las respuestas recibidas del servidor que se está probando. Podemos añadir aserciones a cualquier sampler. Se trata de probar que la aplicación devuelve el resultado esperado

Response Assertion

Name: Response Assertion

Comments:

Apply to:

☐ Main sample and sub-samples ☒ Main sample only ☐ Sub-samples only ☐ JMeter Variable Name to use

Field to Test

☒ Text Response ☐ Response Code ☐ Response Message ☐ Response Headers

☐ Request Headers ☐ URL Sampled ☐ Document (text) ☐ Ignore Status

☐ Request Data

Pattern Matching Rules

☐ Contains ☐ Matches ☐ Equals ☒ Substring ☐ Not ☐ Or

Patterns to Test

Patterns to Test

Add Add from Clipboard Delete

Cualquier driver SIEMPRE debe incluir aserciones!!!

JMETER: LISTENERS (I)

http://jmeter.apache.org/usermanual/component_reference.html#listeners

- Los **listeners (receptores)** se utilizan para ver y/o almacenar en el disco los resultados de las peticiones realizadas. Proporcionan acceso a la información que JMeter va acumulando sobre los casos de prueba a medida que se ejecutan los tests
 - TODOS los listeners guardan los MISMOS datos; la única diferencia es la forma en que presentan dichos datos en la pantalla
 - Ejemplos de listeners: Escritor de Datos Simple, Gráfico, Gráfico de resultados, Informe agregado, Reporte resumen, Response Time Graph, Resultados de la Aserción,...

Simple Data Writer

Name:

Comments:

Write results to file / Read from file

Filename Log/Display Only: ☐ Errors ☐ Successes

Aggregate Graph

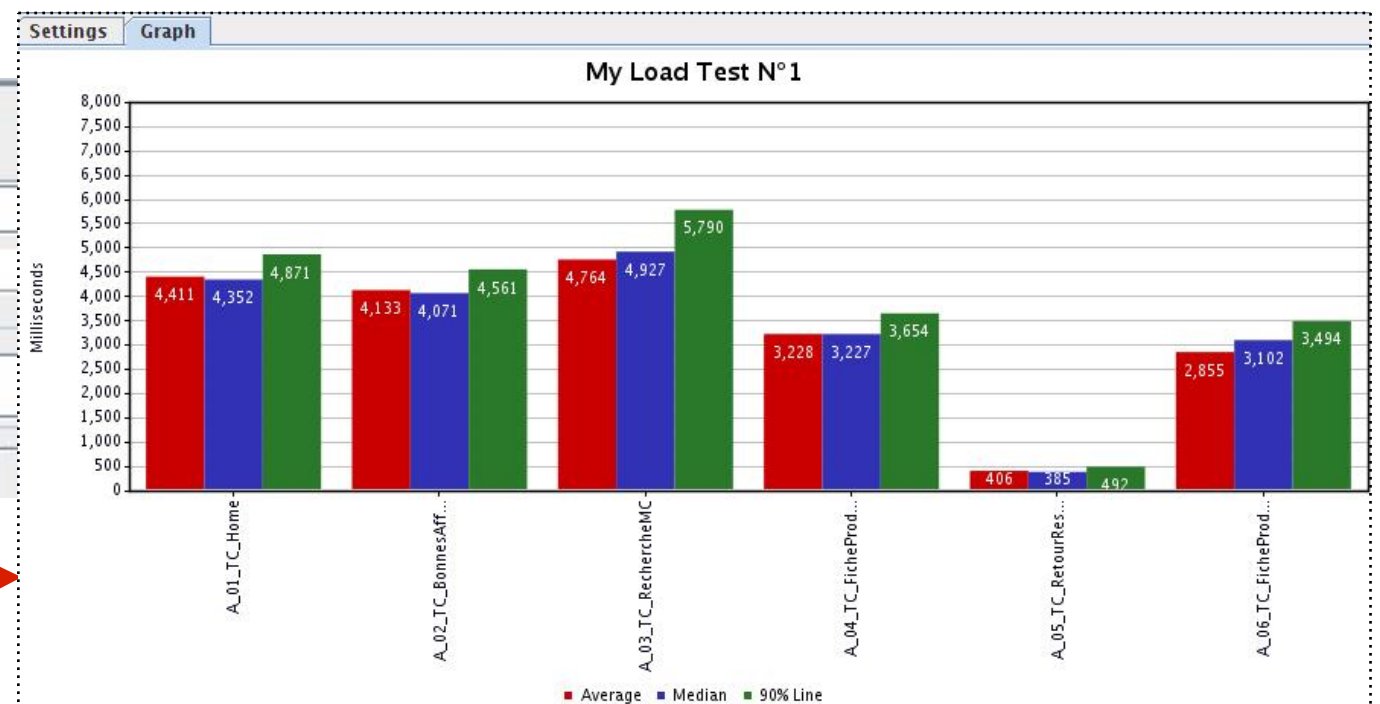
Name:

