



Sesión S10: Pruebas de aceptación (3)

Propiedades emergentes no funcionales

Métricas utilizadas

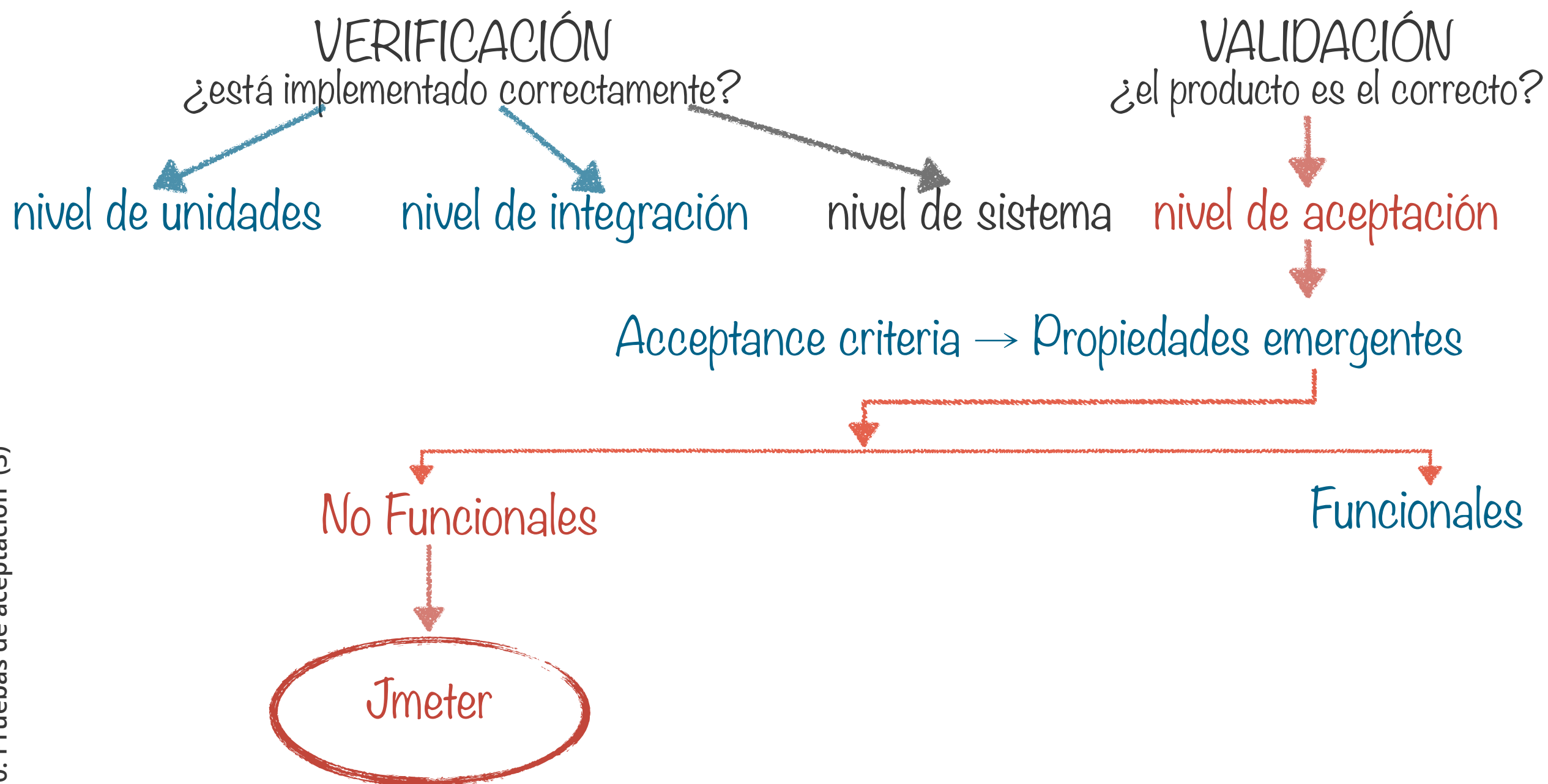
Ejemplos de pruebas

Pasos a seguir durante el proceso de pruebas

Algunos tipos de pruebas

Automatización de las pruebas: JMeter

- Las pruebas se realizan a diferentes niveles, durante el proceso de desarrollo del software





PROPIEDADES EMERGENTES NO FUNCIONALES



- 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.



ALGUNAS MÉTRICAS UTILIZADAS



- 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.



