



# Pruebas de Software



# ÍNDICE

	Página
Presentación	5
Red de contenidos	6
<b>Unidad de aprendizaje 1: Fundamentos de Pruebas de Software</b>	
1.1 Tema 1 : Pruebas de Software	8
1.1.1. : Validación y Verificación en el desarrollo de software	8
1.1.2. : Tipos de pruebas	11
1.1.3. : Diseño de casos de prueba	17
1.2 Tema 2 : Administración de Pruebas	25
1.2.1. : Estrategias de pruebas	25
1.2.2. : Roles y responsabilidades	30
1.2.3. : Técnicas de pruebas	33
1.2.4. : Herramientas de pruebas	42
<b>Unidad de aprendizaje 2: Fundamentos Rational Functional Tester</b>	
2.1 Tema 3 : Introducción al Rational Functional Tester	54
2.1.1. : Arquitectura de Rational Functional Tester	54
2.1.2. : Configuración del entorno de pruebas	56
2.1.3. : Configuración de aplicaciones Java a probar	62
2.1.4. : Proyectos de pruebas funcionales en Rational Functional Tester	67
2.2 Tema 4 : Script de pruebas funcionales	70
2.2.1. : Grabación de un script	71
2.2.2. : Reproducción de un script	89
2.2.3. : Revisión de los resultados	90
2.2.4. : Características avanzadas de script de pruebas	90
<b>Unidad de aprendizaje 3: Fundamentos Rational Performance Tester</b>	
3.1 Tema 5 : Introducción al Rational Performance Tester	98
3.1.1. : Arquitectura de Rational Performance Tester	98
3.1.2. : Características y beneficios	99

3.2 Tema 6 : Pruebas de rendimiento	105
3.2.1. : Crear y ejecutar pruebas de rendimiento	
3.2.2. : Análisis de resultados	
4.1.3. : Modificar pruebas de rendimiento	

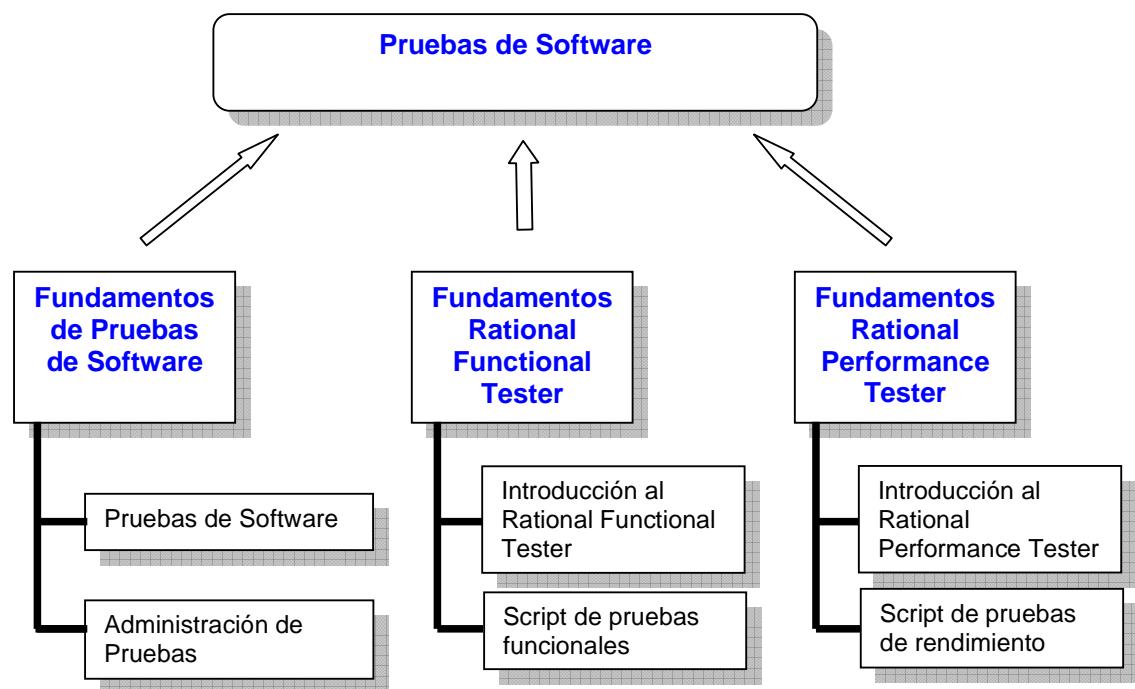
## PRESENTACIÓN

**Pruebas de Software** pertenece a la línea de carrera y se dicta en la carrera profesional de Computación e Informática. Brinda los conceptos básicos relacionados al área de aseguramiento de calidad de software y administración de pruebas de software, alineados a las mejores prácticas en desarrollo de software.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros, que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es eminentemente práctico: consiste en sesiones teóricas acompañadas con aplicaciones prácticas. En primer lugar, se explica la importancia de la verificación y validación de software para el control de calidad del producto de software. Continúa con la presentación de los fundamentos del Rational Functional Tester para la creación de scripts de pruebas funcionales. Por último, se concluye con la aplicación del Rational Performance Tester para el diseño de pruebas de rendimiento.

## RED DE CONTENIDOS





# FUNDAMENTOS DE PRUEBAS DE SOFTWARE

## LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno reconoce la importancia de la validación y verificación de software para el control de calidad del producto de software.

## TEMARIO

### 1.1. Tema 1: Pruebas de software

- 1.1.1. Validación y Verificación en el desarrollo de software
- 1.1.2. Tipos de pruebas
  - 1.1.2.1. En función de qué conocemos
  - 1.1.2.2. Según el grado de automatización
  - 1.1.2.3. En función de qué se prueba
- 1.1.3. Diseño de casos de prueba

### 1.2. Tema 2: Administración de pruebas

- 1.2.1. Estrategias de pruebas
- 1.2.2. Roles y responsabilidades
- 1.2.3. Técnicas de pruebas
- 1.2.4. Herramientas de pruebas

## ACTIVIDADES PROPUESTAS

- Los alumnos diseñan los casos de pruebas de un caso de uso a partir de su especificación.
- Los alumnos diseñan los casos de pruebas de un caso de uso a partir de su prototipo y consideraciones del llenado de datos.

## 1.1. PRUEBAS DE SOFTWARE

Las pruebas de software (*testing* en inglés) son los **procesos que permiten verificar y revelar la calidad de un producto software antes de su puesta en marcha**. Básicamente, es una fase en el desarrollo de software que consiste en probar las aplicaciones construidas.

Las pruebas de software se integran dentro de las diferentes fases del ciclo de vida del software dentro de la Ingeniería de software. En este sentido, se ejecuta el aplicativo a probar y mediante técnicas experimentales se trata de descubrir qué errores tiene.

Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema.

Existen muchas definiciones de pruebas de software. A continuación, se hace referencia a la definición citada por IEEE y SWEBOK.

Una prueba es una actividad en la que un sistema o un componente es ejecutado bajo condiciones especificadas, los resultados son observados o registrados, y una evaluación es realizada de un aspecto del sistema o componente. [IEEE Std.610.12-1990]

Una prueba es una actividad ejecutada para evaluar y mejorar la calidad del producto a través de la identificación de defectos y problemas. [SWEBOK]

Otros especialistas de pruebas manifiestan que las pruebas de software son actividades claves para que los procesos de validación y verificación tengan éxito, ya que ayudan a entregar el producto con la calidad suficiente para satisfacer las necesidades del cliente y con la certeza de que el producto cumple las especificaciones definidas. En este sentido, las pruebas pueden considerarse como un proceso que intenta proporcionar confianza en el software y cuyo objetivo fundamental es demostrar al desarrollador y al cliente que el software satisface sus requisitos.

Algo que los especialistas de pruebas deben considerar es que las pruebas de software nunca se deben realizar en un entorno de producción. Es necesario probar los nuevos sistemas en un entorno de pruebas separado físicamente del de producción. Para crear un entorno de pruebas en una máquina independiente de la máquina de producción, es necesario crear las mismas condiciones que existe en la de producción.

Como parte del proceso de validación y verificación, se debería tomar decisiones sobre quién debería ser responsable de las diferentes etapas de las pruebas. Dichas etapas de pruebas se integran dentro de las diferentes fases del ciclo del software dentro de la Ingeniería de Software.

En la figura 1.1 se observa un modelo de cómo las etapas de pruebas se integran en el ciclo de vida de desarrollo de software genérico. Durante la etapa de planificación es importante establecer una buena estrategia de pruebas y seleccionar las técnicas adecuadas de estimación en función de los factores que afecten a las pruebas del proyecto. La siguiente fase de desarrollo es el diseño del producto, que trae consigo el diseño de casos de prueba. Durante las siguientes fases de codificación y pruebas del producto, se ejecutan las pruebas

unitarias, de sistemas, de integración, etc., de las que se explicará en apartados siguientes.

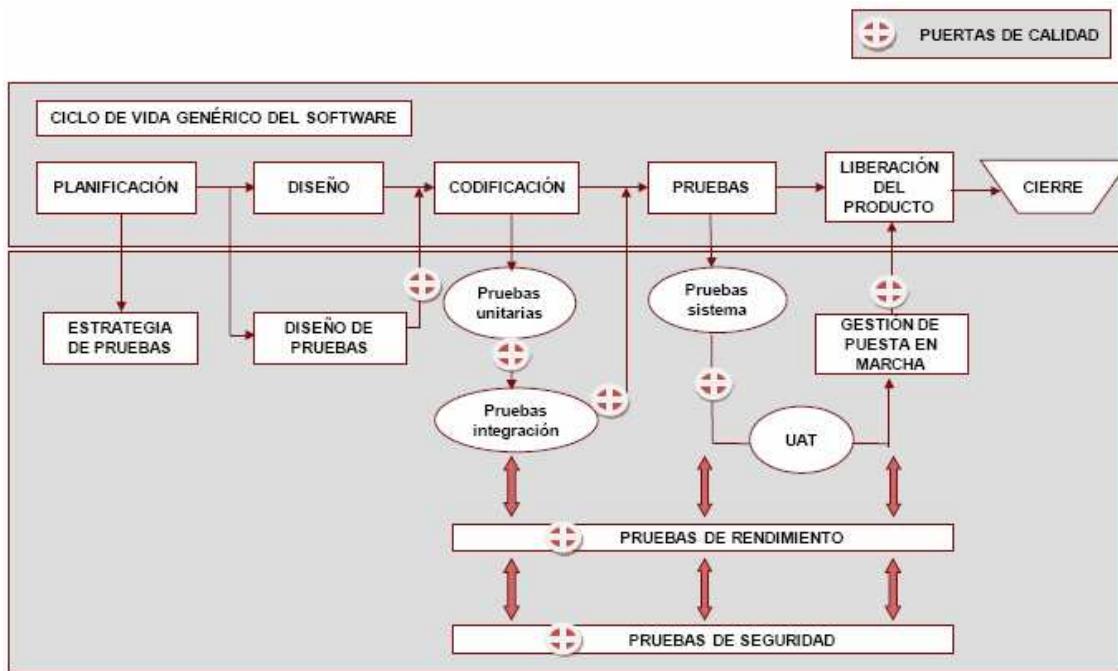


Figura 1.1. Proceso de pruebas en el ciclo de vida de desarrollo de software

De lo anterior, el proceso de pruebas puede considerarse como un subproyecto dentro del proyecto sobre el cual se están ejecutando las pruebas, y como tal requiere la definición de un plan a seguir. Cuando el proceso de pruebas existe dentro del contexto del proyecto, debería prestarse atención a la efectividad y eficiencia de las pruebas desde la perspectiva del proyecto y no desde la perspectiva del propio subproyecto de pruebas.

La eficiencia consiste en conseguir el efecto deseado de la manera correcta, es decir, sin desaprovechamiento de recursos, ni de tiempo ni de dinero. Por consiguiente, la eficiencia está relacionada con dos conceptos: productividad y ausencia de pérdidas. Para conseguir esta eficiencia deseada durante el proceso de pruebas, se pueden considerar los siguientes aspectos:

- **Evitar redundancias:** Las redundancias traen consigo una pérdida o desaprovechamiento de tiempo por lo que hay que intentar ejecutar las pruebas más adecuadas a cada situación y lo más rápido posible. Es importante encontrar los errores que más impacto puedan tener sobre el producto lo más rápido posible. Aunque sea aconsejable no desaprovechar el tiempo, no hay que olvidarse de la planificación, preparación de las pruebas, y de prestar atención a todos los detalles. No abandonar las estrategias previamente establecidas ante la primera señal de problemas o retrasos. Es decir, en un intento de ahorrar tiempo, se debe tener cuidado de no cometer errores que tendrán como consecuencias invertir más tiempo del que se intenta ahorrar.
- **Reducir costes:** Para reducir costes se debería prestar especial atención a la adquisición de elementos que puedan ayudar a la ejecución de pruebas, del tipo de herramientas para la ejecución de pruebas o entornos de pruebas. Habría que cerciorarse de que realmente fueran necesarias y de

que existe el tiempo y las habilidades suficientes para emplearlas de manera ventajosa. También, es aconsejable evaluar las herramientas antes de comprarlas y ser capaz de justificar estos gastos frente al análisis de costos-beneficios.

Un posible flujo del proceso de pruebas, identificando distintas actividades y entregables se muestra en la figura 1.2.

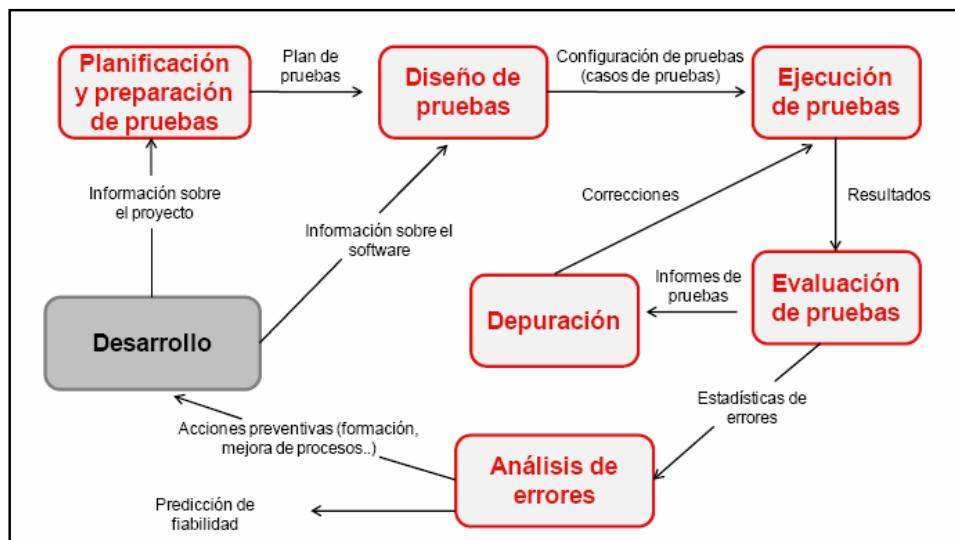


Figura 1.2. Flujo del Proceso de pruebas

### 1.1.1. Validación y Verificación en el desarrollo de software

Los procesos de validación y verificación determinan si un producto software satisface las necesidades del negocio y si se está construyendo acorde a las especificaciones.

Con respecto a las tareas asociadas a estos procesos, las pruebas están más relacionadas con el proceso de validación, mientras que las revisiones son tareas más orientadas al proceso de verificación.

El objetivo principal de la validación y verificación es comprobar que el sistema está hecho para un propósito, es decir, que el sistema debe ser lo suficientemente bueno para su uso previsto. El nivel de confianza requerido depende de tres factores:

- **El propósito o función del sistema.** El nivel de confianza necesario depende de lo crítico que sea el software para una organización.
- **Las expectativas del usuario.** Actualmente, es menos aceptable entregar sistemas no fiables, por lo que las compañías deben invertir más esfuerzo en llevar a cabo las actividades de verificación y validación.
- **Entorno de mercado actual.** Cuando un sistema se comercializa, los vendedores del sistema deben tener en cuenta los sistemas competidores, el precio que sus clientes están dispuestos a pagar por el sistema y los plazos requeridos para entregar dicho sistema. Cuando una compañía tiene pocos competidores, puede decidir

entregar un programa antes de que haya sido completamente probado y depurado, debido a que quiere ser el primero en el mercado. Cuando los clientes no están dispuestos a pagar precios altos por el software, pueden estar dispuestos a tolerar más defectos en él.

Todos estos factores pueden considerarse a fin de decidir cuánto esfuerzo debería invertirse en el proceso de validación y verificación.

#### 1.1.1.1. Validación.

El propósito de la Validación en CMMI® es demostrar que un producto o componente del mismo satisface el uso para el que se creó al situarlo sobre el entorno previsto.

Según Boehm, la validación responde la siguiente pregunta: ¿Se está construyendo el producto correcto?

#### 1.1.1.2. Verificación.

El propósito de la Verificación en CMMI® es asegurar que los productos seleccionados cumplen los requisitos especificados.

Para diferenciar esta tarea con la validación, Boehm indica que debe responderse a la siguiente pregunta: ¿Se está construyendo el producto de la manera correcta?

### 1.1.2. Tipos de pruebas

La disciplina de pruebas es una de las más costosas del ciclo de vida software. En sentido estricto, deben realizarse las pruebas de todos los artefactos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, casos de uso, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de productos que forman parte de la aplicación (por ejemplo, la base de datos), e infraestructura. Obviamente, se aplican diferentes técnicas de prueba a cada tipo de producto software.

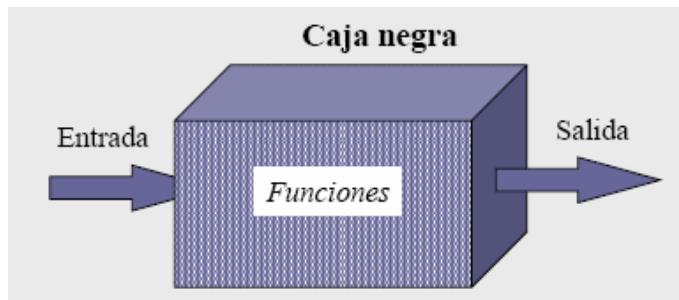
A continuación, se describirá los tipos de pruebas en función de qué conocemos, según el grado de automatización y en función de qué se prueba.

#### 1.1.2.1. En función de qué conocemos.

##### a. Pruebas de caja negra

En este tipo de prueba, tan sólo, podemos probar dando distintos valores a las entradas. Los datos de prueba se escogerán atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

Este tipo de prueba se centra en los requisitos funcionales del software y permite obtener entradas que prueben todos los flujos de una funcionalidad (casos de uso).



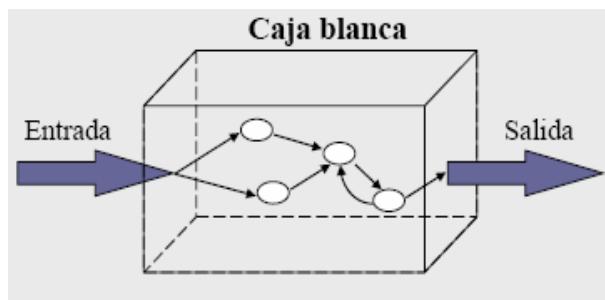
**Figura 1.3. Pruebas de caja negra**

Con este tipo de pruebas se intenta encontrar:

- Funcionalidades incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y finalización.

#### b. Pruebas de caja blanca

Consiste en realizar pruebas para verificar que líneas específicas de código funcionan tal como está definido. También se le conoce como prueba de caja-transparente. La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.