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Median	90% Line
-------	-----------	---------	--------	----------



JMETER: LISTENERS (II)

ver <http://www.guru99.com/jmeter-performance-testing.html>

Aggregate Report

Name:

Comments:

Write results to file / Read from file

Filename

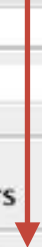
Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request-3	80	222	224	317	104	353	0.00%	1.5/sec	2.62	0.45
HTTP Request-1	80	231	233	336	108	354	0.00%	1.5/sec	10.49	0.13
HTTP Request-2	80	233	233	329	102	354	1.25%	1.5/sec	5.23	0.28
TOTAL	240	229	229	330	102	354	0.42%	4.4/sec	18.18	0.86

☐ Include group data ☒ Save Table Header

Rendimiento = num_peticiones/segundo

Rendimiento = Throughput



Graph Results

Name:

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☒ Data ☒ Average ☒ Median ☒ Deviation ☒ Throughput

No of Samples 832 Latest Sample 203 Average 298
Deviation 216 Throughput 149,115/minute Median 204

El **Throughput** representa la capacidad del servidor para soportar una determinada carga. cuanto más alto sea, mayor será el rendimiento del servidor!!!



JMETER: LISTENERS (III)

Response Time Graph

Name: Response Time Graph

Comments:

Write results to file / Read from file

Filename /tmp/results.csv

Browse...