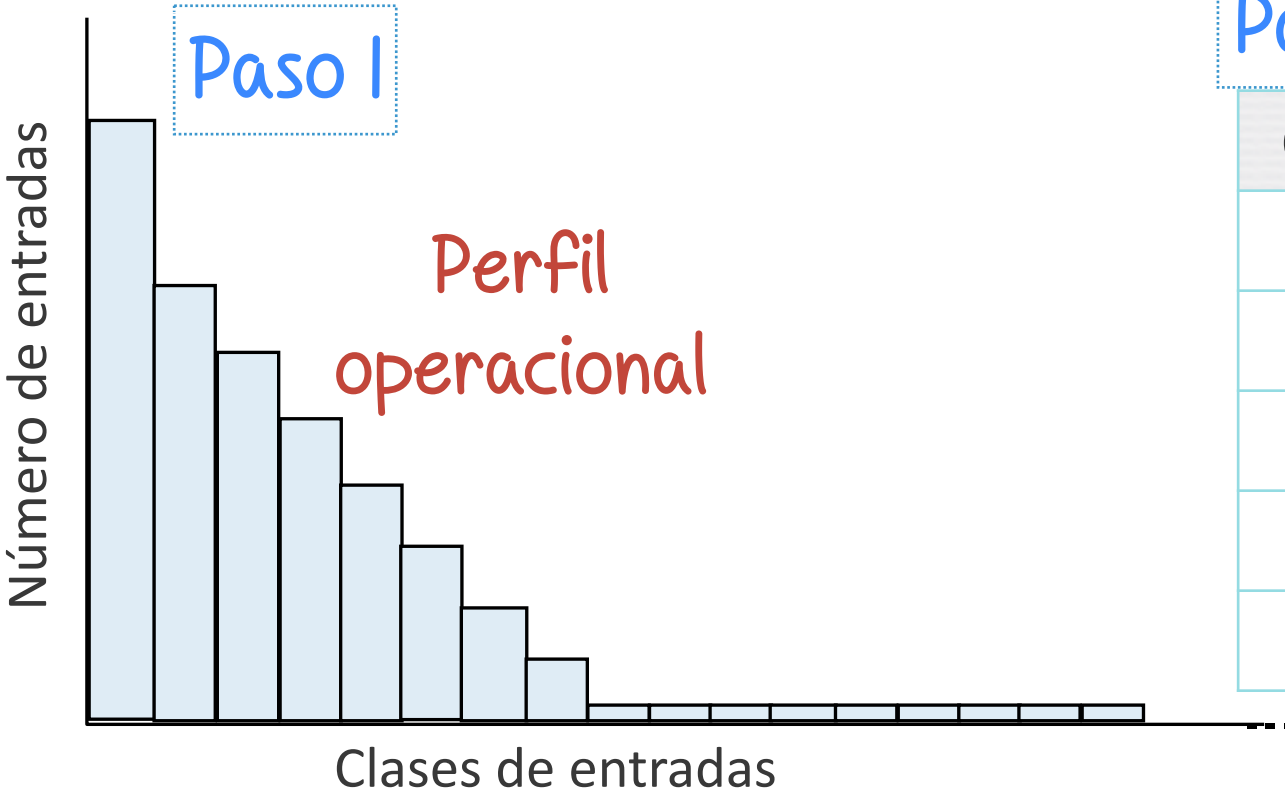
EJEMPLOS DE PRUEBAS (PROCEDIMIENTOS)



- 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



EJEMPLO DE GENERACIÓN DE PRUEBAS (DISEÑO)




Paso 2

Clase entrada	Distrib. Probab.	Intervalo
C1	50 %	1-49
C2	15 %	50-63
C3	15 %	64-78
C4	15 %	79-94
C5	5 %	95-99

Paso 3 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-...

Paso 4 Se derivan casos de prueba según su distribución de probabilidad:
C1-C4-C1-C1-C1-C2-C4-C1-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