**Figura 1.4. Pruebas de caja blanca**

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo
- Se utilizan las decisiones en su parte verdadera y en su parte falsa
- Se ejecuten todos los bucles en sus límites
- Se utilizan todas las estructuras de datos internas.
- Para esta prueba, se consideran tres importantes puntos.
  - ✓ Conocer el desarrollo interno del programa, determinante en el análisis de coherencia y consistencia del código.

- ✓ Considerar las reglas predefinidas por cada algoritmo.
- ✓ Comparar el desarrollo del programa en su código con la documentación pertinente.

#### 1.1.2.2. Según el grado de automatización

##### a. Pruebas manuales

Una prueba manual es una descripción de los pasos de prueba que realiza un evaluador (usuario experto). Las pruebas manuales se utilizan en aquellas situaciones donde otros tipos de prueba, como las pruebas unitarias o las pruebas Web, serían demasiado difíciles de realizar o su creación y ejecución sería demasiado laboriosa.

También, podría utilizar una prueba manual en situaciones donde no sea posible automatizar los pasos, por ejemplo, para averiguar el comportamiento de un componente cuando se pierde la conexión de red; esta prueba podría realizarse de forma manual, desenchufando el cable de red. Otro ejemplo, sería en caso de comprobar cómo se visualiza el contenido de una página web en dos navegadores diferentes.

Las pruebas manuales ayudarán a descubrir cualquier problema relacionado con la funcionalidad de su producto, especialmente defectos relacionados con facilidad de uso y requisitos de interfaces.

##### b. Pruebas automáticas

A diferencia de las pruebas manuales, para este tipo de pruebas, se usa un determinado software para sistematizarlas y obtener los resultados de las mismas. Por ejemplo, verificar un método de ordenación.

La suite de *IBM Rational* nos provee varias herramientas que permiten automatizar las pruebas de un sistema.

#### 1.1.2.3. En función de qué se prueba

##### a. Pruebas unitarias

Se aplican a un componente del software. Podemos considerar como componente (elemento indivisible) a una función, una clase, una librería, etc.

Estas pruebas las ejecuta el desarrollador, cada vez que va probando fragmentos de código o *scripts* para ver si todo funciona como se desea. Estas pruebas son muy técnicas. Por ejemplo, probar una consulta, probar que un fragmento de código envíe a imprimir un documento, probar que una función devuelva un *flag*, etc.

<b>Entradas</b>	<i>Código Software Diseño Detallado</i>
<b>Salidas</b>	<i>Módulo probado y listo para integrar</i>
<b>Roles</b>	<i>Desarrollador</i>

**Tabla 1.1. E/S y roles en las pruebas unitarias**

Para que una prueba unitaria, tenga éxito se deben cumplir los siguientes requisitos:

- Automatizable: no debería existir intervención manual. Esto es, especialmente, útil para la integración continua.
- Completas: deben cubrir la mayor cantidad de código.
- Repetibles o Reutilizables: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También, es útil para integración continua.
- Independientes: la ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el fragmento de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

- Fomentan el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se llama refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Simplifica la integración: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.
- Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- Separación de la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos.

Es importante darse cuenta de que las pruebas unitarias no descubrirán todos los errores del código, pues solo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

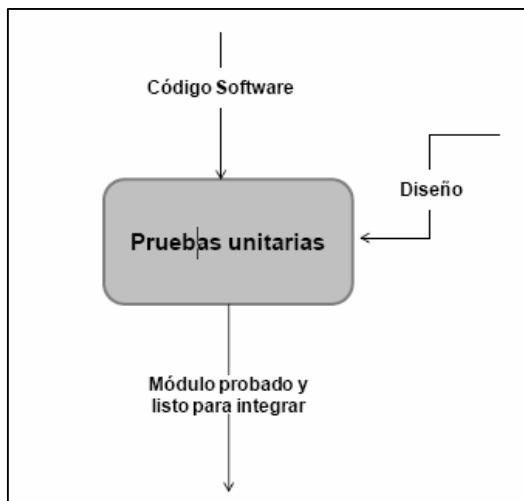


Figura 1.5. Flujo de control de pruebas unitarias

b. Pruebas de integración

Consiste en construir el sistema a partir de los distintos componentes y probarlo con todos integrados. Estas pruebas deben realizarse progresivamente. El foco de atención es el diseño y la construcción de la arquitectura de software.

<i>Entradas</i>	<i>Producto Integrado</i> <i>Plan de Pruebas de Integración</i>
<i>Salidas</i>	<i>Informe de Pruebas de Integración</i> <i>Producto listo para su entrega a pruebas</i>
<i>Roles</i>	<i>Ingeniero de pruebas</i> <i>Jefe de desarrollo</i>

Tabla 1.2. E/S y roles en las pruebas de integración

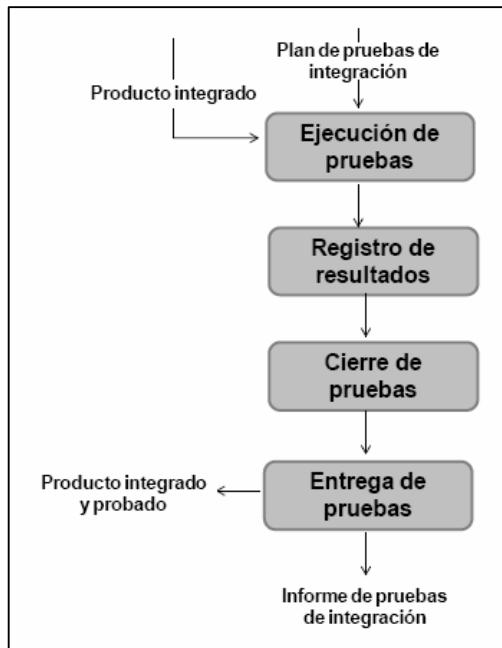


Figura 1.6. Flujo de control de pruebas de integración

c. Pruebas de aceptación

Son las únicas pruebas que son realizadas por los usuarios expertos, todas las anteriores las lleva a cabo el equipo de desarrollo. Consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario. Podemos distinguir entre dos tipos de pruebas; en ambas existe retroalimentación por parte del usuario experto:

- Pruebas alfa: las realiza el usuario en presencia de personal de desarrollo del proyecto haciendo uso de una máquina preparada para las pruebas.
- Pruebas beta: las realiza el usuario después de que el equipo de desarrollo les entregue una versión casi definitiva del producto.

<b>Entradas</b>	<i>Especificación de Requisitos</i>
	<i>Manuales de Usuario</i>
	<i>Sistema probado</i>
	<i>Plan de Pruebas</i>
<b>Salidas</b>	<i>Resultados de pruebas</i>
	<i>Producto aceptado</i>
	<i>Informe de Pruebas de Aceptación</i>
<b>Roles</b>	<i>Ingeniero de pruebas</i>
	<i>Jefe de pruebas</i>
	<i>Jefe de proyecto</i>

Tabla 1.3. E/S y roles en las pruebas de aceptación

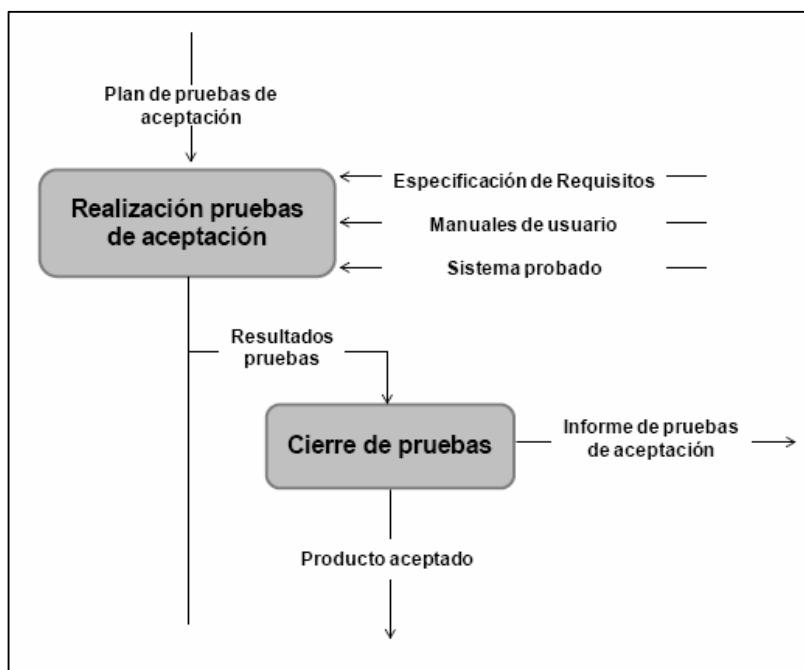


Figura 1.7. Flujo de control de pruebas de aceptación

d. Pruebas funcionales

Este tipo de prueba se realiza sobre el sistema funcionando, comprobando que cumpla con la especificación (normalmente a través de los casos de uso). Para estas pruebas, se utilizan las especificaciones de casos de prueba.

e. Pruebas de rendimiento

Las pruebas de rendimiento se basan en comprobar que el sistema puede soportar el volumen de carga definido en la especificación, es decir, hay que comprobar la eficiencia (por ejemplo, se ha montado una página web sobre un servidor y hay que probar qué capacidad tiene el estado de aceptar peticiones, es decir capacidad de concurrencia).

### 1.1.3. Diseño de casos de pruebas

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba como, por ejemplo, verificar el cumplimiento de un requisito específico. Para llevar a cabo un caso de prueba, es necesario definir las precondiciones y post condiciones, identificar unos valores de entrada, y conocer el comportamiento que debería tener el sistema ante dichos valores. Tras realizar ese análisis e introducir dichos datos en el sistema, se observará si su comportamiento es el previsto o no y por qué. De esta forma se determinará si el sistema ha pasado o no la prueba. De ahí su importancia durante la ejecución de pruebas.

A continuación, se describirá los pasos que se realizarán en el curso para diseñar casos de pruebas.

#### 1.1.3.1. Definir escenarios.

Consiste en identificar todos los escenarios (caminos) a probar de un caso de uso: flujo básico, sub flujos y flujos alternativos.

Para definir el mínimo número de escenarios (caminos independientes) para un caso de uso se puede aplicar la técnica de la complejidad ciclomática. El cálculo de la complejidad ciclomática (CC) consiste en obtener un valor a partir del grafo del caso de uso; este valor es conocido como  $V(G)$ . Existen 3 métodos de cálculo:

a. Según aristas y nodos

$V(G) = a - n + 2$ , siendo  $a$  el número de arcos o aristas del grafo y  $n$  el número de nodos.

b. Según áreas cerradas

$V(G) = r + 1$ , siendo  $r$  el número de regiones cerradas del grafo.

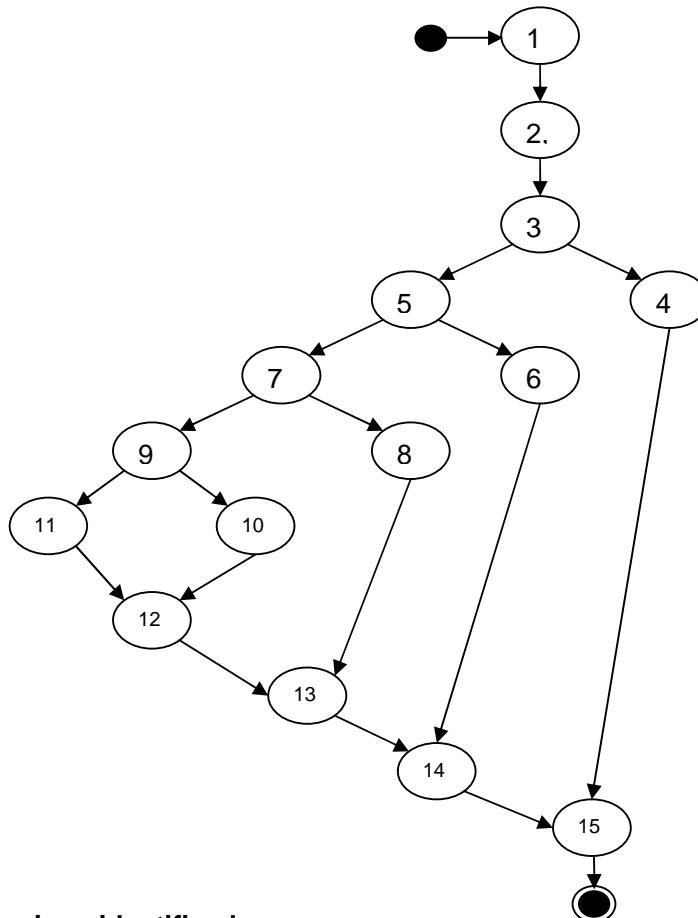
c. Según nodos predicados

$V(G) = c + 1$ , siendo  $c$  el número de nodos predicados (condicionados).

La experiencia en este campo asegura que:

- $V(G)$  marca el límite mínimo de casos de prueba para un caso de uso
- Cuando  $V(G) > 10$  la probabilidad de defectos crece mucho.

**Ejemplo 1:** Para el siguiente diagrama de grafo se calcula el mínimo número de escenarios. Para ello, se aplica los 3 métodos de cálculo de la complejidad ciclomática.



#### Caminos identificados

Camino 1: 1 - 2 - 3 - 4 - 15

Camino 2: 1 - 2 - 3 - 5 - 6 - 14 - 15

Camino 3: 1 - 2 - 3 - 5 - 7 - 8 - 13 - 14 - 15

Camino 4: 1 - 2 - 3 - 5 - 7 - 9 - 10 - 12 - 13 - 14 - 15

Camino 5: 1 - 2 - 3 - 5 - 7 - 9 - 11 - 12 - 13 - 14 - 15

#### Cálculo de la Complejidad Ciclomática (CC)

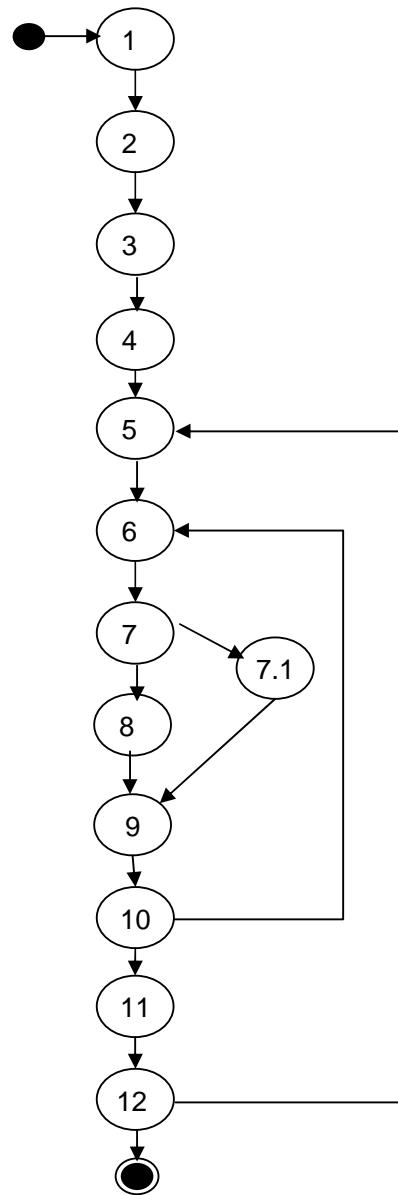
$$\begin{aligned} \text{a. } V(G) &= A - N + 2 \\ V(G) &= 18 - 15 + 2 \\ V(G) &= 5 \end{aligned}$$

Donde: A = Aristas y N = Nodos

$$\begin{aligned} \text{b. } V(G) &= 4 \text{ Regiones} + 1 \\ V(G) &= 5 \end{aligned}$$

$$\begin{aligned} \text{c. } V(G) &= 4 \text{ Nodos Predicados} + 1 \\ V(G) &= 5 \end{aligned}$$

**Ejemplo 2:** Para el siguiente diagrama de grafo se calcula el mínimo número de escenarios. Para ello, se aplica los 3 métodos de cálculo de la complejidad ciclomática.



#### Caminos identificados

Camino 1: 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12

Camino 2: 1 - 2 - 3 - 4 - 5 - 6 - 7 - 7.1 - 9 - 10 - 11 - 12

Camino 3: 1-2-3-4-5-6-7-8-9-10-6-7-8-9-10-11-12

Camino 4: 1-2-3-4-5-6-7-8-9-10-11-12-5-6-7-8-9-10-11-12

#### Cálculo de la Complejidad Ciclomática (CC)

a.  $V(G) = A - N + 2$

$$V(G) = 15 - 13 + 2$$

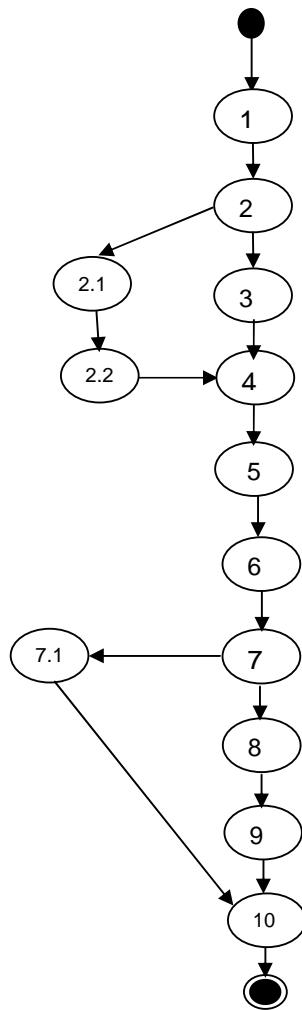
$$V(G) = 4$$

Donde: A = Aristas y N = Nodos

- b.  $V(G) = 3$  Regiones + 1  
 $V(G) = 4$
- c.  $V(G) = 3$  Nodos Predicados + 1  
 $V(G) = 4$

Tal como se especificó en este apartado, con la técnica de la CC se define el mínimo número de escenarios para un caso de uso. En algunos casos, se pueden identificar más escenarios combinando escenarios simples con flujos alternativos anidados.

Por ejemplo: Sea el siguiente diagrama de grafo de un caso de uso.



Si solo consideramos el número de caminos independientes que existen en el grafo, según el cálculo de la CC, el caso de uso tendría 3 escenarios:

Escenarios	Flujo inicial	Alternativo	Secuencia de eventos
1	Básico		1 al 10
2	Básico	FA1, Básico	1,2,2.1,2.2,4 al 10
3	Básico	FA2, Básico	1,2,3,4,5,6,7,7.1,10

Tabla 1.4. Escenarios independientes

Considerando flujos alternativos anidados se tendría 4 escenarios:

Escenarios	Flujo inicial	Alternativo	Secuencia de eventos
1	Básico		1 al 10
2	Básico	FA1, Básico	1,2,2.1,2.2,4 al 10
3	Básico	FA2, Básico	1,2,3,4,5,6,7,7.1,10
4	Básico	FA1, Básico, FA2	1,2,2.1,2.2,4,5,6,7,7.1,10

Tabla 1.5. Escenarios con flujos anidados

#### 1.1.3.2. Identificar condiciones de entrada.

Las condiciones de entrada son parte del dominio de valores de entrada. Se pueden identificar condiciones de entrada con estados **válidos (V)** y **no válidas (NV)**; asimismo se consideran condiciones de entrada con el estado que **no se aplica (N/A)** para un determinado escenario.