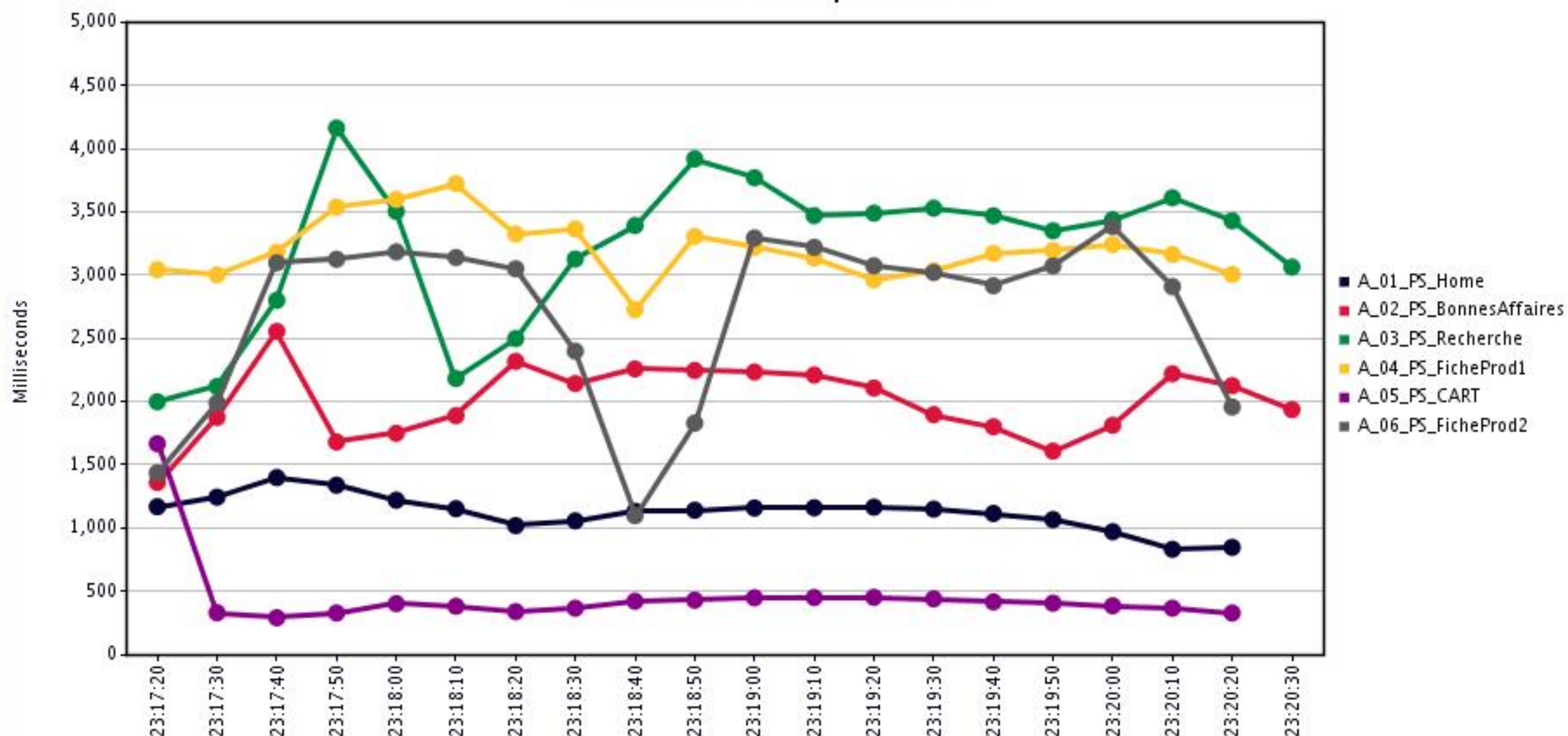
Log/Display Only: ☐ Errors ☐ Successes

Configure

Settings

Graph

Load test N° 1: response time





SOBRE LOS DATOS REGISTRADOS POR JMeter

El tiempo se calcula en milisegundos



○ Para cada muestra (sampler), JMeter calcula:

- ❑ # Muestras – Número de muestras con la misma etiqueta
- ❑ Media – Tiempo medio de respuesta (en milisegundos)
- ❑ Mediana – The median is the time in the middle of a set of results. 50% of the samples took no more than this time; the remainder took at least as long.
- ❑ Línea de 90% (percentil): 90% of the samples took no more than this time. The remaining samples took at least as long as this
- ❑ Min – Tiempo mínimo de respuesta para las muestras con la misma etiqueta
- ❑ Max – Tiempo máximo de respuesta para las muestras con la misma etiqueta
- ❑ % Error – Porcentaje de peticiones con errores
- ❑ Rendimiento (Throughput) – número de peticiones por segundo/ minuto/hora. La unidad de tiempo se elige en función de que el valor visualizado sea como mínimo 1.
- ❑ Kb/sec – rendimiento expresado en Kilobytes por segundo

CONSEJOS JMeter



- Utiliza escenarios de prueba significativos, y construye planes de prueba que prueben situaciones representativas del mundo real
 - Los casos de uso ofrecen un punto de partida ideal. A partir de ellos debes generar un perfil operacional
- Asegúrate de ejecutar JMeter en una máquina distinta a la del sistema a probar.
 - Esto previene que JMeter afecte sobre los resultados de las pruebas
- El proceso de pruebas es un proceso científico. Todas las pruebas se deben realizar bajo condiciones completamente controladas
 - Si estas trabajando con un servidor compartido, primero comprueba que nadie más esta realizando pruebas de carga contra la misma aplicación web.
- Asegúrate de que dispones de ancho de banda en la estación que ejecuta JMeter
 - La idea es probar el rendimiento de la aplicación y el servidor, y no la conexión de la red.
- Utiliza diferentes instancias de JMeter ejecutándose en diferentes máquinas para añadir carga adicional al servidor
 - Esta configuración suele ser necesaria para realizar pruebas de stress. JMeter puede controlar las instancias JMeter de las otras máquinas y coordinar la prueba
- Deja una prueba JMeter ejecutarse durante largos periodos de tiempo, posiblemente varios días o semanas
 - Estarás probando la disponibilidad del sistema y resaltando las posibles degradaciones en el rendimiento del servidor debido a una mala gestión de los recursos



Y AHORA VAMOS AL LABORATORIO...