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



RESUMEN DEL PROCESO DE PRUEBAS



- *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



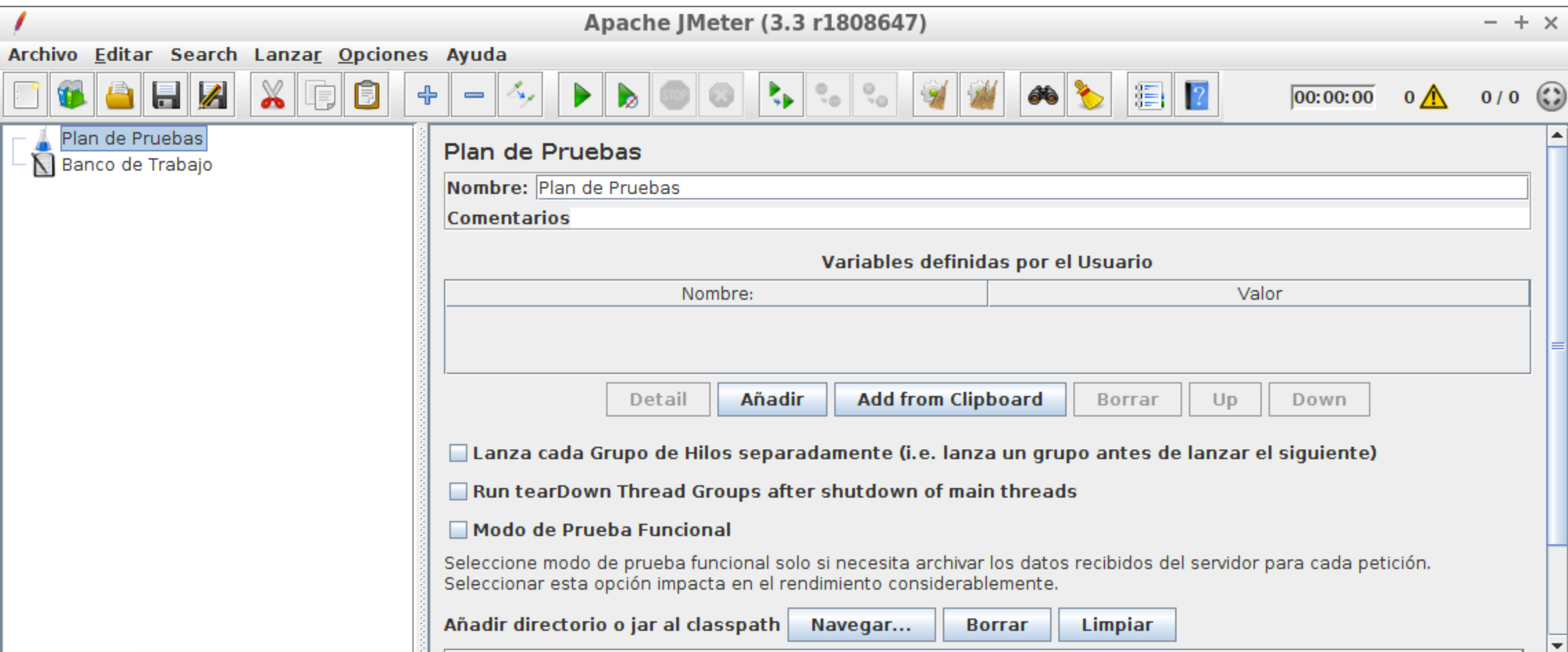
CONSIDERACIONES SOBRE EL RENDIMIENTO



- ❶ No hay que dejar las pruebas de rendimiento para el final del proyecto !!!
- ❷ 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
- ❸ Minimizaremos los problemas derivados de la validación de dichas 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.

Ver <http://jmeter.apache.org/usermanual/index.html>

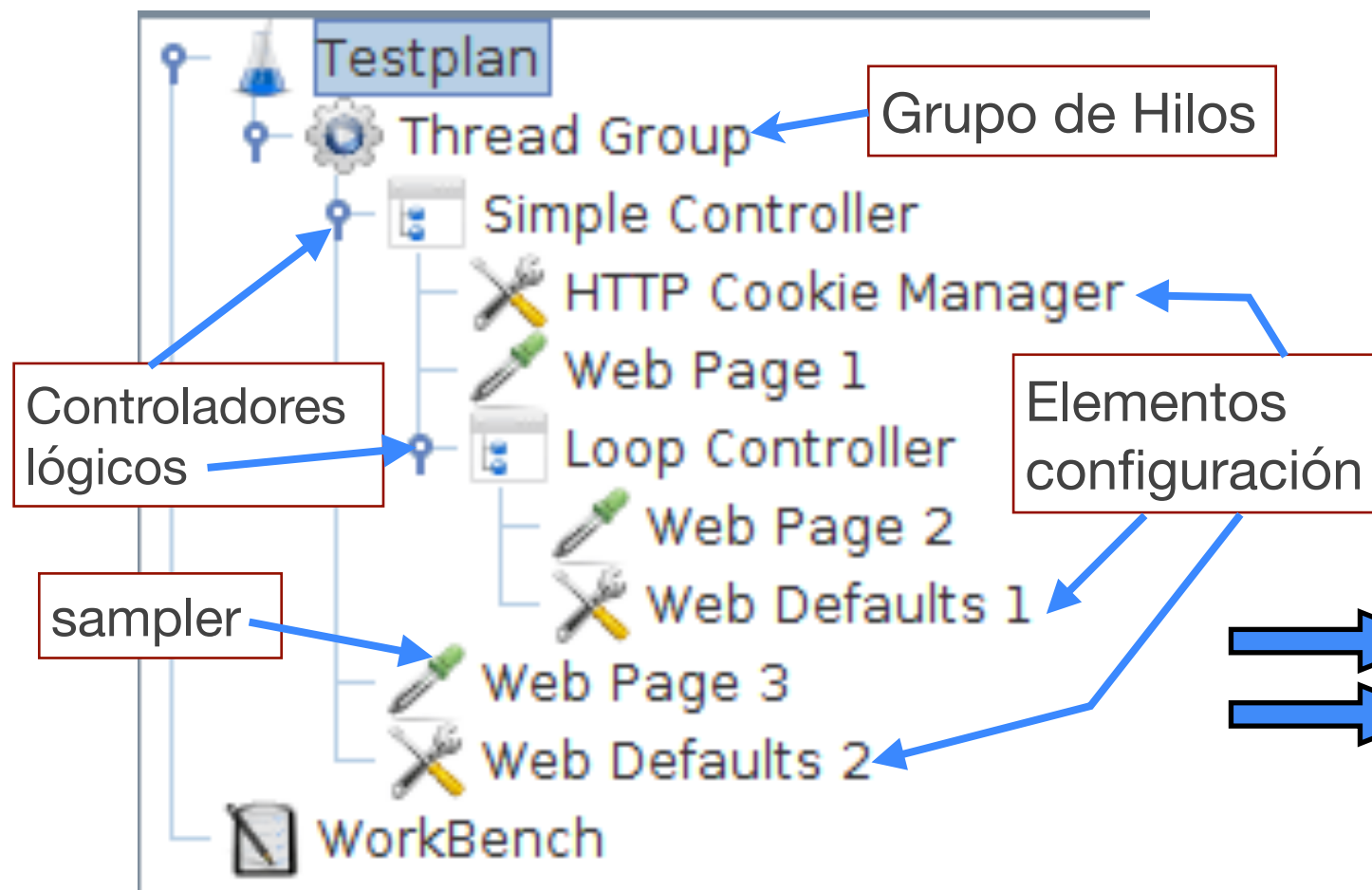
- Apache JMeter (<http://jmeter.apache.org/>) es una herramienta de escritorio 100% Java diseñada para medir el rendimiento mediante pruebas de carga
- Permite múltiples hilos de ejecución concurrente, procesando diversos y diferentes patrones de petición
 - JMeter permite trabajar con muchos tipos distintos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