Existen los siguientes tipos de condiciones de entrada:

- Miembro de un conjunto
- Lógico
- Valor
- Rango

Veamos un ejemplo. Considérese una aplicación bancaria, donde el usuario puede conectarse al banco por Internet y realizar una serie de operaciones bancarias. Una vez accedido al banco con las siguientes medidas de seguridad (clave de acceso y demás), la información de entrada del procedimiento que gestiona las operaciones concretas a realizar por el usuario requiere las siguientes entradas:

- Código de banco. En blanco o número de tres dígitos. En este último caso, el primer dígito tiene que ser mayor que 1.
- Código de sucursal. Un número de cuatro dígitos. El primero de ellos mayor de 0.

- Número de cuenta. Número de cinco dígitos.
- Clave personal. Valor alfanumérico de cinco posiciones.
- Orden. Este valor se seleccionará de una lista desplegable, según la orden que se desee realizar. Puede estar en "Seleccione Orden" o una de las dos cadenas siguientes: "Talonario" o "Movimientos"

En el caso "Talonario" el usuario recibirá un talonario de cheques, mientras que en "Movimientos" recibirá los movimientos del mes en curso. Si no se especifica este dato, el usuario recibirá los dos documentos.

A continuación, se muestra una tabla con estados de las condiciones de entrada por cada resultado esperado.

**NOTA:** La generación de casos de prueba se completará, después de identificar las clases de equivalencia:

ID CP	Escenario	CONDICIONES DE ENTRADA					Resultado esperado
		Código de banco	Código de sucursal	Número de cuenta	Clave personal	Orden	
CP1	Escenario 1	V	V	V	V	V	Mensaje "Envío de talonarios"
CP2	Escenario 1	V	V	V	V	V	Mensaje "Envío de movimientos "
CP3	Escenario 1	V	V	V	V	V	Mensaje "Envío de talonarios y movimientos"
CP4	Escenario 2	NV	V	V	V	V	Mensaje "Código de banco incorrecto"
CP5	Escenario 3	V	NV	V	V	V	Mensaje "Código de sucursal incorrecto"
CP6	Escenario 4	V	V	NV	V	V	Mensaje "Número de cuenta incorrecto"
CP7	Escenario 5	V	V	V	NV	V	Mensaje "Clave incorrecta"

**Tabla 1.6. Condiciones de entrada**

#### 1.1.3.3. Definir clases de equivalencia.

Pueden usarse varias técnicas para identificar los valores de los datos de entrada, la técnica de particiones o clases de equivalencias es una de ellas.

Las clases de equivalencia se identifican examinando cada condición de entrada (normalmente una frase en la especificación) y dividiéndola en dos o más grupos. Se definen dos tipos de clases de equivalencia:

- **Clases Válidas:** Entradas válidas al programa.
- **Clases no Válidas:** Valores de entrada erróneos.

Estas clases se pueden representar en una tabla. A continuación, se muestra las clases de equivalencia para el caso de gestión bancaria anterior:

Sec.	Condición de Entrada	Tipo	Clases Válidas		Clases No Válidas	
			Entrada	Código	Entrada	Código
1	Código de banco	Lógico (puede estar o no)	En blanco	CEV<01>	Un valor no numérico	CENV<01>
		Si está, es Rango	200 <= Código de banco <= 999	CEV<02>	Código de banco < 200 Código de banco > 999	CENV<02> CENV<03>
2	Código de sucursal	Rango	1000 <= Código de sucursal <= 9999	CEV<03>	Código de sucursal < 1000 Código de sucursal > 9999	CENV<04> CENV<05>
3	Número de cuenta	Valor	Cualquier número de 5 dígitos	CEV<04>	Número de más de cinco dígitos Número de menos de cinco dígitos	CENV<06> CENV<07>
4	Clave personal	Valor	Cualquier cadena de caracteres alfanuméricos de 5 posiciones	CEV<05>	Cadena de más de cinco posiciones Cadena de menos de cinco posiciones	CENV<08> CENV<09>
5	Orden	Miembro de un conjunto, con comportamiento distinto	Orden = "Seleccione Orden" Orden = "Talonario" Orden = "Movimientos"	CEV<06> CEV<07> CEV<08>		

Tabla 1.7. Clases de equivalencia

#### 1.1.3.4. Realizar casos de prueba.

En esta última etapa, se generan los casos de pruebas. Para ello, se considera como referencia la tabla de condiciones de entrada, indicando en cada caso de prueba las clases de equivalencia creadas. Por ejemplo, para el caso bancario se tendría lo siguiente:

ID CP	Clases de equivalencia	CONDICIONES DE ENTRADA					Resultado esperado
		Código de banco	Código de sucursal	Número de cuenta	Clave personal	Orden	
CP1	CEV<02>, CEV<03>, CEV<04>, CEV<05>, CEV<07>	200	1000	10000	Aaaaa	"Talonario"	Mensaje "Envío de talonarios"
CP2	CEV<01>, CEV<03>, CEV<04>, CEV<05>, CEV<08>	820	9999	99999	Zzzzz	"Movimientos"	Mensaje "Envío de movimientos "
CP3	CEV<02>, CEV<03>, CEV<04>, CEV<05>, CEV<06>	999	1001	12345	A1b2c	"Seleccione Orden"	Mensaje "Envío de talonarios y movimientos"
CP4	CENV<01>, CEV<03>, CEV<04>, CEV<05>, CEV<07>	30A	1989	12345	1a2b3	"Seleccione Orden"	Mensaje "Código de banco incorrecto"
CP5	CENV<04>, CEV<03>, CEV<04>, CEV<05>, CEV<07>	210	999	12345	1a2b3	"Seleccione Orden"	Mensaje "Código de sucursal incorrecto"
CP6	CENV<07>, CEV<03>, CEV<04>, CEV<05>, CEV<07>	210	1989	123	1a2b3	"Seleccione Orden"	Mensaje "Número de cuenta incorrecto"
CP7	CENV<09>, CEV<03>, CEV<04>, CEV<05>, CEV<07>	210	1989	12345	""	"Seleccione Orden"	Mensaje "Clave incorrecta"

Tabla 1.8. Casos de prueba

Complete la generación de casos de prueba. Para ello considere las clases de equivalencia.

ID CP	Clases de equivalencia	CONDICIONES DE ENTRADA					Resultado esperado
		Código de banco	Código de sucursal	Número de cuenta	Clave personal	Orden	

## 1.2. ADMINISTRACIÓN DE PRUEBAS

Sin importar la metodología utilizada por un equipo de desarrollo, las pruebas realizadas sobre el producto, ya sean unitarias, de integración o de sistema, reflejan si los objetivos planteados a nivel de requerimientos son satisfechos por el producto desarrollado. Este proceso, típicamente involucra el diseño e implementación de las pruebas, su ejecución, el reporte de los defectos encontrados, la planeación de las correcciones y la implementación de dichas correcciones. Estos pasos no siempre están formalmente definidos y en muchas ocasiones se encuentran en las cabezas de los miembros del equipo de trabajo quienes deciden en un momento dado cómo manejar la información asociada a las pruebas y qué pasos seguir durante este proceso.

Por otro lado, los responsables del proceso de pruebas deben considerar optimizar las pruebas. Para ello, es necesaria la definición de una buena estrategia, es decir, definir una serie de principios e ideas que puedan ayudar a guiar las actividades de pruebas.

### 1.2.1. Estrategia de pruebas

Existen distintas estrategias de pruebas, y dependiendo de su alineamiento con el objetivo de las pruebas y con los propios proyectos, estas estrategias pueden tener éxito o fallar. Otro punto a tener en cuenta es que no tiene por qué elegirse una sola estrategia. Puede utilizarse una estrategia de manera dominante y utilizar otras de complemento.

Algunos autores como Krutchen, Pressman, Pfleger, Cardoso y Sommerville afirman que el proceso de ejecución de pruebas debe ser considerado durante todo el ciclo de vida de un proyecto, para así obtener un producto de alta calidad. Su éxito dependerá del seguimiento de una **Estrategia de Prueba** adecuada.

Los tipos de estrategia de pruebas más importantes se describen a continuación:

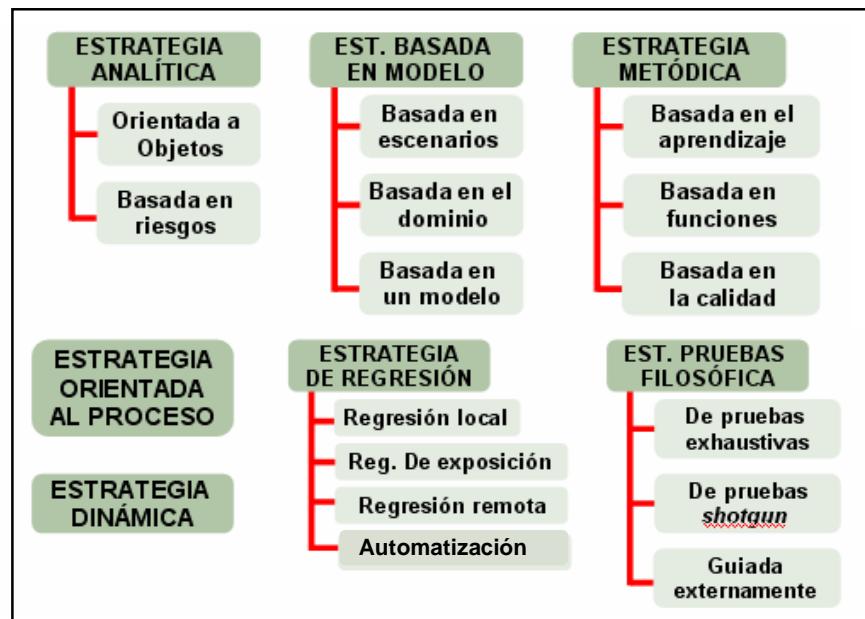


Figura 1.8. Resumen de estrategias de prueba

#### 1.2.1.1. Estrategia analítica.

Las estrategias de pruebas analíticas tienen en común el uso de alguna técnica analítica formal o informal normalmente durante la etapa de gestión de requisitos y de diseño del proyecto.

Por ejemplo:

- a) Estrategia orientada a objetos. Para determinar el enfoque de las pruebas, se presta atención a los requisitos, el diseño, y la implementación de objetos. Estos objetos pueden incluir especificaciones de requisitos, especificaciones de diseño, diagramas UML, casos de uso, código fuente, esquemas de base de datos y diagramas entidad-relación. Este enfoque se basa en una buena documentación, por lo que la ausencia de la misma implica no poder utilizarla.
- b) Estrategia basada en riesgos. Consiste en identificar riesgos estudiando el sistema, documentación de proyectos anteriores, objetivos de la configuración y cualquier otro dato que pueda encontrarse o que pueda aportar la gente involucrada en el proyecto. El diseño, desarrollo y ejecución de pruebas basadas en el conocimiento previo puede ser beneficioso. Este enfoque es apropiado si se dispone de tiempo para investigar el sistema.

Los elementos que están siendo analizados a menudo se denominan “base de las pruebas”. Los resultados del análisis guían el esfuerzo de pruebas, a menudo a través de algunas formas de análisis de cobertura durante el diseño, desarrollo de la ejecución y obtención de resultados de las pruebas. Estas estrategias tienden a ser minuciosas y buenas a la hora de mitigar riesgos de calidad y encontrar errores. Sin embargo, se requiere una inversión de tiempo importante.

#### 1.2.1.2. Estrategia basada en modelo

Las estrategias de pruebas basadas en modelo desarrollan modelos que representan cómo el sistema debería comportarse o trabajar. Las estrategias de pruebas basadas en modelo tienen en común la creación o selección de algún modelo formal o informal para simular comportamientos de sistemas críticos, normalmente durante las etapas de requisitos y diseño del proyecto. Existen distintos tipos de estrategias basadas en modelos:

- a) Con una **estrategia basada en escenario** se realizan pruebas acorde a escenarios del mundo real. Estos escenarios deberían abarcar la funcionalidad del sistema. En el mundo orientado a objetos, se podría usar una estrategia basada en casos de uso, donde se confíe en documentos de diseño orientados a objetos conocidos como casos de uso. Estos casos de uso son modelos de cómo el usuario, clientes y otras partes involucradas en el negocio utilizan el sistema y cómo debería trabajar bajo estas condiciones.

- b) Con una **estrategia basada en el dominio** se pueden analizar diferentes dominios de datos de entrada aceptados por el sistema, procesamiento de datos del sistema, y datos de salida entregados por el sistema. Se decidirá cuál será el mejor caso de pruebas para cada dominio, se determinará la probabilidad de errores, frecuencia de uso y entornos desplegados.
- c) Con una **estrategia basada en un modelo**, se diseñan, desarrollan y ejecutan las pruebas que cubran los modelos que se hayan construido. Esta estrategia será útil dependiendo de la capacidad del modelo para capturar los aspectos esenciales o potencialmente problemáticos del sistema.

#### 1.2.1.3. Estrategia metódica

La estrategia metódica tiende a seguir una línea relativamente informal pero con un enfoque ordenado y predecible que ayuda a comprender dónde probar.

- a) **Estrategia basada en el aprendizaje.** Se utilizan listas de control que se han desarrollado para guiar el proceso de pruebas. Para desarrollar estas listas de control puede ser de ayuda el basarse en los errores que se han encontrado previamente, en lecciones aprendidas de otros proyectos y en cualquier otra fuente.
- b) **Estrategia basada en funciones.** Se identifica y se prueba cada función del sistema por separado. Lo mismo ocurre con la **estrategia basada en estados**, donde se identifica y prueba cada estado y cada posible transición de estados que pueda ocurrir.
- c) **Estrategia basada en la calidad:** Se utiliza una jerarquía de calidad, como la que ofrece la ISO 9126, para identificar y probar la importancia de cada una de las características de la jerarquía como, por ejemplo, la facilidad de uso, el rendimiento, la funcionalidad o la escalabilidad.

Con una estrategia de pruebas metódica, se siguen estos estándares como objetivos de pruebas. Estas estrategias pueden ser rápidas y efectivas en contra de sistemas que permanecen relativamente estables, o sistemas que son similares a otros ya probados. Los cambios significativos pueden frenar temporalmente estas estrategias hasta que se puedan volver a ajustar los objetivos de las pruebas a la nueva situación del sistema.

#### 1.2.1.4. Estrategia orienta al proceso o estándar conformista

Las estrategias de pruebas orientadas al proceso llevan el enfoque metódico un paso más allá a la hora de regular el proceso de pruebas. Estas estrategias siguen un desarrollo externo orientado a pruebas a menudo con pocas posibilidades de personalización. Un ejemplo de este tipo de estrategias es la

**estrategia de pruebas estandarizada**, se sigue un estándar oficial y reconocido, por ejemplo el estándar IEEE 829 orientado a la documentación de las pruebas. Este estándar, por ejemplo, es utilizado en algunas organizaciones para asegurar la regularidad y completitud de todos los documentos de pruebas. Esta estandarización puede ayudar a hacer que el proceso de pruebas sea más transparente y comprensible para los programadores, responsables, analista de negocio y otras personas ajenas a las pruebas.

#### 1.2.1.5. Estrategia dinámica

Las estrategias de pruebas dinámicas, como las estrategias de pruebas ágiles, minimizan la planificación por adelantado y prueban el diseño. Adaptan todo lo posible el sistema bajo prueba a las condiciones que habrá cuando se libere. Típicamente, enfatizan las últimas etapas de pruebas. Se trata de crear un pequeño conjunto de pautas de pruebas que se enfoquen en debilidades conocidas del software.

- a) Con una **estrategia de pruebas intuitiva**, se puede probar acorde a la experiencia e instinto del equipo de pruebas.
- b) Con una **estrategia de pruebas exploratoria**, se puede aprender simultáneamente sobre el comportamiento y diseño de pruebas, a la vez que las pruebas se van ejecutando y se van encontrando errores. Se irá refinando el enfoque de las pruebas en función de los resultados obtenidos de las pruebas.

Las estrategias de pruebas dinámicas valoran la flexibilidad y la facilidad de encontrar errores. Estas estrategias no producen buena información normalmente acerca de cobertura, mitigación sistemática de riesgos ni ofrecen la oportunidad de detectar defectos en las primeras fases del ciclo de vida del producto.

Obviamente, aplicar estas estrategias es mejor que no realizar ningún tipo de pruebas y, cuando van unidas a estrategias analíticas, pueden servir como una excelente herramienta para identificar el vacío dejado por las mismas.

#### 1.2.1.6. Estrategia de pruebas filosófica

Las estrategias de pruebas filosóficas comienzan con una filosofía o creencia de las pruebas.

- a) Cuando se usa una **estrategia de pruebas exhaustiva**, se asume que todo puede tener errores. Es decir, se decide que la posibilidad de que haya fallos latentes es inaceptable por lo que se soportará un considerable esfuerzo al encontrar todos los errores.
- b) Con la **estrategia de pruebas shotgun**, se asume que todo y nada puede tener y tendrá errores. Sin embargo, en este caso, se acepta que no se puede probar todo. Cuando se llegue al punto de no tener una idea sólida acerca de por

dónde empezar a probar, se intentará distribuir de manera aleatoria el esfuerzo de pruebas teniendo en cuenta las restricciones de recursos y la agenda.

- c) Con una **estrategia guiada externamente**, no sólo se acepta que no se puede probar todo, sino que también se asume que no se sabe donde están los errores. Sin embargo, se confía en que otras personas puedan tener una buena idea sobre donde están. Para ello, se pedirá ayuda a estas personas sobre la decisión a tomar acerca de si los resultados obtenidos son o no correctos.

Por ejemplo, se puede preguntar a los usuarios, personas que den soporte técnico, analistas de negocio o desarrolladores del sistema sobre qué probar o incluso confiar en ellos para hacer las pruebas. Enfatizan las últimas etapas de pruebas simplemente debido a la falta de reconocimiento del valor de pruebas tempranas.

#### 1.2.1.7. Regresión

La regresión es la mala conducta de una función, atributo o característica previamente correctos. La palabra regresión también se suele usar para definir el descubrimiento de un error que previamente no se encontró durante la ejecución de una prueba. La regresión normalmente está asociada a algún cambio en el sistema, como añadir una característica o arreglar un error.

La regresión puede ser de tres tipos:

- a) **Regresión local**: al producirse un cambio o arreglarse un error se crea un nuevo error.
- b) **Regresión de exposición**: al producirse un cambio o arreglarse un error se identifican errores ya existentes.
- c) **Regresión remota**: un cambio o el arreglo de un error en un determinado área produce un fallo en otro área del sistema. La regresión remota es la regresión más difícil de detectar, ya que los usuarios, clientes y el resto de los interesados en el proyecto tienden a confiar en las características ya existentes del sistema. Por ello, los técnicos de pruebas deberían tener estrategias que sean efectivas en contra de todas las causas y efectos de las regresiones.

Una posible estrategia para enfrentarse a la regresión, quizás el enfoque más simple, es la **fuerza bruta**. Para la mitigación del riesgo de regresión, la estrategia de la fuerza bruta consiste en repetir todas las pruebas. Imaginemos que se ha desarrollado un conjunto de pruebas bien alineadas con la calidad. En este caso, se habrá desarrollado un análisis de riesgos de calidad sólido y se tendrá tiempo y recursos suficientes para cubrir todos los riesgos críticos de calidad. Si se repiten todas las pruebas después del último cambio realizado, se deberían encontrar todos los errores de regresión importantes.

- d) La **automatización**. Se puede considerar la automatización como una estrategia para aumentar la calidad de un producto a un bajo coste y optimizar el esfuerzo de las pruebas.

La automatización es la única manera de repetir todas las pruebas sobre sistemas complejos y grandes. La automatización es práctica cuando los costes del diseño e implementación de pruebas automatizadas sean recuperables, y se vayan a ejecutar de manera frecuente.