Vamos a implementar tests de aceptación (para validar propiedades emergentes NO funcionales) sobre una aplicación web con JMeter

Camino	Datos Entrada	Resultado Esperado
C1	d1=... d2=... ...	r1
..		
CM	d1=... d2=... ...	rM

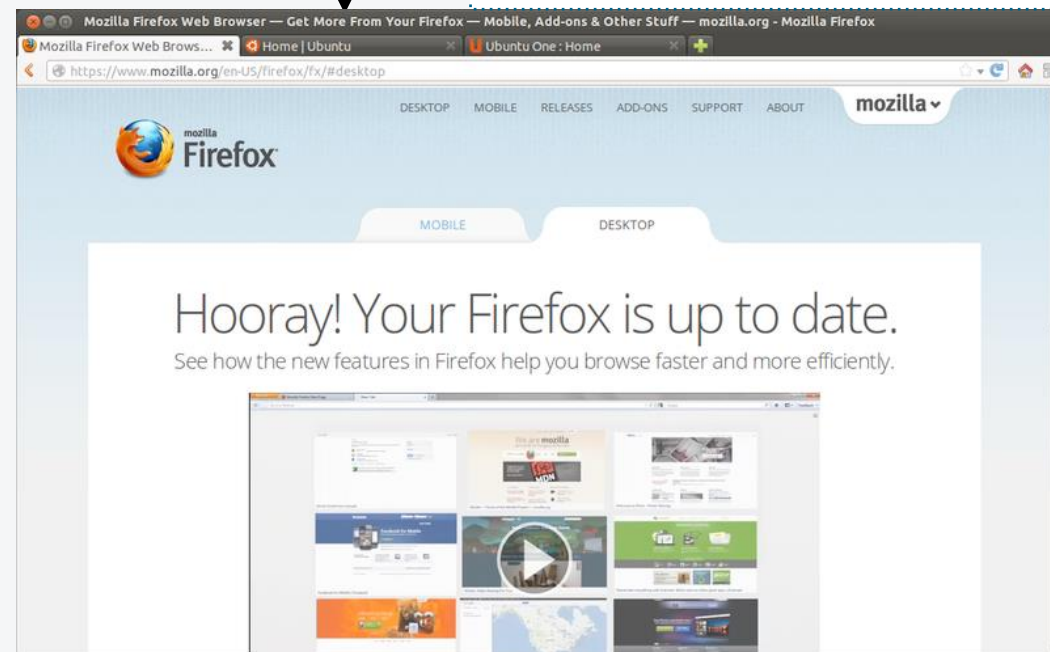
usaremos JMeter (el driver estará implementado en un fichero .jmx)

Tests aceptación

driver

Informe

Navegador Firefox



Sevidor web

SUT



REFERENCIAS BIBLIOGRÁFICAS



- Software testing and quality assurance. Kshirasagar Naik & Priyadarshi Tripathy. Wiley. 2008
 - Capítulos 8 y 15
- Página oficial JMeter:
 - <http://jmeter.apache.org/usermanual/index.html>
- Otras referencias interesantes:
 - <http://www.guru99.com/jmeter-performance-testing.html>
 - <http://www.wikishown.com/apache-jmeter-understanding-summary-report/>
 - <http://jmeterperftest.blogspot.com.es>
 - https://www.blazemeter.com/blog/understanding-your-reports-part-3-key-statistics-performance-testers-need-understand?utm_source=Blog&utm_medium=BM_Blog&utm_campaign=kpis-part1