JMETER: PLAN DE PRUEBAS (I)

- 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)

JMETER: HILOS DE EJECUCIÓN



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

Grupo de Hilos

Nombre: Grupo de Hilos

Comentarios

Acción a tomar después de un error de Muestreador

☒ Continuar ☐ Comenzar siguiente iteración ☐ Parar Hilo ☐ Parar Test ☐ Parar test ahora

Propiedades de Hilo

Número de Hilos 1

Periodo de Subida (en segundos): 1

Contador del bucle: ☐ Sin fin 1

☐ Delay Thread creation until needed

☒ Planificador

Cada hilo ejecuta **COMPLETAMENTE** el plan de forma independiente de otros hilos

Configuración del Planificador

Tiempo de Arranque 2015/04/18 18:59:31

Tiempo de Finalización 2015/04/18 18:59:31

Duración (segundos)

Retardo de arranque (segundos)

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.

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



Grupo de Hilos

Cada hilo representa a un usuario



- 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 árbol

Petición HTTP

Nombre: Petición HTTP

Comentarios: Muestreador o sampler

Basic **Advanced**

Servidor Web

Nombre de Servidor o IP:

Puerto:

Timeout (milisegundos)

Conexión:

Petición HTTP

Implementación HTTP:

Protocolo:

Método: GET

Codificación del

Ruta:

☐ Redirigir Automáticamente ☒ Seguir Redirecciones ☒ Utilizar KeepAlive ☐ Usar 'multipart/form-data' para HTTP POST ☐ Cabeceras compatibles

Parameters **Body Data** **Files Upload**

Enviar Parámetros Con la Petición:

Nombre:

Valor

Detail

Añadir

Add from Clipboard

Borrar

Up

Down

Servidor Proxy

Nombre de Servidor o IP:

Puerto:

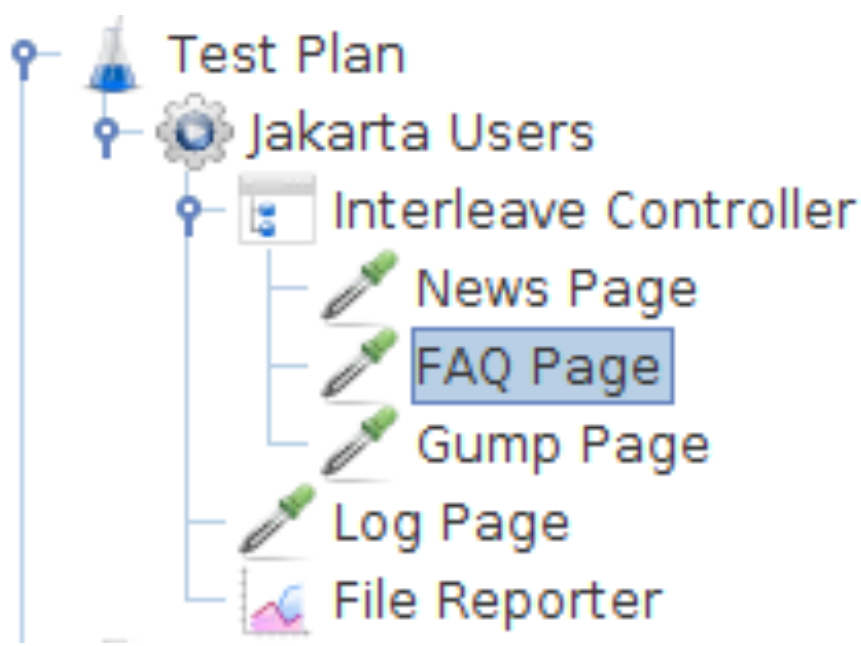
Nombre de Usuario



- Determinan la lógica que JMeter utiliza para decidir cuándo enviar las peticiones (orquestran el flujo de control). Actúan sobre sus elementos hijo
- Ejemplos de controladores
 - ❑ **Simple controller**: No tiene efecto sobre cómo procesa JMeter los elementos hijos del 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 10: Pruebas de aceptación (3)