A pesar de su gran utilidad, en ocasiones, la inversión extra que supone para el proyecto y la necesidad de ciertas habilidades por parte de miembros de la organización hacen que la automatización sea un proceso costoso. Sin embargo, algunas de sus ventajas son:

- La automatización puede reducir drásticamente el esfuerzo de las pruebas de regresión.
- La automatización permite realizar validaciones durante los ciclos de cambios, cosa que sería imposible hacer manualmente debido a restricciones de tiempo.
- La automatización habilita la consistencia y cobertura lógica. No hay riesgo alguno de excluir casos de prueba o pasar por alto errores si se diseña correctamente.

Esta estrategia suele envolver pruebas funcionales automatizadas antes de la liberación del producto, pero algunas veces, se centra por completo en funciones liberadas con anterioridad. Por ejemplo, se puede intentar automatizar todas las pruebas de sistemas de tal forma que cuando se produzca cualquier cambio se pueda volver a ejecutar cada prueba para asegurarse de que ningún área se ha visto afectada. Pero, incluso con una automatización efectiva, no siempre es posible repetir todas las pruebas ya que no resulta práctico o su gestión es insostenible. Por ello, en ocasiones se necesitará realizar una selección sobre el conjunto de pruebas a automatizar.

### 1.2.2. Roles y Responsabilidades

En RUP se definen los siguientes roles:

- Administrador de pruebas
- Analista de pruebas
- Diseñador de pruebas
- Ejecutor de pruebas

#### 1.2.2.1. Administrador de pruebas

Es el responsable del éxito total de la prueba. Involucra calidad y aprobación de la prueba, recurso de planeación, dirección, y solución a los problemas que impiden el éxito de la prueba.

El Administrador de pruebas es responsable de los siguientes artefactos: Plan de Pruebas, Resumen de Resultado de Pruebas, Lista de Problemas y Cambios de Requerimientos.

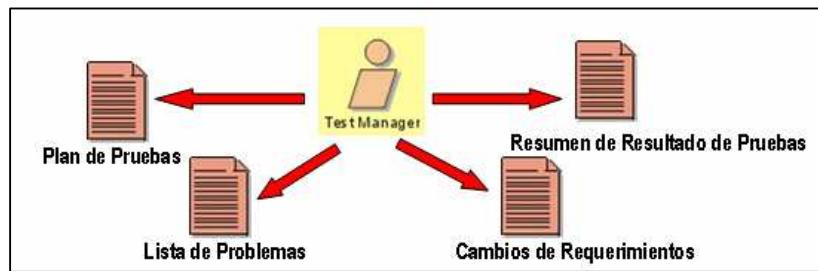


Figura 1.9. El rol de Administrador de pruebas

- **El Plan de Pruebas**, contiene la definición de las metas y objetivos a probar dentro del alcance de cada iteración del proyecto. Proporciona el marco de trabajo en el que el equipo llevará a cabo la prueba dentro de el horario coordinado.
- **El Resumen de Resultados de Pruebas**, organiza y presenta un análisis resumen de los resultados de las pruebas y las medidas clave para revisar y definir estas, típicamente por los Stakeholders claves. Además, puede contener una declaración general de calidad relativa, puede mantener las recomendaciones de las pruebas que se realizaran a futuro.
- La **Lista de los Problemas**, proporciona una manera de registrar para el Administrador del Proyecto los: problemas, excepciones, anomalías, u otras tareas incompletas que requieren atención que relaciona a la dirección del proyecto.
- **Cambios de Requerimientos**. Se proponen cambios a los artefactos de desarrollo a través de Cambios de requerimientos (CR). Se usan los **Cambios de Requerimientos** para documentar los problemas, las mejoras solicitadas y cualquier otro tipo de solicitud para un cambio en el producto. El beneficio de CR es que proporcionan un registro de decisiones, debido a su proceso de valoración, asegura los impactos del cambio que puedan darse en el proyecto.

#### 1.2.2.2. Analista de pruebas

Es el responsable de identificar y definir las pruebas requeridas, mientras supervisa el progreso de la comprobación detallada y resultado por cada ciclo de realización de las pruebas.

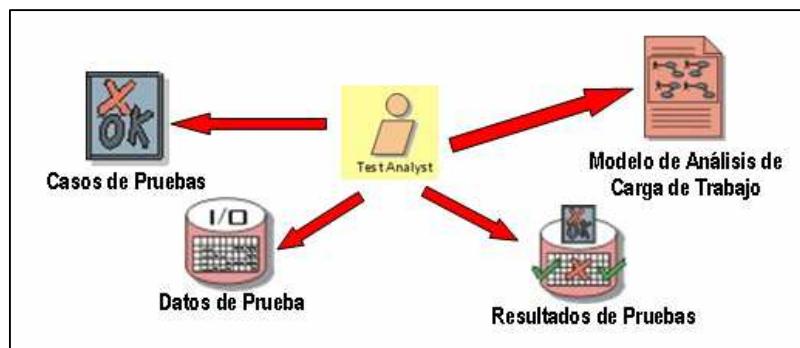


Figura 1.10. El rol de Analista de pruebas

El Analista de pruebas es responsable de los siguientes artefactos: Casos de Pruebas, Modelo de Análisis de Carga de Trabajo, Datos de Prueba, Resultados de Pruebas.

- **Casos de Prueba.** El propósito de estos artefactos es especificar y comunicar las condiciones específicas que necesitan ser validadas para permitir una definición de algunos aspectos particulares de los ítems de prueba objetivo.
- **El Modelo de Análisis de la carga de Trabajo,** trata de definir acertadamente las condiciones de carga bajo los cuales, los ítems de prueba objetivo deben operar en su entorno de configuración del sistema.
- **Datos de Prueba,** es la definición (usualmente formal) de una colección de valores de entrada de prueba que son consumidos durante la ejecución de una prueba
- **Resultados de Pruebas,** es una colección de información sumaria determinada del análisis de unas o más peticiones de los registros y del cambio de la prueba. Referido a veces como un depósito más grande de muchos resultados de la prueba.

#### 1.2.2.3. Diseñador de pruebas

Es el responsable de definir la estrategia de pruebas y de asegurar su puesta en práctica acertadamente. El rol implica identificar técnicas, herramientas y pautas apropiadas para poner en ejecución las pruebas requeridas, y dotar de los recursos necesarios para conducir los requisitos de la prueba.

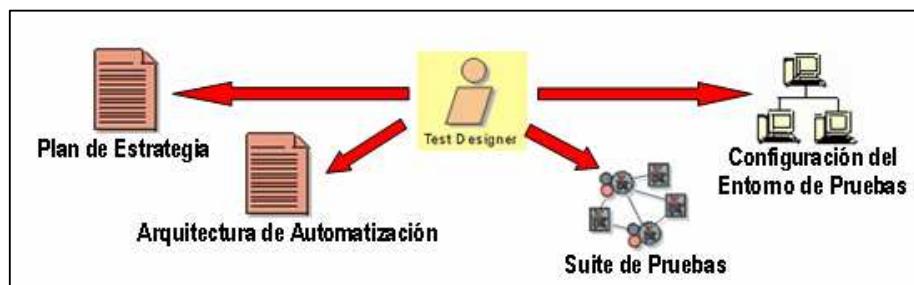


Figura 1.11. El rol de Diseñador de pruebas

El Diseñador de pruebas es responsable de los siguientes artefactos: Plan de Estrategia, Arquitectura de Automatización, Configuración del Entorno de Pruebas, Suite de Pruebas.

- **El Plan de Estrategia,** define el plan estratégico de como el esfuerzo de la prueba será conducido contra uno o más aspectos del sistema. Una estrategia de prueba necesita poder convencer a la gestión y a otros Stakeholders que el acercamiento es alcanzable.
- **Arquitectura de automatización,** es útil cuando la ejecución automatizada de las pruebas del software se debe mantener y ampliar durante ciclos múltiples de prueba. Proporciona una descripción arquitectónica comprensible del sistema de automatización de las pruebas, usando un

número de diversas visiones arquitectónicas para representar diversos aspectos del sistema.

- **Configuración del entorno de pruebas**, es la especificación para un arreglo de hardware, software, y ajustes asociados al entorno que se requieran para permitir las pruebas exactas a ser conducidas que evaluarán uno o más ítems de la prueba objetivo.
- **Suite de Pruebas**, es un tipo de paquete que agrupa las colecciones de pruebas, para ordenar la ejecución de esas pruebas y para proporcionar un sistema útil y relacionado de información del registro de prueba del cual los resultados de prueba pueden ser determinados.

#### 1.2.2.4. Ejecutor de pruebas

Es el responsable de todas las actividades de las pruebas. El rol implica verificar y ejecutar pruebas, y analizar y recoger las ejecuciones de errores.

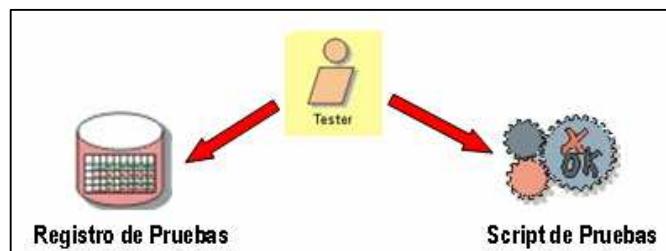


Figura 1.12. El rol del Ejecutor de pruebas

El Ejecutor de pruebas es responsable de los siguientes artefactos: Registro de Pruebas, Script de Pruebas.

- **El registro de las pruebas (Test Log)**, es una colección de salidas capturadas durante la ejecución de una o más pruebas, usualmente representa la salida resultante de la ejecución de un conjunto de pruebas para un ciclo de pruebas.
- **Scripts de prueba (Test Script)**, son instrucciones paso a paso de cómo realizar una prueba, permitiendo su ejecución de una manera efectiva y eficiente.

#### 1.2.3. Técnicas de pruebas

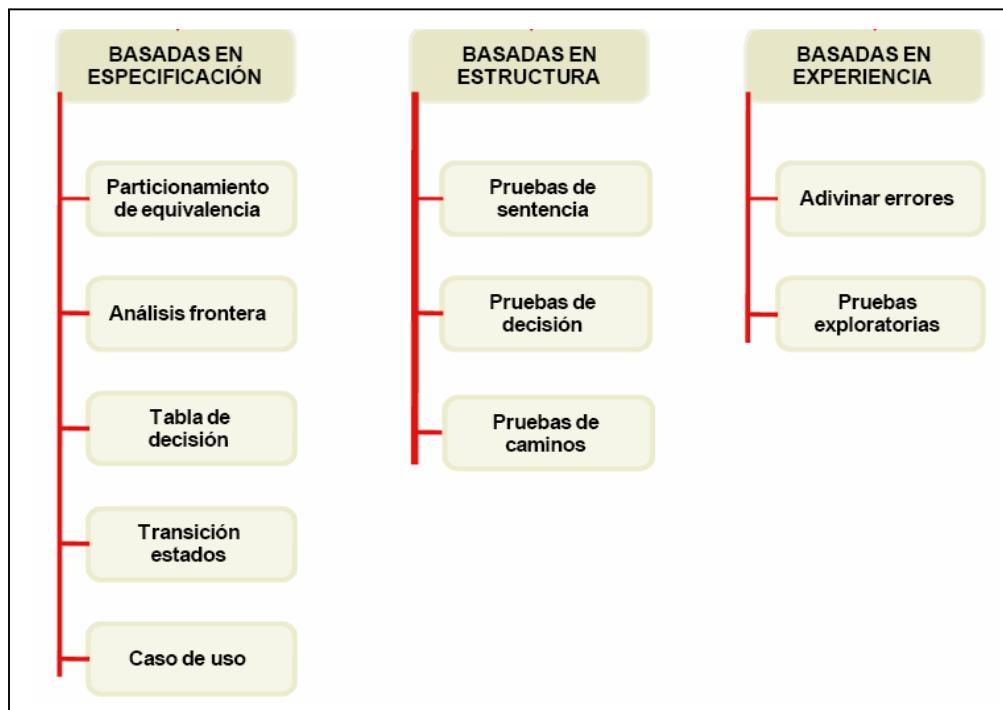
Existen distintas técnicas de pruebas de software, con sus debilidades y sus puntos fuertes. Cada una de ellas es buena para encontrar un tipo específico de defectos. Las pruebas realizadas en distintas etapas del ciclo de desarrollo del software encontrarán diferentes tipos de defectos.

Las pruebas dinámicas sólo se aplican sobre el código del producto para detectar defectos y determinar atributos de calidad del software, por lo que estarían más orientadas al área de validación. Por otra parte, las técnicas de pruebas estáticas son útiles para evaluar o analizar documentos de requisitos, documentación de diseño, planes de prueba, o manuales de usuario, e incluso para examinar el código fuente antes de ejecutarlo. En este sentido, ayudan más a la implementación del proceso de verificación.

Las pruebas estáticas y las pruebas dinámicas son métodos complementarios ya que ambas tratan de detectar distintos tipos de defectos de forma efectiva y eficiente, aunque las pruebas estáticas están orientadas a encontrar defectos mientras que las dinámicas fallos.

#### 1.2.3.1. Técnicas de pruebas dinámicas

Las **técnicas de pruebas dinámicas** ejecutan el software. Para ello introducen una serie de valores de entrada, y examinan la salida comparándola con los resultados esperados.



**Figura 1.13. Clasificación de técnicas dinámicas**

##### a) Basadas en la especificación

Son conocidas como técnicas de pruebas de **caja negra** o conducidas por entradas/salidas, porque tratan el software como una caja negra con entradas y salidas, pero no tienen conocimiento de cómo está estructurado el programa o componente dentro de la caja. Esencialmente, el técnico de pruebas se concentra en qué hace el software y no en cómo lo hace. Las técnicas basadas en especificaciones obtienen los casos de prueba directamente de las especificaciones o de otros tipos de artefactos que contengan lo que el sistema debería hacer.

A continuación, se detallarán las técnicas basadas en la especificación más comunes.

- **Partitionamiento de equivalencia**

Esta técnica consiste en dividir un conjunto de condiciones de prueba en grupos o conjuntos que puedan ser considerados iguales por el sistema. Esta

técnica requiere probar sólo una condición para cada partición. Esto es así porque se asume que todas las condiciones de una partición serán tratadas de la misma manera por el software. Si una condición en una partición funciona, se asume que todas las condiciones en esa partición funcionarán. De la misma forma, si una de las condiciones en una partición no funciona, entonces se asume que ninguna de las condiciones en esta partición en concreto funcionará. Si existe tiempo suficiente, se debería probar más de un valor para una partición, especialmente si se quiere confirmar una selección típica de entradas de usuario.

- **Análisis de valor frontera**

Los errores generados por los programadores tienden a agruparse alrededor de las fronteras. Por ejemplo, si un programa debería aceptar una secuencia de números entre 1 y 10, el error más probable será que los valores justo fuera del rango sean aceptados de forma incorrecta o que los valores justo en los límites del rango sean rechazados. El análisis del valor frontera está basado en probar los valores frontera de las particiones. Al hacer “comprobaciones de rango”, probablemente se esté usando de forma inconsciente el análisis del valor frontera. En esta técnica, también, se cuenta con fronteras válidas (en las particiones válidas) y fronteras no válidas (en las particiones no válidas).

- **Tablas de decisión**

Las técnicas de particionamiento de equivalencia y análisis del valor frontera son aplicadas con frecuencia a situaciones o entradas específicas. Sin embargo, si diferentes combinaciones de entradas dan como resultado diferentes acciones, resulta más difícil usar las técnicas anteriores.

Las especificaciones suelen contener reglas de negocio para definir las funciones del sistema y las condiciones bajo las que cada función opera. Las decisiones individuales son normalmente simples, pero el efecto global de las condiciones lógicas puede ser muy complejo. Los técnicos de pruebas necesitan ser capaces de asegurarse que todas las combinaciones de las condiciones que puedan ocurrir han de ser probadas, por lo tanto, hay que capturar todas las decisiones de una forma que permita explorar sus combinaciones. El mecanismo usado con frecuencia para capturar las decisiones lógicas es la tabla de decisión.

Una tabla de decisión lista todas las condiciones de entrada que pueden ocurrir y todas las acciones que pueden surgir de ellas. Están estructuradas por filas, con las condiciones en la parte de arriba de la tabla y las posibles acciones en la parte de abajo. Las reglas de

negocio, que incluyen combinaciones de condiciones para producir algunas combinaciones de acciones, se incluyen en la parte de arriba de un extremo a otro. Cada columna, por lo tanto, representa una regla de negocio individual y muestra cómo se combinan las condiciones de entrada para producir acciones. De esta manera, cada columna representa un posible caso de prueba, ya que identifica ambas entradas y salidas.

- **Transición de estados**

La técnica anterior (tabla de decisión) es particularmente útil cuando las combinaciones de condiciones de entrada producen varias acciones. La técnica de transición de estados se utiliza con sistemas en los que las salidas son desencadenadas por cambios en las condiciones de entrada, o cambios de “estado”.

Las pruebas de transición de estados se usan cuando alguno de los aspectos del sistema se pueden describir en lo que se denomina una “máquina de estados finitos”. Esto significa que el sistema puede estar en un número (finito) de estados, y las transiciones de un estado a otro están determinadas por las reglas de la “máquina”. Este es el modelo en el que se basan el sistema y las pruebas. Cualquier sistema en el que se puedan conseguir diferentes salidas con la misma entrada, dependiendo de lo que haya pasado antes, es un sistema de estados finitos. Un sistema de estados finitos se suele representar mediante un diagrama de estados.

- **Pruebas de casos de Uso**

Las pruebas de caso de uso son una técnica que ayuda a identificar casos de prueba que ejerciten el sistema entero transición a transición desde el principio al final.

Un caso de uso es una descripción de un uso particular del sistema por un actor (un usuario del sistema). Cada caso de uso describe las interacciones que el actor tiene con el sistema para conseguir una tarea concreta (o, al menos, producir algo de valor al usuario). Los actores son generalmente gente pero pueden ser también otros sistemas. Resumidamente, los casos de uso son una secuencia de pasos que describen las interacciones entre el actor y el sistema.

Los casos de uso están definidos en términos del actor, no del sistema, describiendo qué hace y ve el actor, más que qué entradas o salidas espera el sistema. De forma muy frecuente, usan el lenguaje y términos de negocio más que términos técnicos, especialmente cuando el actor es un usuario de negocio. Sirven como fundamento para desarrollar casos de prueba, en su mayoría, en los niveles de pruebas de sistema y pruebas de aceptación.

Los casos de uso pueden descubrir defectos de integración, defectos causados por la interacción incorrecta entre diferentes componentes.

Los casos de uso describen el flujo de proceso a través de un sistema basado en su uso más probable. Esto hace que los casos de prueba obtenidos de los casos de uso sean particularmente útiles a la hora de encontrar defectos en el uso real del sistema (p.ej. los defectos que los usuarios son más propensos a hacer la primera vez que usan el sistema). Cada caso de uso tiene un escenario dominante (o más probable) y algunas ramas alternativas (cubriendo, por ejemplo, casos especiales o condiciones excepcionales). Cada caso de uso debe especificar cualquier precondición que tenga resultados observables y una descripción del estado final del sistema después de que el caso de uso haya sido ejecutado de forma exitosa.

b) Basadas en la estructura

Las pruebas estructurales son una aproximación al diseño de casos de prueba donde las pruebas se derivan a partir del conocimiento de la estructura e implementación del software. Esta aproximación se denomina a veces pruebas de **caja blanca**.

La comprensión del algoritmo utilizado en un componente puede ayudar a identificar particiones adicionales y casos de prueba, asegurando así un mayor rango de pruebas. Las técnicas más sofisticadas proporcionan, de forma incremental, una cobertura de código completa. A continuación, se detallarán las técnicas basadas en la estructura más comunes.

- **Pruebas de sentencia**

El objetivo de las pruebas de sentencia es ir probando las distintas sentencias a lo largo del código. Si se prueban todas y cada una de las sentencias ejecutables del código, habrá una cobertura de sentencia total. Es importante recordar que estas pruebas sólo se centran en sentencias ejecutables a la hora de medir la cobertura. Es muy útil el uso de gráficos de flujo de datos para identificar este tipo de sentencias, que se representan mediante rectángulos.

Generalmente, la cobertura de las sentencias es demasiado débil para ser considerada una medida adecuada para probar la efectividad.

- **Pruebas de decisión**

El objetivo de estas pruebas es asegurar que las decisiones en un programa son realizadas adecuadamente. Las decisiones son parte de las

estructuras de selección e iteración, por ejemplo, aparecen en construcciones tales como: IF THEN ELSE, o DO WHILE, o REPEAT UNTIL. Para probar una decisión, es necesario ejecutarla cuando la condición que tiene asociada es verdadera y también cuando es falsa. De esta forma, se garantiza que todas las posibles salidas de la decisión se han probado.

Al igual que las pruebas de sentencias, las pruebas de decisión tienen una medida de cobertura asociada e intentan conseguir el 100% de la cobertura de decisión.

- **Pruebas de caminos**

Las pruebas de caminos son una estrategia de pruebas estructurales cuyo objetivo es probar cada camino de la ejecución independientemente. En todas las sentencias condicionales, se comprueban los casos verdadero y falso. En un proceso de desarrollo orientado a objetos, pueden utilizarse las pruebas de caminos cuando se prueban los métodos asociados a los objetos.

Las pruebas de caminos no prueban todas las posibles combinaciones de todos los caminos en el programa. Para cualquier componente distinto de un componente trivial sin bucles, este es un objetivo imposible. Existe un número infinito de posibles combinaciones de caminos en los programas con bucles. Incluso cuando todas las sentencias del programa se han ejecutado al menos una vez, los defectos del programa todavía pueden aparecer cuando se combinan determinados caminos.

c) **Basadas en la experiencia**

Las técnicas basadas en experiencia son aquellas a las que se recurre cuando no hay una especificación adecuada desde la que crear casos de prueba basados en especificación, ni hay tiempo suficiente para ejecutar la estructura completa del paquete de pruebas.

- **Adivinar errores**

Las técnicas basadas en experiencia usan la experiencia de los usuarios y de los técnicos de pruebas para determinar las áreas más importantes de un sistema y ejercitarse en dichas áreas de forma que sean consistentes con el uso que se espera que tengan.

- **Pruebas exploratorias**

Técnicas de pruebas más sofisticadas que se realizan sobre la base del conocimiento y experiencia de los técnicos de pruebas; dicha base es un factor decisivo para el éxito de las pruebas.

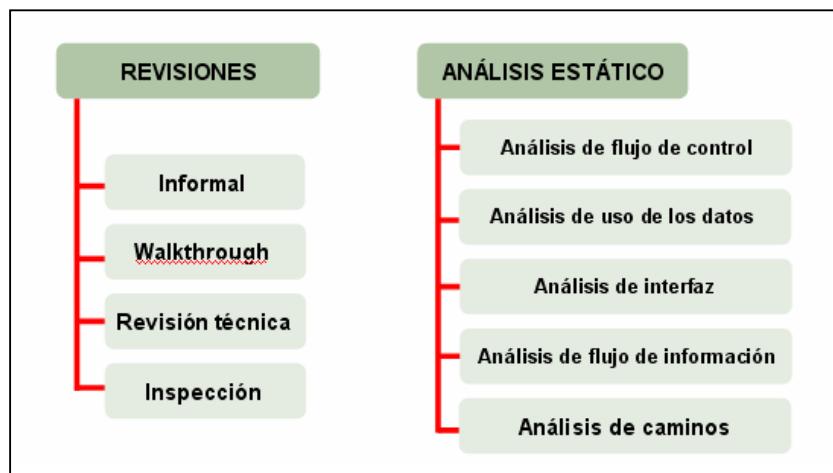
### 1.2.3.2. Técnicas de control estáticas

Aunque las **técnicas estáticas** son utilizadas cada vez más, las pruebas dinámicas siguen siendo las técnicas predominantes de validación y verificación. Las técnicas estáticas permiten encontrar las causas de los defectos. Aplicar un proceso de pruebas estáticas sobre el proceso de desarrollo permite una mejora del proceso evitando cometer errores similares en el futuro.

Las técnicas de pruebas estáticas proporcionan una forma de mejorar la productividad del desarrollo del software. El objetivo fundamental de las pruebas estáticas es mejorar la calidad de los productos software, ayudando a los ingenieros a reconocer y arreglar sus propios defectos en etapas tempranas del proceso de desarrollo. Aunque las técnicas de este tipo de pruebas son muy efectivas, no conviene que sustituyan a las pruebas dinámicas.

Estas pruebas, son las primeras que se aplican al software y buscan defectos sin ejecutar código. Por esta razón, también se llaman “técnicas de no ejecución”. Las técnicas estáticas tienen que ver con el análisis y control de las representaciones del sistema, es decir de los diferentes modelos construidos durante el proceso de desarrollo de software.

La mayoría de las técnicas estáticas son técnicas de verificación que pueden probar cualquier tipo de documentación ya sea código fuente, o documentación y modelos de diseño, o especificación funcional o de requisitos.



**Figura 1.14. Clasificación de técnicas de control estáticas**

#### a) Revisiones

Las revisiones son una **técnica estática** que consiste en realizar un análisis de un documento con el objetivo de encontrar y eliminar errores.

El tipo de una revisión varía en función de su nivel de formalidad. En este caso, ‘formalidad’ se refiere a nivel de estructura y documentación asociada con la revisión. Unos tipos de revisión son completamente informales mientras que otros son muy formales.

No existe una revisión perfecta, sino que cada una tiene un propósito distinto durante las etapas del ciclo de vida del documento. Los principales tipos de revisiones se describen a continuación:

- **Informal**

Dar un borrador de un documento a un colega para que lo lea es el ejemplo más simple de revisión. Es una forma de conseguir beneficios (limitados) a un bajo costo ya que no siguen ningún proceso formal a seguir. La revisión puede implementarse mediante '**pares de programadores**', donde uno revisa el código del otro, o mediante un técnico que lidere el diseño de las revisiones y el código.

- **Walkthrough**

Un walkthrough se caracteriza porque el autor del documento bajo revisión va guiando al resto de participantes a través del documento exponiendo sus ideas para conseguir un entendimiento común y recoger respuestas. Es especialmente útil si los asistentes no están relacionados con el software, o no son capaces de entender los documentos de desarrollo de software. En las revisiones walkthrough se explica el contenido del documento paso a paso hasta llegar a un consenso en los cambios y en el resto de información. La reunión es dirigida por los autores y a menudo está presente un documentador, y para facilitar su ejecución pueden usarse **escenarios** y **simulaciones** para validar el contenido.

- **Revisión técnica**

Una revisión técnica es una reunión que se centra en conseguir consenso sobre el contenido técnico de un documento, por lo que es posible que sea dirigida por un experto técnico. Los expertos necesarios para una revisión técnica son, por ejemplo, responsables del diseño y usuarios clave.

El objetivo principal de este tipo de revisiones es evaluar el valor de conceptos técnicos y alternativas del entorno del proyecto y del propio producto. Este tipo de revisión a menudo se lleva a cabo por “pares” (*peer reviews*) y para facilitar su ejecución suelen utilizarse listas de comprobación o listas de registro.

- **Inspección**

La inspección es el tipo de revisión más formal. El documento bajo inspección es preparado y validado

minuciosamente por revisores antes de la reunión, se compara el producto con sus fuentes y otros documentos, y se usan listas de comprobación. En la reunión de la inspección, se registran los defectos encontrados y se pospone toda discusión para la **fase de discusión**. Todo esto hace que la inspección sea una reunión muy eficiente.

La inspección es dirigida normalmente por un moderador formado, no por el propio autor del documento, quien realiza un seguimiento formal aplicando criterios de salida. Los defectos encontrados son documentados en una lista de registro, y de manera opcional, para mejorar los procesos y aprender de los defectos encontrados, se realiza un análisis de las causas de los mismos. También, es habitual tomar métricas que posteriormente son agrupadas y analizadas para optimizar el proceso.

b) Análisis estático

El análisis estático, al igual que las revisiones, busca defectos sin ejecutar el código. Sin embargo, el análisis estático se lleva a cabo una vez que se escribe el código. Su objetivo es encontrar defectos en el código fuente y en los modelos del software.

Se utilizan herramientas de software para procesar código fuente. Éstas analizan sintácticamente el programa y tratan de descubrir condiciones potencialmente erróneas y llamar la atención del equipo de validación y verificación.

Existen distintos tipos de análisis sintáctico, entre los que se encuentran:

- **Análisis de flujo de control.** Comprueba los bucles con múltiples puntos de entrada o salida, encuentra códigos inalcanzables.
- **Análisis de uso de los datos.** Detecta variables no inicializadas, variables escritas dos veces sin que intervenga una asignación, variables que se declaran pero nunca se usan.
- **Análisis de interfaz.** Comprueba la consistencia de una rutina, las declaraciones del procedimiento y su uso.
- **Análisis de flujo de información.** Identifica las dependencias de las variables de salida. No detecta anomalías en sí pero resalta información para la inspección o revisión del código.
- **Análisis de caminos.** Identifica los caminos del programa y arregla las sentencias ejecutadas en el camino. Es potencialmente útil en el proceso de revisión.

#### 1.2.4. Herramientas de pruebas

Mediante la utilización de herramientas se puede conseguir reproducir cierto trabajo previamente realizado, obteniendo resultados consistentes. Esta característica es especialmente útil para confirmar que un defecto se ha arreglado, o para introducir datos de entrada a pruebas.

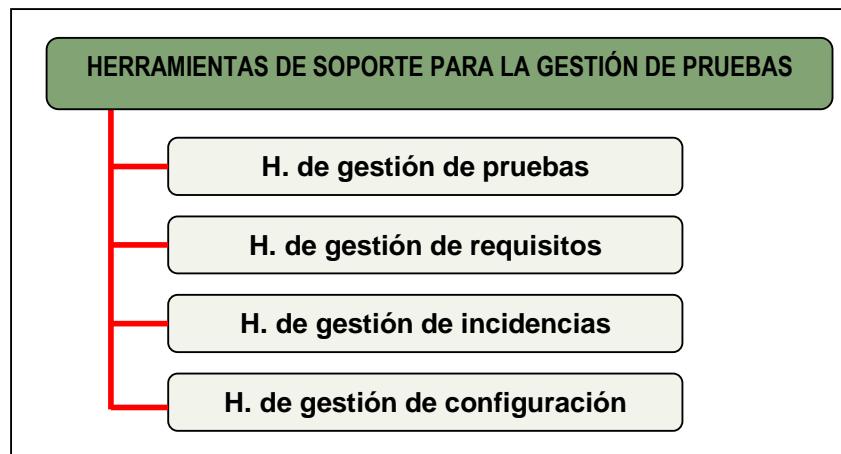
Los principales beneficios que aporta el uso de herramientas como medio para llevar a cabo el proceso de pruebas son:

- Reducir el trabajo repetitivo
- Mejorar la consistencia
- Facilitar las evaluaciones objetivas
- Facilitar el acceso a información relacionada con pruebas

A continuación, se definen una serie de herramientas en base a su funcionalidad. Las herramientas están clasificadas dependiendo de las áreas o actividades de pruebas a las que se enfocan. La mayoría de las herramientas comerciales pueden usarse para diferentes funciones. Por ejemplo, una herramienta de gestión puede proporcionar soporte para la gestión de pruebas, gestión de la configuración, gestión de incidencias y gestión de los requisitos y trazabilidad.

##### 1.2.4.1. Herramientas de soporte para la gestión de pruebas

La gestión de pruebas se aplica sobre el conjunto del ciclo de vida de desarrollo del software, por lo que una herramienta de gestión de pruebas debería de ser la primera en usarse en el proyecto. Esta área abarcaría cuatro tipos de pruebas:



**Figura 1.15. Herramientas de soporte para la gestión de pruebas**

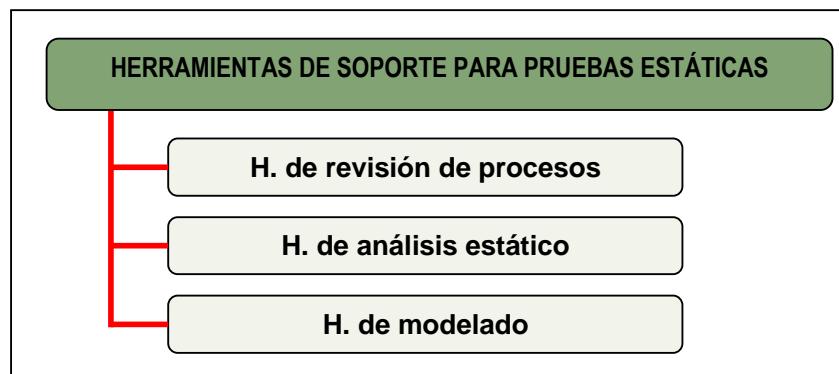
a) Herramientas de gestión de pruebas

Estas herramientas ayudan a recoger, organizar y comunicar información sobre las pruebas en un proyecto. Esta información puede usarse para monitorear el proceso de pruebas y decidir las acciones a tomar. Asimismo, estas herramientas también proporcionan información sobre el componente o sistema que es probado.

- b) Herramientas de gestión de requisitos  
Son útiles para las pruebas dado que éstas se basan en los requisitos. De esta forma, cuanto mayor sea la calidad de los requisitos, más fácil será desarrollar pruebas para comprobar que son correctos. Asimismo, es importante mantener una trazabilidad bidireccional entre los requisitos y las pruebas, y estas herramientas pueden ser de gran ayuda en esta tarea.
- c) Herramientas de gestión de incidencias  
Facilitan el mantenimiento del registro de incidencias a lo largo del tiempo. Este tipo de herramientas permiten identificar defectos, problemas, anomalías o posibles mejoras.
- d) Herramientas de gestión de configuración  
Las herramientas de gestión de configuración no son estrictamente herramientas de pruebas, sin embargo, una buena gestión de configuración es crítica para controlar las pruebas. Es necesario conocer exactamente qué debe ser probado así como la versión exacta de todos los componentes del sistema.

#### 1.2.4.2. Herramientas de soporte para pruebas estáticas

Las herramientas de las que se habla en este apartado pueden dar soporte a las actividades de prueba descritas en el apartado relativo a las pruebas estáticas. Esta área abarca tres tipos de pruebas, a continuación se describen.



**Figura 1.16. Herramientas de soporte para pruebas estáticas**

- a) Herramientas de revisión de procesos  
Las herramientas de revisión de procesos se hacen especialmente útiles para revisiones formales, donde hay mucha gente involucrada o donde las personas implicadas están en lugares geográficamente separados.

Es cierto que mediante el uso de hojas de cálculo o documentos de texto se puede llevar un seguimiento de toda la información del proceso de revisión, pero también hay que recordar que una herramienta de revisión está diseñada exclusivamente para este propósito, y por lo tanto

será más probable conseguir un buen resultado utilizándola. Otra ventaja que ofrece este tipo de herramientas es que facilitan mucho el trabajo de recogida de información del proceso de pruebas.

b) Herramientas de análisis estático

Las herramientas de análisis estático, normalmente, son utilizadas por desarrolladores como parte del proceso de pruebas. La característica principal de estas herramientas es que el código no se ejecuta, sino que sirve de elemento o dato de entrada a la herramienta. Algunas de las características de estas herramientas son que calculan métricas (p.ej., complejidad ciclomática), imponen estándares de codificación, analizan estructuras y dependencias, identifican anomalías en el código, entre otras.

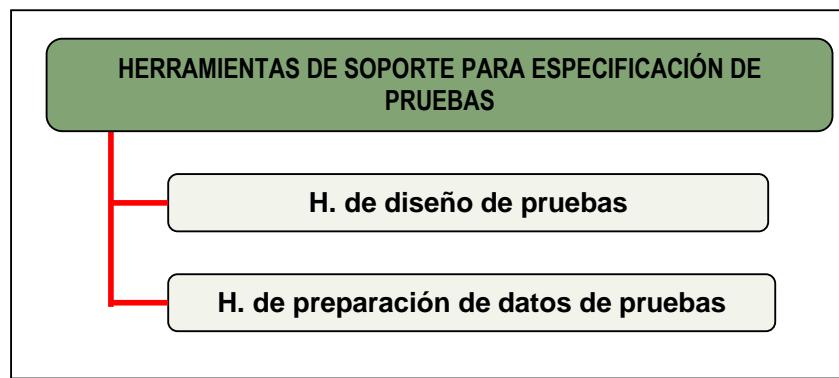
c) Herramientas de modelado

Ayudan a validar modelos del sistema, por ejemplo, validando la consistencia de los objetos en una base de datos o encontrando inconsistencias y defectos. Estas herramientas son de gran utilidad en el diseño de software.

Una ventaja que tienen tanto las herramientas de análisis estático como las de modelado es que pueden utilizarse antes de la ejecución de las pruebas dinámicas. Esto permite detectar defectos tan pronto como sea posible, lo que supone que arreglarlos sea más sencillo y barato.

#### 1.2.4.3. Herramientas de soporte para la especificación de pruebas

Estas herramientas dan soporte a actividades de pruebas tales como el análisis de pruebas, el diseño de pruebas o la implementación de pruebas. En este apartado destacamos dos tipos de esta categoría de herramientas.



**Figura 1.17. Herramientas de soporte para especificación de pruebas**

a) Herramientas de diseño de pruebas

Estas son de gran utilidad a la hora de comenzar con el diseño de pruebas, aunque no harán todo el trabajo. El beneficio que ofrece este tipo de herramientas es que pueden identificar de manera rápida y sencilla las pruebas a

ejecutar sobre todos los elementos del sistema. Es decir, ayuda a que el proceso de pruebas sea más exhaustivo.

Las herramientas de diseño de pruebas también pueden ayudar a identificar valores de entradas a las pruebas (requisitos, modelos de diseño, condiciones de prueba...), a construir casos de pruebas, a seleccionar los factores a tener en cuenta para asegurar que todos los pares de combinación son probados, etc.

- b) Herramientas de preparación de datos de pruebas  
Establecer datos de pruebas puede ser una tarea tediosa, especialmente si se contempla un gran volumen de datos para las pruebas. Las herramientas de preparación de datos de prueba sirven de ayuda en esta área. Estas herramientas suelen usarse por los desarrolladores, aunque también se utilizan durante las pruebas de sistema y de aceptación. Son especialmente útiles en las pruebas de rendimiento, donde es imprescindible trabajar con gran cantidad de datos realistas.

#### 1.2.4.4. Herramientas para ejecución y registro de pruebas

En este apartado, se van a definir tres tipos de herramientas relacionadas con la ejecución y registro de pruebas.