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

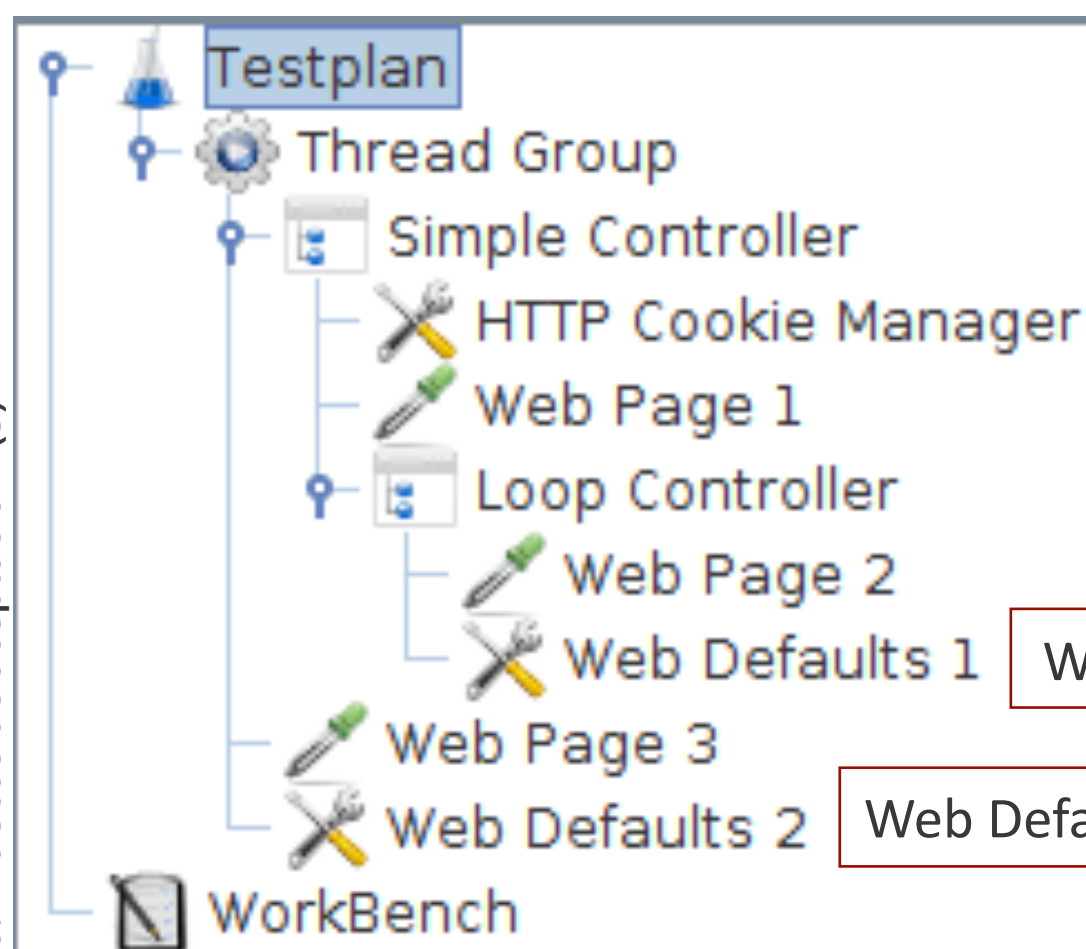


JMETER: ELEMENTOS DE CONFIGURACIÓN



Elemento de configuración

- "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

- 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
- 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



Temporizador Aleatorio Uniforme

Nombre:

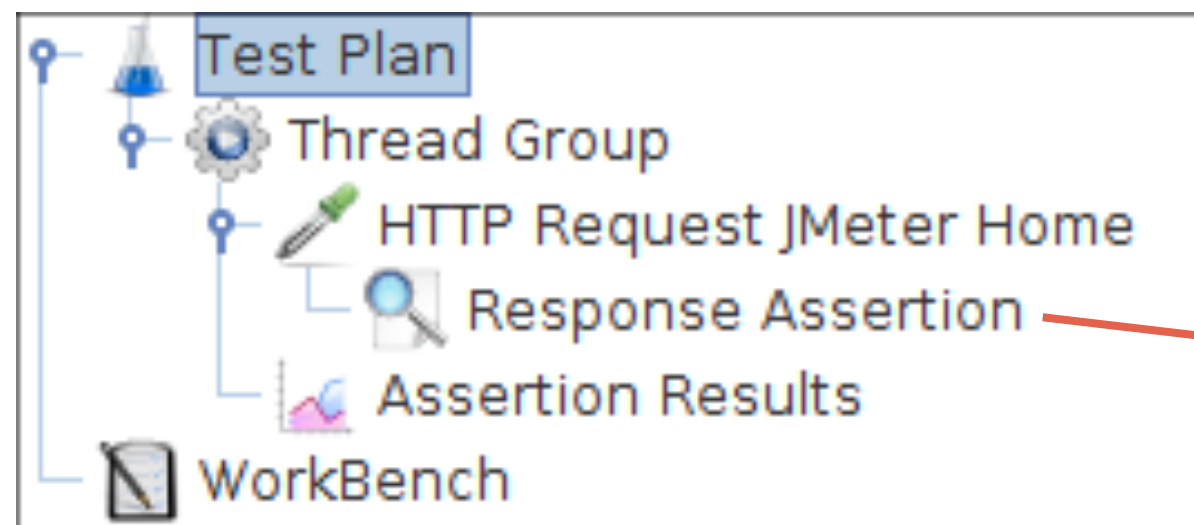
Comentarios

Propiedades de Retardo de Hilos

Máximo retardo Aleatorio (en milisegundos)

Desplazamiento de Retraso Constante (en milisegundos):

- 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 only ☐ Sub-samples only ☐ Main sample and sub-samples ☐ JMeter Variable

Response Field to Test

☒ Text Response ☐ Document (text) ☐ URL Sampled ☐ Response Code ☐ Response Message ☐ Response Headers ☐ Ignore Status

Pattern Matching Rules

☒ Contains ☐ Matches ☐ Equals ☐ Substring ☐ Not

Patterns to Test

</html>

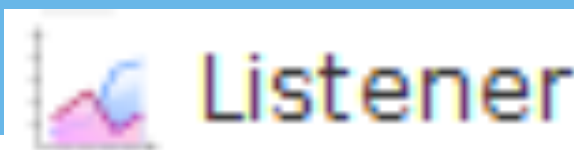
Patterns to Test

Cualquier driver
SIEMPRE debe
incluir aserciones!!!



Add

Delete



- 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,...

Escritor de Datos Simple

Nombre: Escritor de Datos Simple

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo

Navegar...

Log/Mostrar sólo:

☐ Escribir en Log Sólo Errores

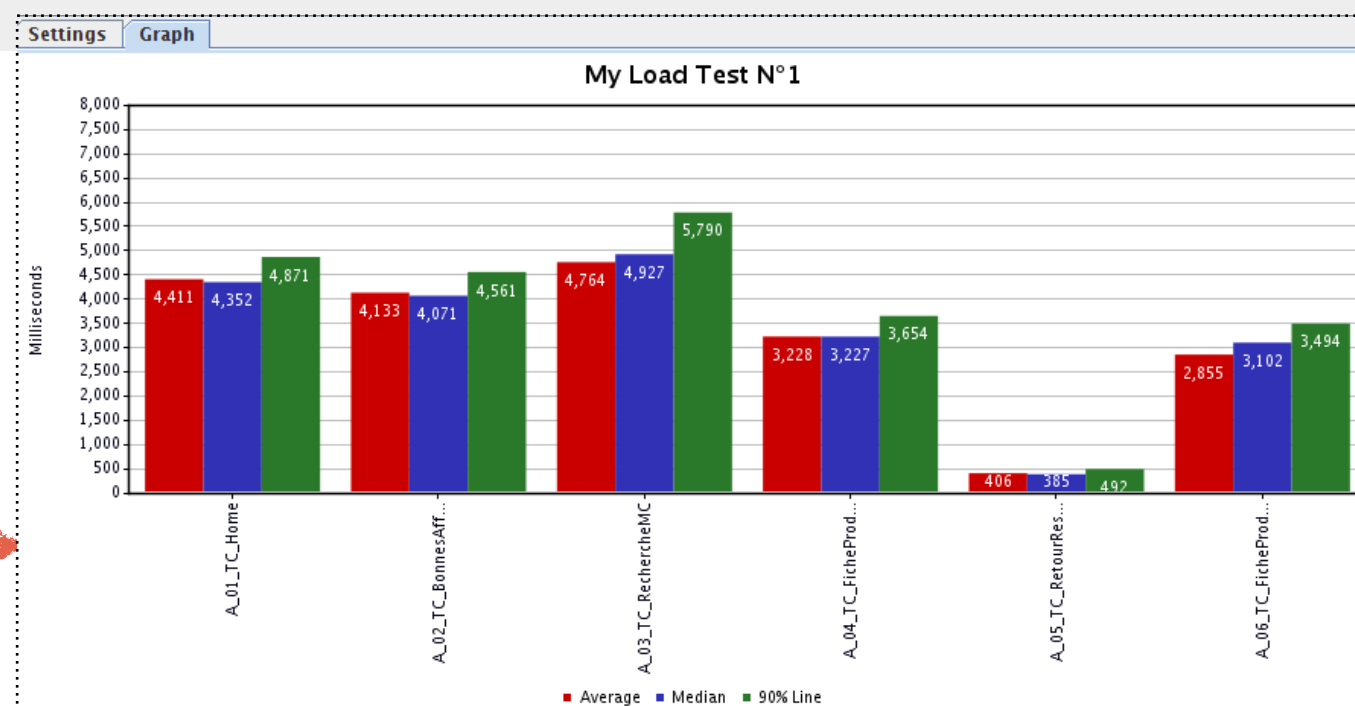
☐ Éxitos

Configurar

Gráfico

Nombre: Gráfico agregado

Comentarios



JMETER: LISTENERS (II)

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

Más ejemplos de *listeners*

Rendimiento = num_peticiones/segundo
Rendimiento = Throughput

Informe Agregado

Nombre: Informe Agregado

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo Navegar... Log/Display Only: ☐ Escribir en Log Sólo Errores ☐ Successes Configurar

Label	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
Petición HTTP Login	211	4	4	6	2	31	0,00%	61,9/sec	75,3
Petición HTTP Bibliotecario Registrado	211	6	6	9	2	33	100,00%	62,1/sec	95,8
TOTAL	422	5	6	8	2	33	50,00%	123,1/sec	169,9

Gráfico de Resultados

Nombre: Gráfico de Resultados

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo Navegar... Log/Display Only: ☐ Escribir en Log Sólo Errores ☐ Successes Configurar

Gráficos a Mostrar ☒ Datos ☒ Media ☒ Mediana ☒ Desviación ☒ Rendimiento

No. de Muestras 422 Última Muestra 15 Media 5 Desviación 2 Rendimiento 7.386,231/minuto Mediana 6

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...