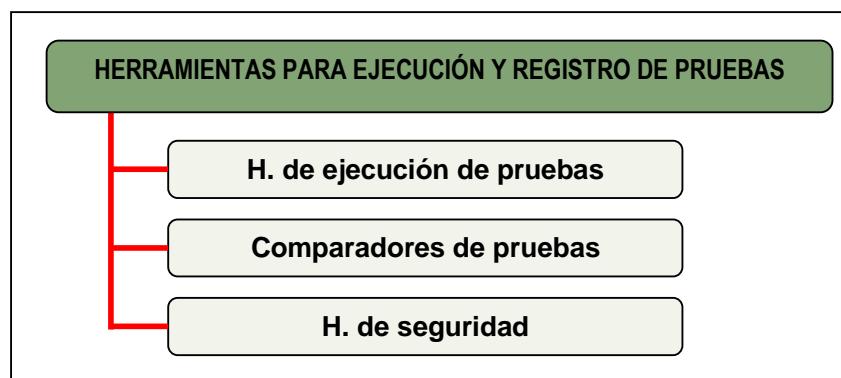


Figura 1.18. Herramientas para ejecución y registro de pruebas

- a) Herramientas de ejecución de pruebas  
La mayoría de las herramientas de este tipo ofrecen un mecanismo para la captura y registro de pruebas.  
  
Estas herramientas suelen utilizar un lenguaje de scripting, por lo que si los técnicos de pruebas desean utilizar una herramienta de ejecución de pruebas necesitarán tener habilidades de programación sobre la creación y modificación de scripts.
- b) Comparadores de pruebas  
La esencia de las pruebas es comprobar si el software produce el resultado correcto, y para ello deben comparar lo que el software produce y lo que debería producir. Los comparadores de pruebas ayudan a automatizar estos aspectos. Hay dos maneras de llevar a cabo esta

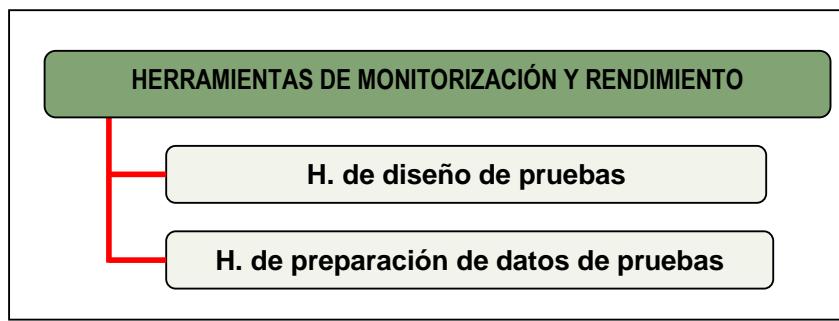
comparación. De manera dinámica, donde la comparación es realizada dinámicamente, es decir, mientras la prueba se está ejecutando, o post- ejecución, donde la comparación se realiza después de que la prueba haya acabado y el software bajo prueba no se vuelva a ejecutar.

c) Herramientas de seguridad

Hay numerosas herramientas que protegen a los sistemas de ataques externos, por ejemplo, los cortafuegos. Las herramientas de pruebas de seguridad se utilizan para probar la seguridad que tienen los sistemas, intentando acceder a un sistema, por ejemplo. También, se encargan de identificar virus, simular ataques externos, detectar intrusos, etc.

#### 1.2.4.5. Herramientas de monitorización y rendimiento

Las herramientas descritas en esta sección soportan pruebas que pueden llevarse a cabo durante el propio proceso de pruebas o después de la liberación del sistema. En esta sección se tratarán dos de estas herramientas.



**Figura 1.19. Herramientas de soporte para especificación de pruebas**

a) Herramientas de análisis dinámico

Las herramientas de análisis dinámico se llaman así porque son “dinámicas”, es decir, requieren que el código se esté ejecutando, y porque no ejecutan pruebas como tal, sino que analizan lo que ocurre cuando el software se está ejecutando.

b) Herramientas de monitorización

Las herramientas de monitorización se utilizan para realizar un seguimiento del estado del sistema en uso, con el objetivo de conseguir avisos tempranos de los problemas y de mejorar el servicio. Hay herramientas de monitorización para servidores, redes, bases de datos, seguridad, rendimiento, páginas web y uso de internet, y para aplicaciones.

# Actividades

## CASO 1: CASO DE USO GENERAR TRANSFERENCIA A OTROS BANCOS

A continuación, se muestra el prototipo del caso de uso “Generar Transferencia a otros Banco” del sistema de Banca por Internet. Revise la pantalla teniendo en cuenta las siguientes consideraciones:

**Transferencias a otros bancos**

*Ingreso de datos* Paso 1 de 4

Todas las solicitudes de las transferencias a otros bancos se atienden en un plazo máximo de dos días útiles. La atención de transferencias a otros bancos es de Lunes a Viernes.

¿Desde qué cuenta deseas transferir?

Ahorro soles nº 193-17550713-0-92 (S/. 3,919.94)

¿En qué moneda y cuánto vas a transferir?

Moneda	Monto
Soles	

Ingrera los datos de tu transferencia

Banco destino	Tipo de Transferencia
selecciona banco	a cuenta de terceros

Código interbancario de la cuenta destino

Nombre del ordenante

(Si deseas, modifica el nombre del ordenante)

Datos opcionales del beneficiario (persona a la que pertenece la cuenta destino de la transferencia)

Nombre	Apellido Paterno	Apellido Materno

Tipo de documento Número de documento

DNI

Para que un usuario pueda realizar una transferencia a otro banco deberá:

- Seleccionar la cuenta desde la cual realizará la transferencia. Aquí se mostrará una lista de todas las cuentas que tiene el usuario.
- Seleccionar el tipo de moneda, que podrá ser soles o dólares.
- Ingresar el monto de la transferencia. El monto debe ser un número entero de máximo 4 posiciones. El monto mínimo de transferencia es de 10 nuevos soles o 5 dólares.
- Seleccionar el banco destino que podrán ser: Banco RSSG, Banco AFSG, Banco BAMC, Banco MFS.
- Seleccionar el tipo de transferencia a realizar: A cuenta de Terceros, Pago de Haberes, Pago a Proveedores, Depósitos CTS.

- f. Ingresar el código interbancario, que es un número de 20 posiciones, donde los 2 primeros dígitos pueden ser 00 ó 01
- g. Ingresar el nombre de la persona que realiza la transferencia, en el campo Nombre del ordenante. Es un campo de 30 caracteres.
- h. Podrá además ingresar los datos de la persona que recibirá la transferencia, indicando sus nombres y apellidos. Los campos son de máximo 30 caracteres.

Se pide que usted realice lo siguiente:

1. Identifique los escenarios al caso propuesto.
2. A partir de los escenarios identificados, en la pregunta anterior, deberá identificar los casos de prueba, las condiciones de entrada (**Válido**, **No Válido** o **No Aplica**), para cada uno de los escenarios, y el resultado esperado.
3. Cree una tabla de clases de equivalencias válidas y no válidas.
4. Genere los casos de prueba (indicando un ejemplo del dato usado), considerando como referencia el cuadro generado en la pregunta 2. Utilice las clases de equivalencia creadas en la pregunta 3, indicándolas en cada caso de prueba.

## CASO 2: CASO DE USO AGENDAR ACTIVIDAD

### Flujo Básico (FB)

1. El Administrativo selecciona “Agendar Actividad” desde la interfaz de menú principal.
2. El Sistema despliega las diferentes categorías de actividades: Físicas, Plantel Competitivo, Deportivas, Culturales.
3. El Administrativo selecciona una categoría distinta de Físicas
4. El sistema muestra las actividades para esa categoría
5. El Administrativo selecciona una actividad
6. El Sistema muestra los grupos y rangos de edad al cual puede estar destinada la actividad:
  - Grupo 1: Niños y adolescentes
    - Preescolares (2.5 a 5 años)
    - Escolar Inicial (6 a 8 años)
    - Escolar Final (9 a 11 años)
    - Preadolescente (12 a 14 años)
  - Grupo 2: Jóvenes y adultos (> 14 años)
7. El Administrativo selecciona uno de esos rangos.
8. El sistema solicita el período en que se realizará la actividad
9. El Administrativo ingresa la fecha de inicio y la fecha de fin del período de la actividad
10. El sistema solicita horario en que se realiza la actividad
11. El Administrativo ingresa horario
12. El Sistema despliega los profesores de carácter permanente asociados a la actividad seleccionada disponibles en el horario, período y franja de edad especificados.
13. El Administrativo selecciona un profesor.
14. El Sistema despliega los lugares disponibles para realizar la actividad seleccionada en el horario y período indicados.
15. El Administrativo selecciona el lugar.
16. El Administrativo selecciona la opción registrar la actividad.
17. El Sistema registra la información ingresada por el administrativo en el cronograma de actividades del club y muestra el mensaje de “Registro exitoso”.
18. El Caso de Uso finaliza.

### Flujos Alternos

#### FA1. El administrativo selecciona la actividad “Físicas”

- 3.1 En el paso 3 del FB si el administrativo selecciona la categoría “Físicas”, el sistema despliega como actividad solamente Físicas y como rango de edad solamente el Grupo 2.
- 3.2 El caso de uso continúa en el paso 8 del FB

#### FA2. No hay profesores disponibles

- 12.1. En el paso 12 del FB si el sistema detecta que no hay profesores disponibles, el sistema muestra el mensaje “No hay profesores disponibles”.
- 12.2. El Caso de Uso finaliza.

#### FA3. No hay lugares disponibles

- 14.1. En el paso 14 del FB si el sistema detecta que no hay lugares disponibles, el sistema muestra el mensaje “No hay lugares disponibles”.
- 14.2. El Caso de Uso finaliza.

Se pide que usted desarrolle los siguientes puntos:

1. Identifique los escenarios correspondientes al CU “Agendar Actividad”.
2. A partir de los escenarios identificados en la pregunta anterior, deberá identificar el tipo de valor de las condiciones de entrada (Válido, Inválido o No Aplica) para cada uno de los escenarios y el resultado esperado.
3. Identifique las clases de equivalencia válidas y no válidas para las condiciones de entrada
4. Identifique los distintos casos de prueba sólo para los escenarios y condiciones de entrada relacionados con el Flujo Básico del Caso de Uso “Agendar Actividad”. Complemente los casos de prueba, de ser necesario, con aquellos que se generen luego de identificar las clases de equivalencia de la pregunta 3.

## Resumen

- **La validación y la verificación** son procesos de evaluación de productos que son útiles para determinar si se satisfacen las necesidades del negocio y si se están construyendo acorde a las especificaciones.
- Las pruebas están más relacionadas con el proceso de validación, mientras que las revisiones son tareas más orientadas al proceso de verificación.
- La **Estrategia de Prueba** de software integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba: la planificación, el diseño de casos de prueba, la ejecución y los resultados, tomando en consideración cuánto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado una correcta construcción del software.
- En RUP se definen 4 roles de pruebas: Administrador de pruebas, Analista de pruebas, Diseñador de pruebas y Ejecutor de pruebas
- Las pruebas estáticas y las pruebas dinámicas son métodos complementarios ya que ambas tratan de detectar distintos tipos de defectos de forma efectiva y eficiente, aunque las pruebas estáticas están orientadas a encontrar defectos mientras que las dinámicas fallos.
- El diseño de los casos de prueba consiste realizar los siguientes pasos:
  - Definir escenarios
  - Identificar condiciones de entrada
  - Definir clases de equivalencia
  - Realizar casos de prueba
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - ☞ [http://www.inteco.es/file/XaXZyrAaEYfaXKiMJlkT\\_g](http://www.inteco.es/file/XaXZyrAaEYfaXKiMJlkT_g)  
Aquí encontrará una guía completa de Validación y Verificación orientada a las áreas de proceso de CMMI®. Esta guía fue elaborado por INTECO (Instituto Nacional de Tecnologías de la Comunicación) de España.
  - ☞ [http://epf.eclipse.org/wikis/openupsp/openup\\_basic/disciplines/test,\\_0TkKQMIgEdmt3adZL5Dmdw.html](http://epf.eclipse.org/wikis/openupsp/openup_basic/disciplines/test,_0TkKQMIgEdmt3adZL5Dmdw.html)  
Aquí hallará mayor información de la disciplina de pruebas según RUP.





## FUNDAMENTOS RATIONAL FUNCTIONAL TESTER

### LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno crea y agrega características avanzadas a los scripts de pruebas funcionales a partir de los casos de prueba de una aplicación Java.

### TEMARIO

#### 2.1. Tema 3: Introducción al Rational Functional Tester

- 2.1.1. Arquitectura de Rational Functional Tester
- 2.1.2. Configuración del entorno de pruebas
- 2.1.3. Configuración de aplicaciones Java a probar
- 2.1.4. Proyectos de pruebas funcionales en Rational Functional Tester

#### 2.2. Tema 2: Script de pruebas funcionales

- 2.2.1. Grabación de un script
- 2.2.2. Reproducción de un script
- 2.2.3. Revisión de los resultados
- 2.2.4. Características avanzadas de script de pruebas

### ACTIVIDADES PROPUESTAS

- Los alumnos crean script de pruebas funcionales para una aplicación de escritorio Java.
- Los alumnos crean script de pruebas funcionales para una aplicación web Java.
- Los alumnos agregan características avanzadas a los script de pruebas funcionales para una aplicación web Java.

## 2.1. INTRODUCCIÓN AL RATIONAL FUNCTIONAL TESTER

IBM *Rational Functional Tester* (RFT) es una herramienta automatizada para la realización de pruebas funcionales y de regresión. Permite probar aplicaciones Java, .NET y basadas en Web.

Con la **tecnología de grabación**, se puede generar scripts mediante la ejecución y el uso de la aplicación bajo prueba. Los scripts del RFT son implementados como programas Java. Asimismo, crea una vista que muestra las acciones del usuario sobre la aplicación.

Durante el proceso de grabación, se puede añadir **comandos controlados por datos** que permitirán probar, durante el proceso de reproducción, otros datos provenientes de un **pool de datos**. Otra característica importante del RFT que pueden ser añadidos en un script, son los **puntos de verificación** que permiten evaluar los datos y propiedades de los objetos de la aplicación y confirmar el estado de dichos objetos probándolas aún en versiones posteriores. Esto es posible gracias a la **tecnología ScriptAssure** que permite crear scripts de prueba más resistentes a los cambios en los objetos de las aplicaciones. Los objetos que son referenciados por las aplicaciones son almacenados automáticamente en un **mapa de objetos de prueba** al crear el script.

La siguiente figura muestra la perspectiva ***Functional Test*** que se utilizará en el RFT:

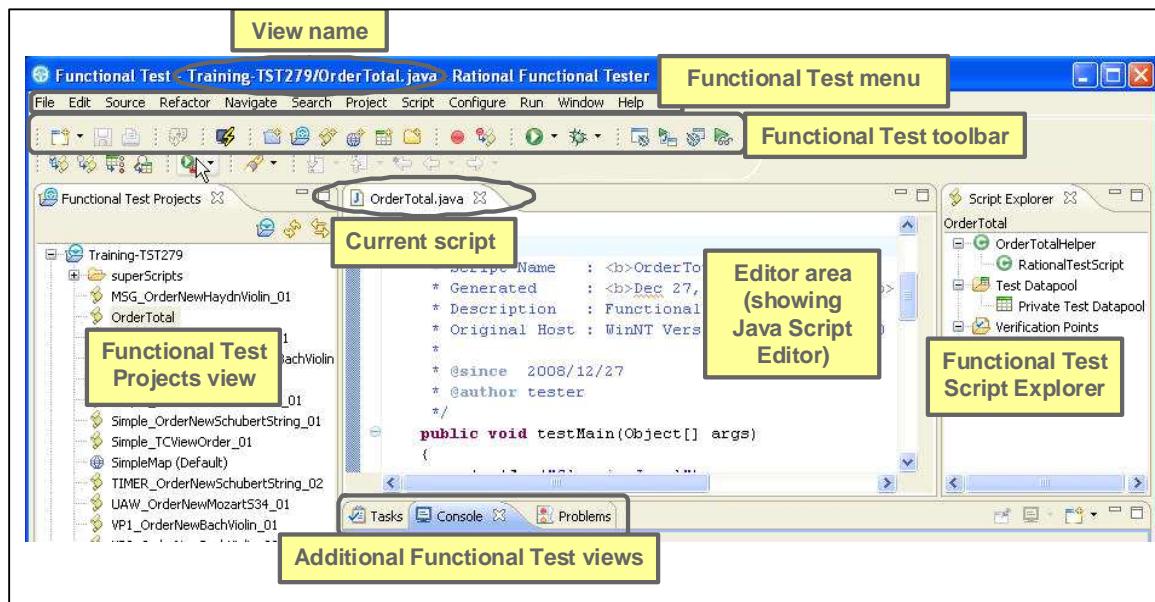


Figura 2.1. Perspectiva ***Functional Test***

### 2.1.1. Arquitectura de Rational Functional Tester

Este apartado introduce una visión general de la arquitectura del RFT el cual es descrito con más detalle en temas posteriores.

#### 2.1.1.1. Almacenamiento de activos de pruebas.

RFT es una herramienta de escritorio basada en archivos y no necesita trabajar con un servidor de componentes. Los activos de pruebas que se almacenan en el RFT son principalmente archivos de

Java y XML. Se puede utilizar una herramienta de gestión de configuración como *Rational Team Concert*, *Rational Clear Case* o herramientas como CVS para el control individual de versiones de los activos de pruebas.

Si se utiliza RFT *Java Scripting*, los proyectos de pruebas funcionales son una clase de proyectos Java de Eclipse™. Si se utiliza RFT *Visual Basic .NET Scripting*, los protectos de pruebas funcionales son una clase de proyectos de Visual Basic de Microsoft Visual Studio®. Ambos son proyectos creados en el entorno del RFT.

#### 2.1.1.2. Almacenamiento de resultados de pruebas.

Los resultados son almacenados en archivos conocidos como *test logs* dentro del IBM RFT, los cuales pueden ser grabados en muchos formatos diferentes que pueden ser configurados, inclusive en formatos que podrán ser utilizados en herramientas de gestión de pruebas tales como *Rational Quality Manager*. Los formatos que podrá configurarse en el IBM RFT para los *test logs* son los siguientes:

- HTML
- Texto
- Plataforma de herramientas de pruebas y rendimiento
- XML
- *Rational Quality Manager*

#### 2.1.1.3. Grabación de pruebas funcionales.

Es probable que utilice las opciones de grabación del RFT para crear nuevos scripts de prueba, pues es la manera más rápida y sencilla de generar líneas de código, aún si los scripts son modificados posteriormente. Durante el proceso de grabación debe asegurarse de no realizar acciones que no deben ser parte de las pruebas.

Los scripts de pruebas contienen información adicional aparte del archivo que contiene las acciones del usuario interactuando con el aplicativo bajo prueba y del script generado con código Java. La información adicional es conocida como activos de pruebas: objetos de pruebas, puntos de verificación y datos de prueba.

#### 2.1.1.4. Ejecución de pruebas.

Las pruebas grabadas son ejecutadas en modo de reproducción. En este modo, el RFT envía todas las acciones del mouse y el teclado que se graban a la aplicación bajo prueba. En general, no se debe manipular el teclado o el mouse cuando el RFT está en modo de reproducción. Sin embargo, a veces, puede ejecutar las pruebas en modo interactivo para manipular los posibles problemas de reproducción.

Un script de prueba se compone en gran parte de instrucciones de interacción, incluyendo la realización de pruebas con varios objetos de la aplicación bajo prueba. Cuando se ejecuta un script de prueba, RFT primero tiene que identificar y encontrar cada objeto, haciendo coincidir las propiedades de reconocimiento en el mapa de objetos de prueba grabados en el script con los objetos actuales que están

presentes en la aplicación que está en ejecución. Si RFT no puede encontrar un objeto que coincida con el que se encuentra en el script grabado, se registra un error con los intentos de continuar o se anula. En cambio, si RFT encuentra un objeto que coincide con las propiedades grabadas en el script, se realiza la acción sobre el objeto. Estas acciones podrían ser las interacciones del usuario, como hacer un clic, hacer una selección, u otras operaciones, tales como obtener o establecer valores sobre el objeto. Por último, las acciones realizadas en el objeto podría ser un punto de verificación, en cuyo caso RFT compara el valor esperado o un intervalo de valores con el resultado real obtenido en tiempo de ejecución.

Puede reproducir una prueba en la misma máquina o en cualquier otra máquina que ejecuta *Rational Agent*, que se instala por defecto con RFT. También puede ejecutar varias pruebas en varios equipos remotos para pruebas funcionales distribuidos. Esto hace que sea posible realizar muchas más pruebas en un corto periodo de tiempo. Una máquina dada sólo puede ejecutar una prueba a la vez, o muchas pruebas en forma secuencial. También, puede ejecutar las pruebas de RFT en máquinas remotas usando las herramientas de administración de pruebas, tales como *Rational Quality Manager*.

#### 2.1.1.5. Integración con otras aplicaciones.

RFT es un producto *stand alone* que no requiere de otras herramientas o aplicaciones. Sin embargo, de modo opcional, puede integrarse con otros tipos de aplicaciones según las necesidades:

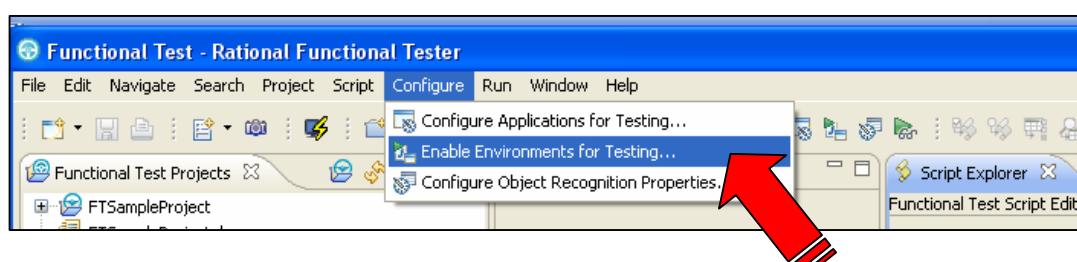
- Para gestión de pruebas o de calidad con *IBM Rational Quality Manager*
- Para gestión de solicitudes de cambios o seguimiento de defectos, *IBM Rational ClearQuest*
- Para gestión de configuraciones o control de versiones, *IBM Rational Team Concert* e *IBM Rational ClearCase*
- Para realizar pruebas unitarias o utilizar herramientas de desarrollo de pruebas, *JUnit*
- Para la generación de pruebas automatizadas, *IBM Rational Performance Tester*
- Herramientas de desarrollo, *IBM Rational Application Developer*

#### **2.1.2. Configuración del entorno de pruebas**

Las tareas que debe efectuar para configurar el entorno para pruebas funcionales son:

- Configure los navegadores web.
- Configure los entornos de Java.
- Habilite plataformas de eclipse.

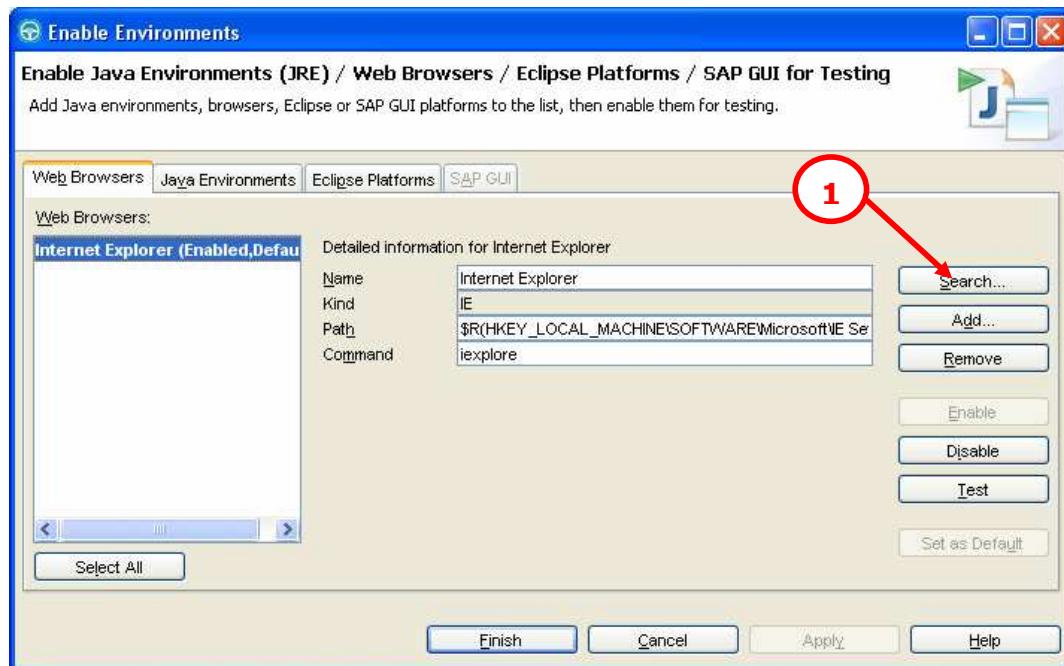
Las tres configuraciones se realiza a partir de la opción **Configure > Enable Environments for Testing** desde la barra de menú del RFT.



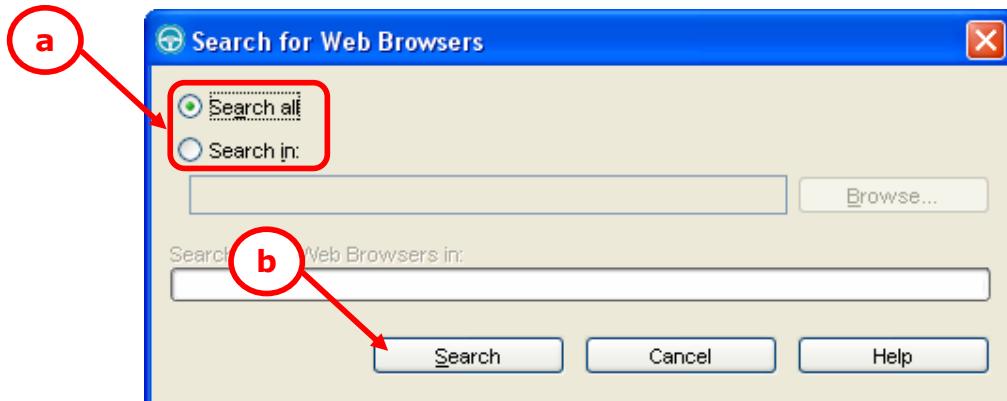
### 2.1.2.1. Configuración de navegadores web.

Por defecto, al instalar RFT se habilita el navegador web *Internet Explorer*, pero si desea configurar otro navegador que instala posteriormente, debe realizar los pasos que se describen a continuación:

1. Desde la pestaña **Web Browsers** seleccione la opción **Search**.



2. A continuación, se abrirá el cuadro de diálogo “Buscar navegadores Web”.

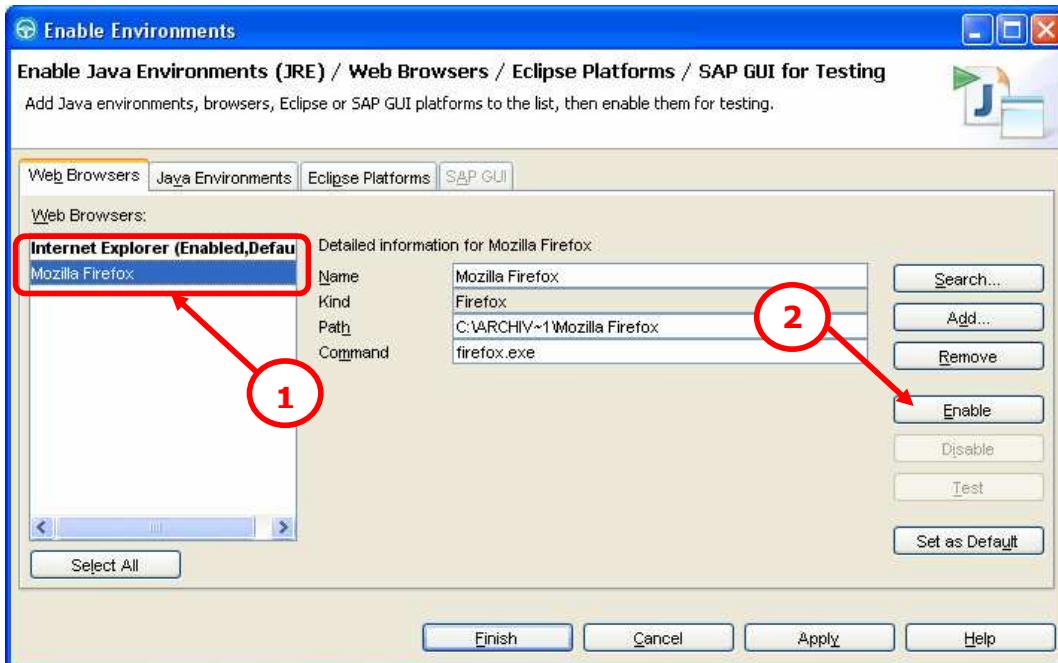


- Selecione uno de los mecanismos de búsqueda siguientes.
  - **Search all** (Buscar todo) para que el habilitador encuentre todos los navegadores del sistema. RFT explorará todos los discos duros o particiones, y enumerará los navegadores en la lista “Navegadores web” de la ventana anterior.
  - **Search in** (Buscar en) para situarse y buscar en una unidad o directorio raíz específicos.

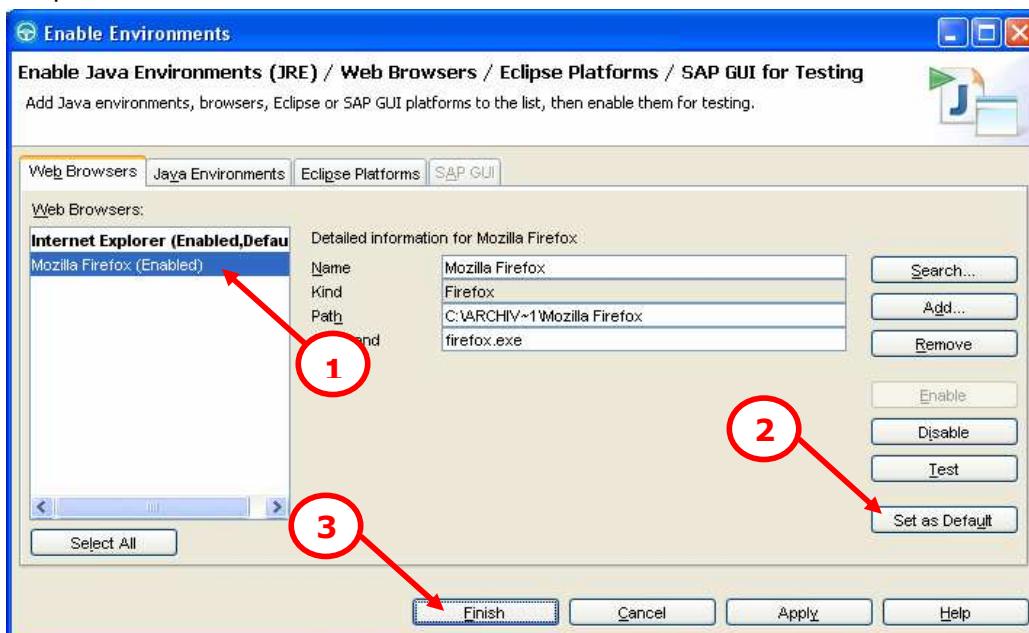
**Nota:** No debe utilizar la opción "Buscar todo" para buscar navegadores en sistemas Linux o UNIX. En lugar de esta opción, utilice la opción "Buscar en" y búsqüelos.

b. Pulse el botón **Search**.

3. Seleccione los navegadores que desea habilitar de la lista **Web Browsers**. Puede seleccionar varios navegadores mediante la tecla **ctrl** al efectuar la selección o seleccionar todo si desea habilitar todos los navegadores de la lista mediante el botón **Select All**. Luego, pulse **Enable**.



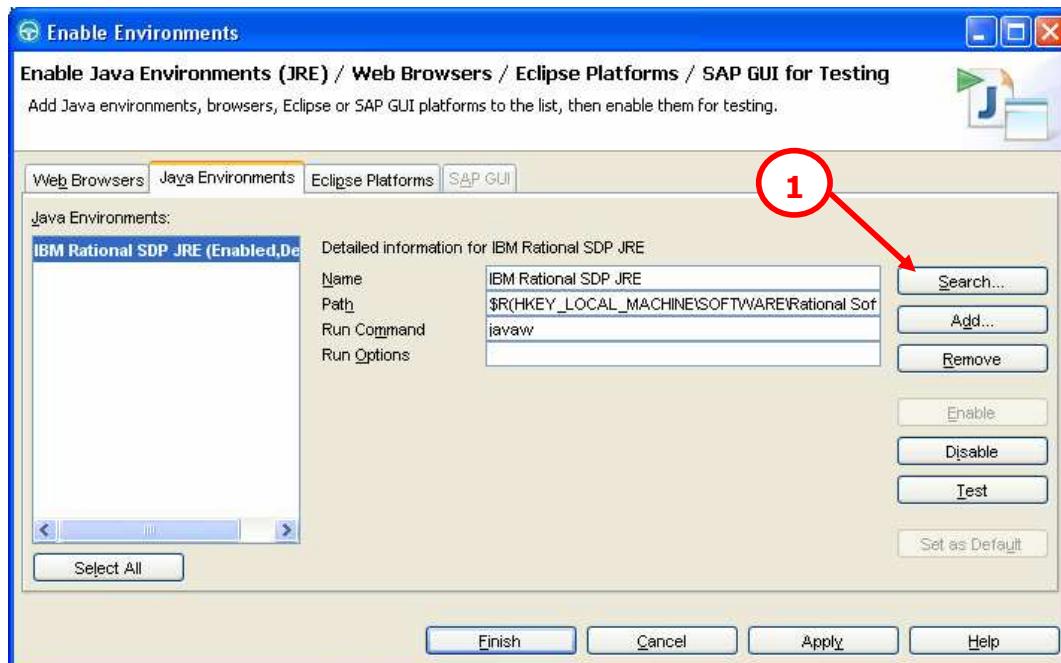
4. Seleccione un navegador como predeterminado y pulse el botón **Set as Default**. Por último, pulse **Apply** para continuar con otra configuración o **Finish** para finalizar.



### 2.1.2.2. Configuración de entornos Java.

Por defecto, al instalar RFT se habilita un entorno JRE, pero si desea configurar otra versión nueva que instala posteriormente, debe realizar los siguientes pasos:

1. Desde la pestaña **Java Environments** seleccione la opción **Search**.



2. A continuación, se abrirá el cuadro de diálogo “Buscar entornos Java”.

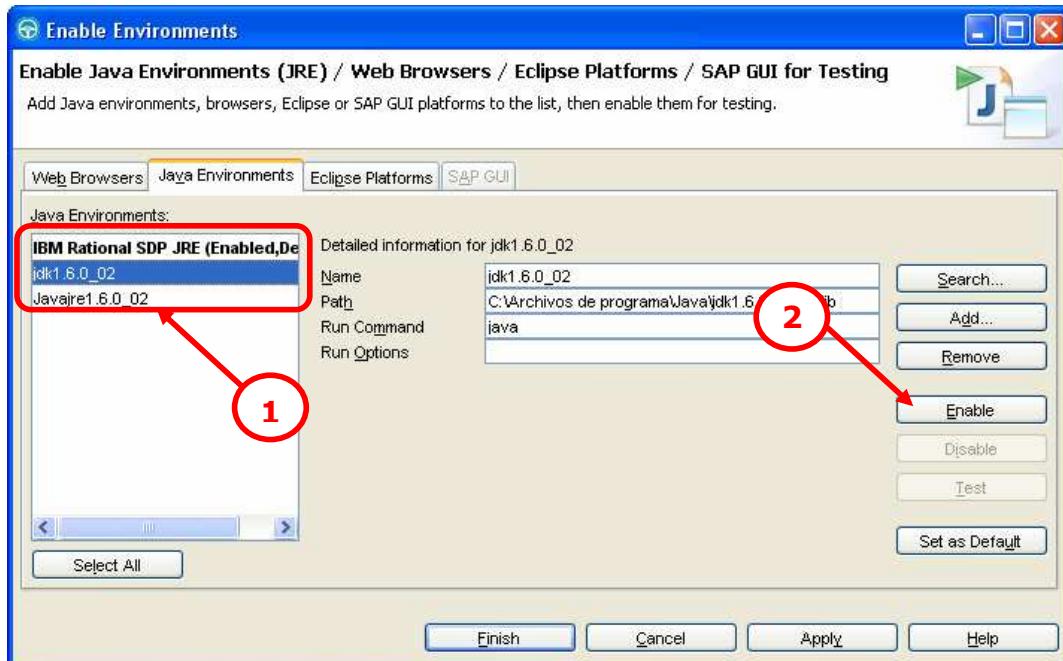


- Selecione uno de los mecanismos de búsqueda siguientes:

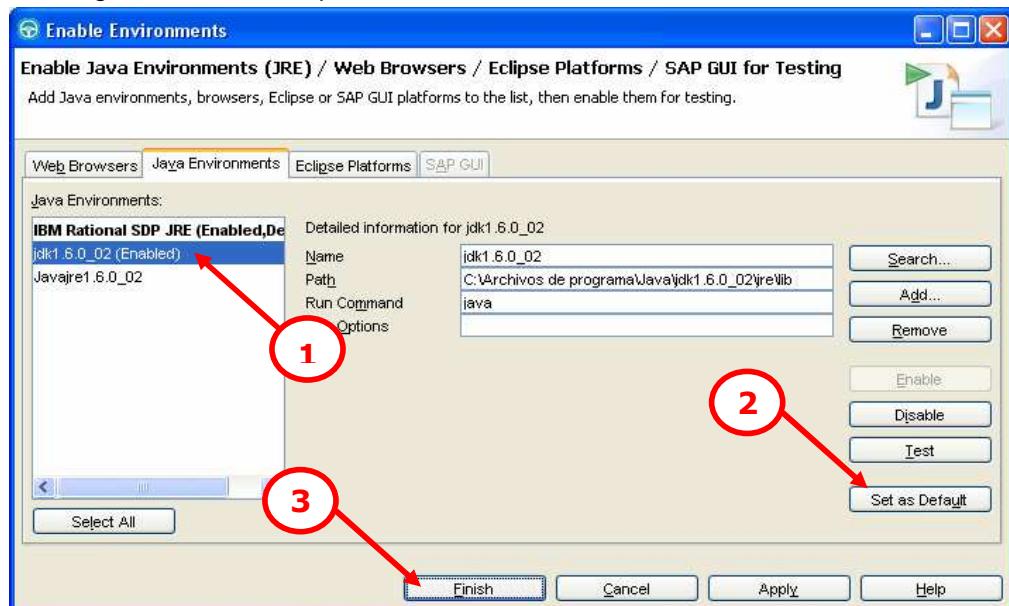
- **Quick search** (Búsqueda rápida), sólo se puede utilizar en sistemas Windows. Éste busca en el registro de Windows todos los entornos de Java y es más rápido que realizar búsquedas en el/los disco(s) duro(s).
- **Search all drives** (Buscar todas las unidades), explora todas las unidades de disco duro o particiones para ubicar todos los entornos de Java del sistema.
- **Search in** (Buscar en) para situarse y buscar en una unidad o directorio raíz específicos.