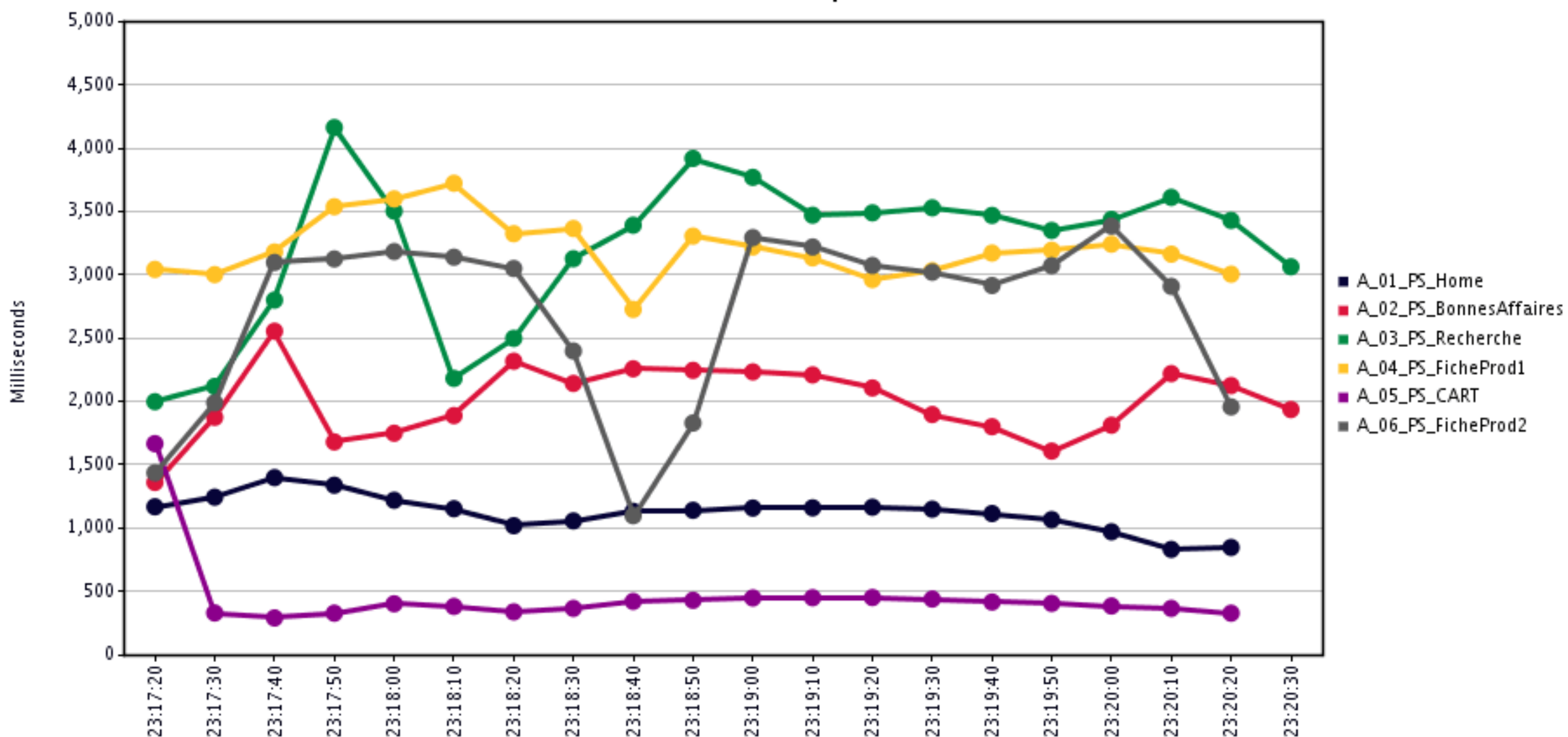
Log/Display Only: ☐ Errors ☐ Successes

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



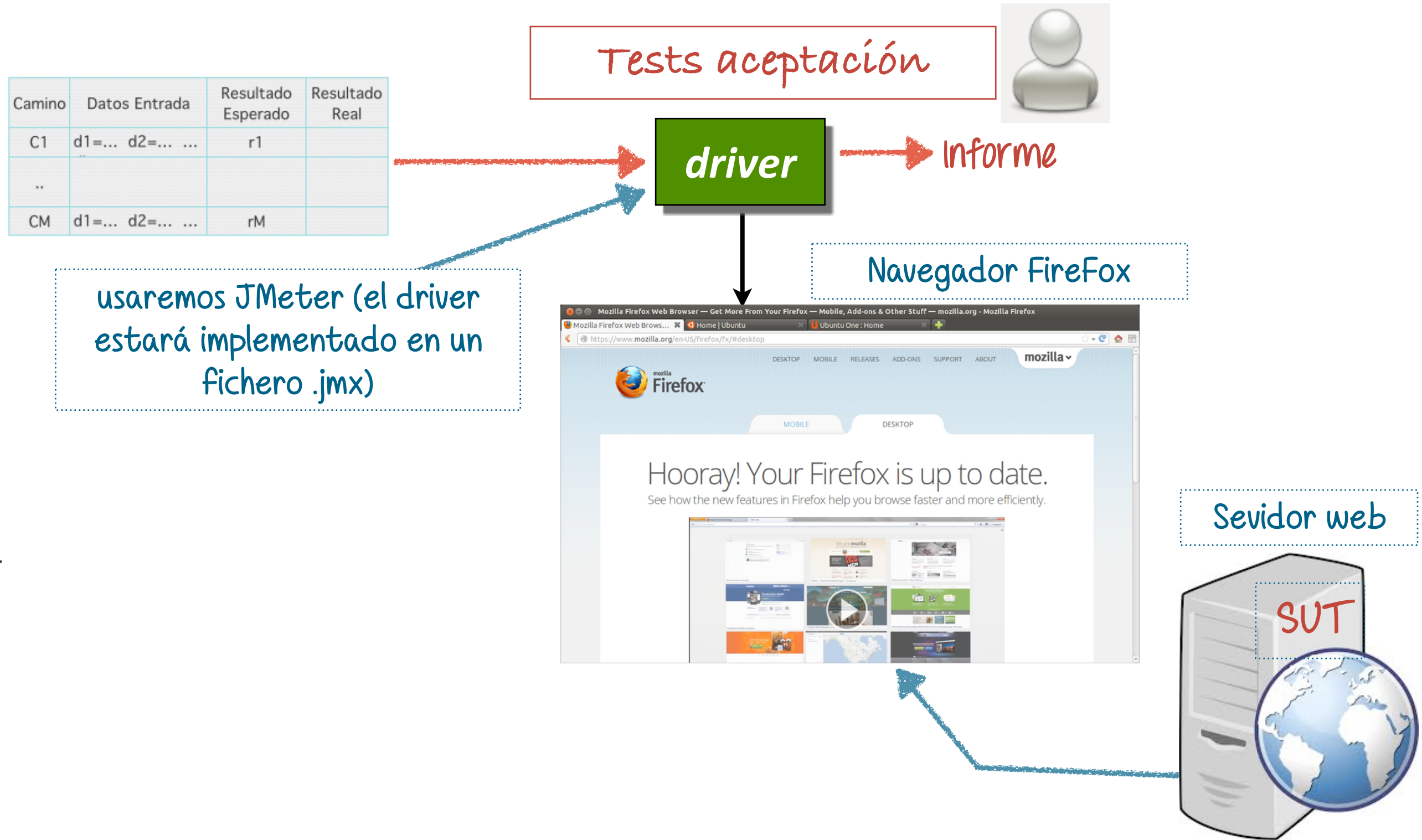
- Utiliza escenarios de prueba significativos, y construye planes de prueba que prueben situaciones representativas del mundo real
 - Los casos de uso ofrecen un punto de partida ideal. A partir de ellos debes generar un perfil operacional
- Asegúrate de ejecutar JMeter en una máquina distinta a la del 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





- 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