**Nota:** No debe utilizar la opción "Buscar todas las unidades" para buscar JRE en sistemas Linux. En lugar de esta opción, utilice la opción Buscar en y busque los JRE.

- b. Después de elegir uno de los mecanismos de búsqueda, pulse el botón **Search**.
3. Al terminar la búsqueda, RFT lista los JRE en la lista "Entornos Java" de la pestaña **Java Environments**. Ahora, decida qué entorno desea habilitar. Para ello, seleccione uno, o varios JRE mediante la tecla **Ctrl** al efectuar la selección o pulsar **Select All** si desea habilitar todos los JRE. Luego, pulse **Enable**.



4. A continuación, seleccione un JRE para que sea el valor predeterminado y pulse el botón **Set as Default**. Por último, pulse **Apply** para continuar con otra configuración o **Finish** para finalizar.

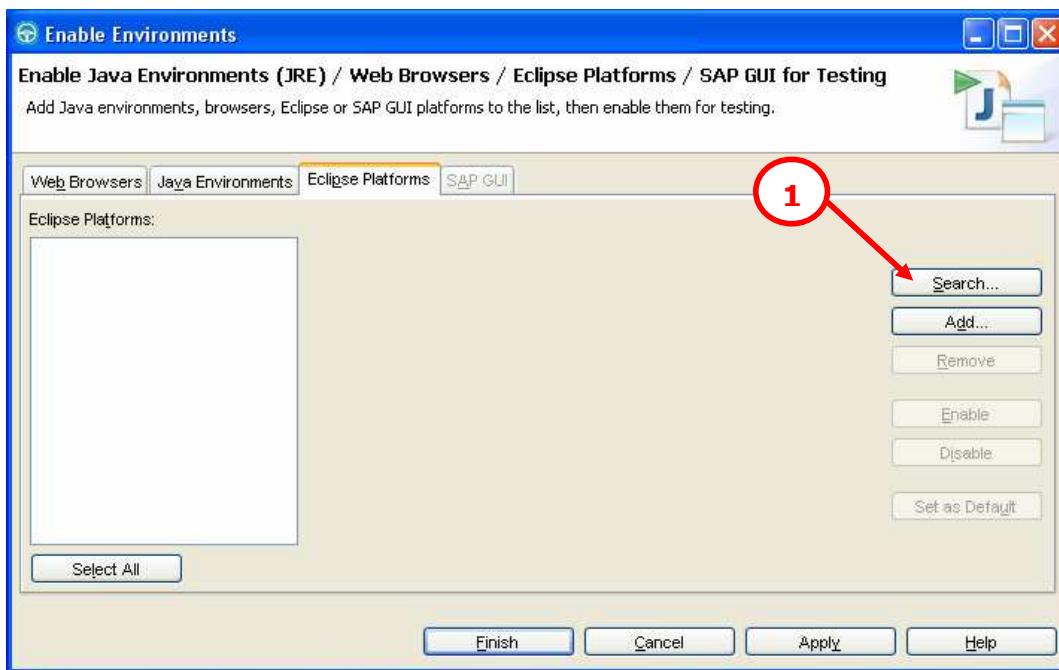


### 2.1.2.3. Habilitación de plataformas de eclipse.

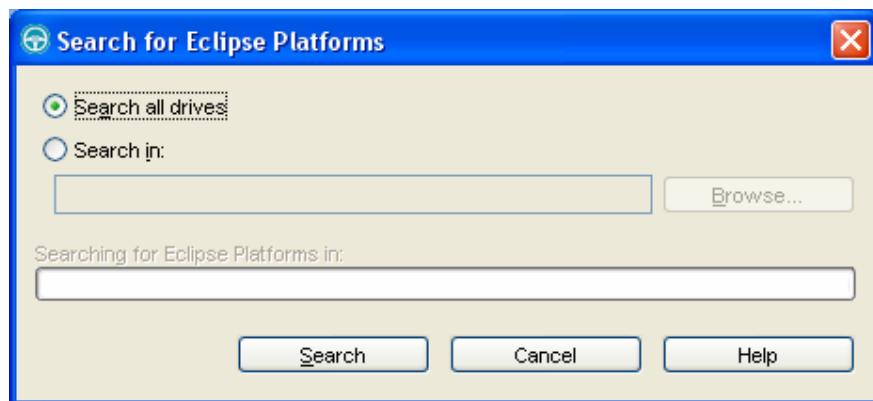
Para habilitar una plataforma de eclipse realice los siguientes pasos:

1. Desde la pestaña **Eclipse Platforms** seleccione la opción **Search**.

**Nota:** Utilice el botón **Add** para ubicar una plataforma de Eclipse y añadirla directamente.

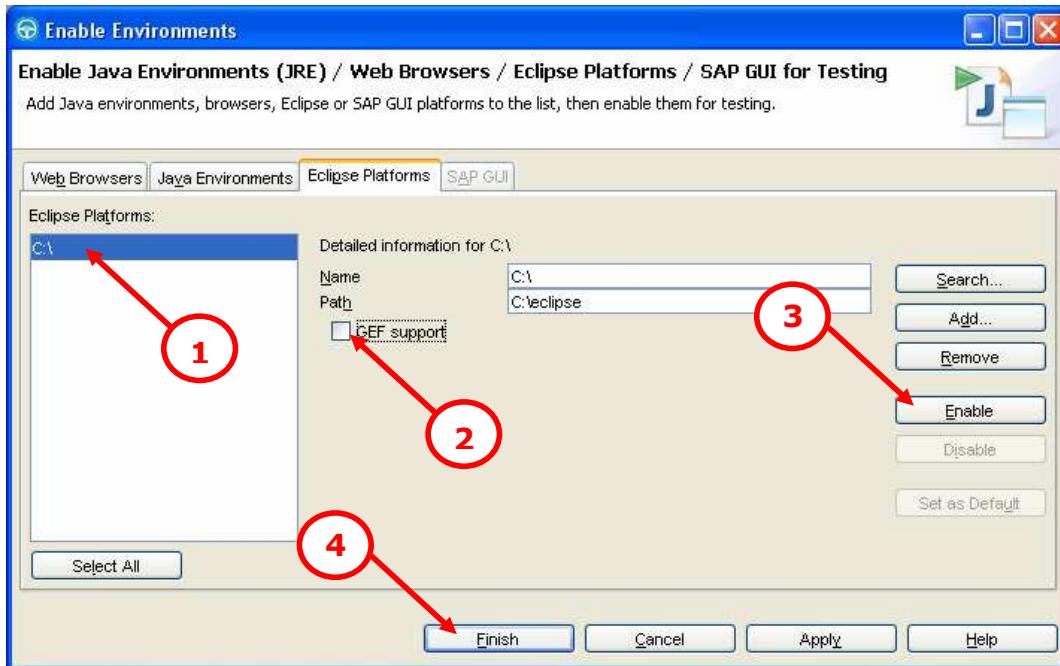


2. De inmediato, se abrirá el cuadro de diálogo “Buscar plataformas de Eclipse”.
  - a. Seleccione uno de los mecanismos de búsqueda siguientes:
    - **Search all drives**, busca las plataformas de Eclipse en todas las unidades de disco.
    - **Search in**, busca la plataforma de Eclipse en un directorio específico.



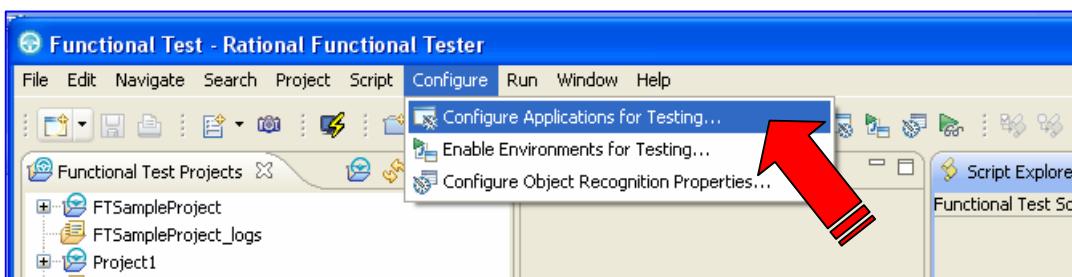
- b. Después de elegir uno de los mecanismos de búsqueda, pulse el botón **Search**.

3. Los resultados de la búsqueda aparecen listados en el panel izquierdo de la pestaña **Eclipse Platforms**. Ahora, seleccione la plataforma de Eclipse que desea habilitar. Luego, para habilitar el soporte para GEF, marque el recuadro de selección Soporte para GEF (el plug-in de habilitación de GEF se copia en el directorio de plug-ins de AUT). A continuación, pulse **Enable**. Por último, pulse **Apply** para continuar con otra configuración o **Finish** para finalizar.



### 2.1.3. Configuración de aplicaciones Java a probar

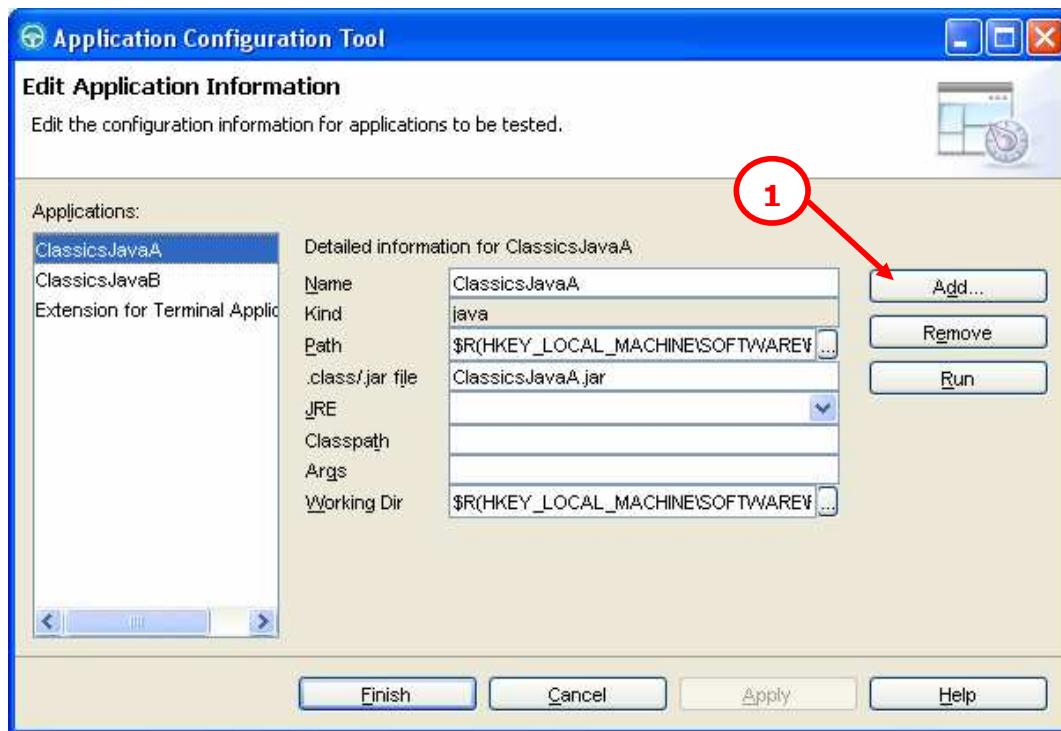
Debe configurar las aplicaciones Java, HTML, VB.NET o Windows para pruebas con RFT a fin de proporcionar el nombre, la vía de acceso y otra información que RFT utiliza para iniciar y ejecutar la aplicación. Para ello, desde la barra de menú principal, seleccione **Configure > Configure Applications for Testing**.



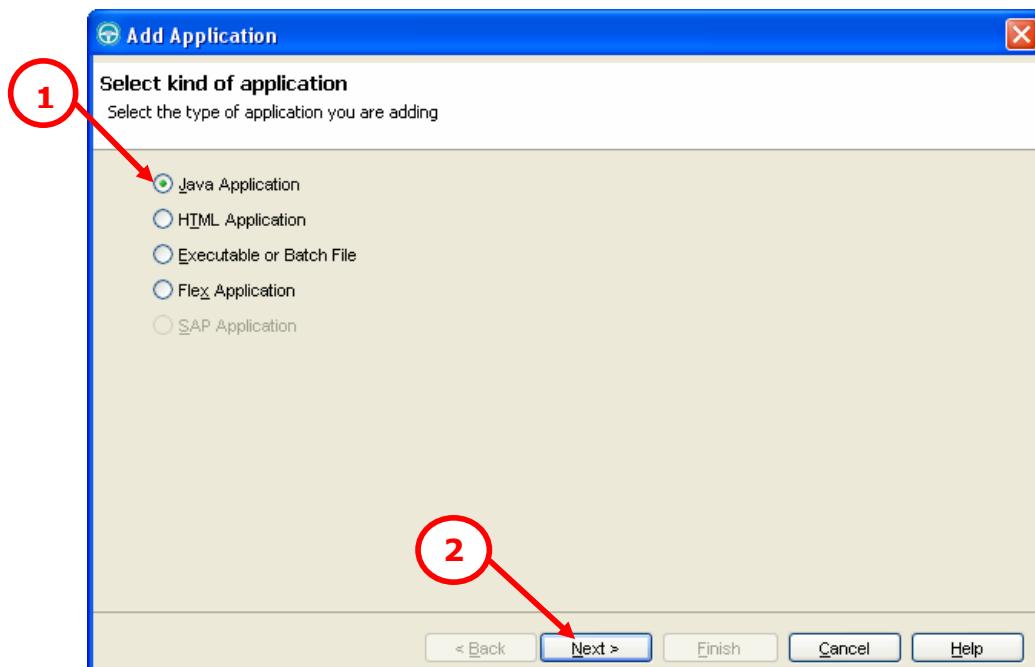
A continuación, como ejercicio agregaremos 2 aplicaciones: una del tipo **Java Application** y otra del tipo **HTML Application**.

### 2.1.3.1. Agregar una aplicación Java.

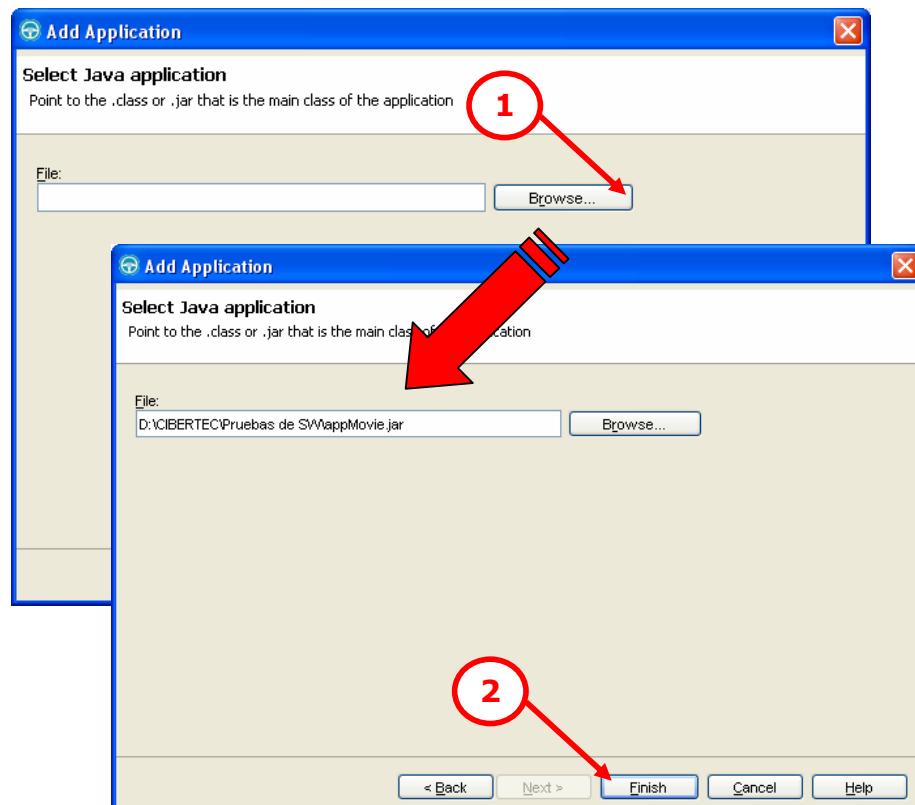
1. Desde la ventana “Herramienta de Configuración de Aplicación” pulse el botón **Add** para añadir una nueva aplicación Java.



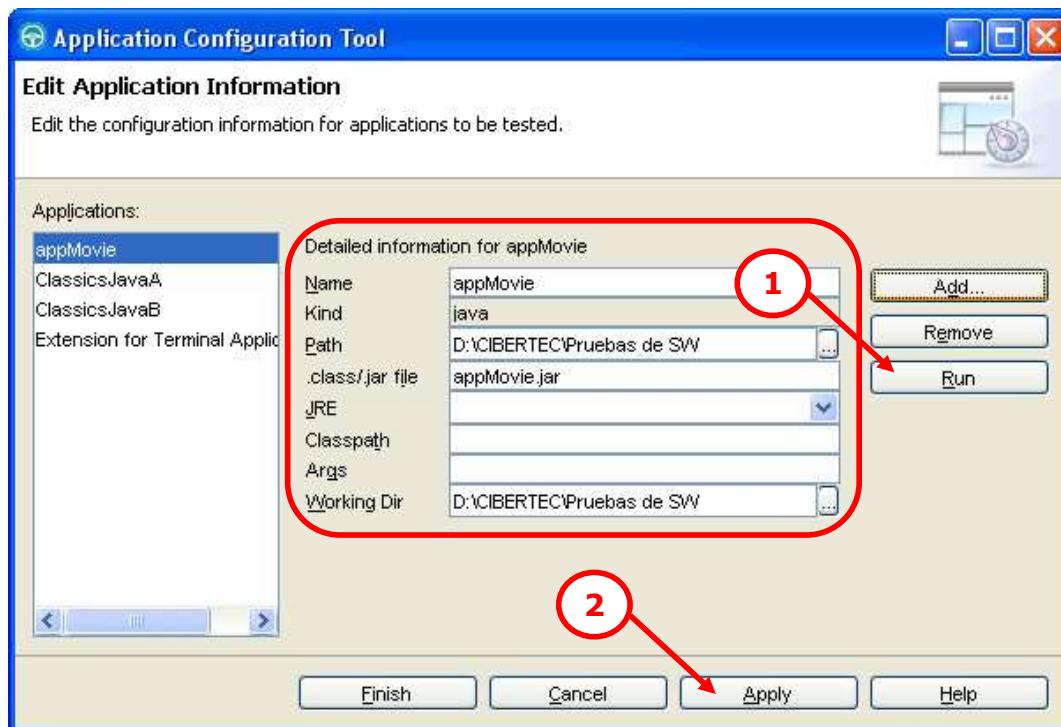
2. En el cuadro de diálogo “Añadir aplicación”, seleccione el tipo de aplicación. En este caso, seleccione **Java Application** y luego pulse **Next**.



3. Pulse **Browse** para ubicar la aplicación y luego pulse **Finish**.

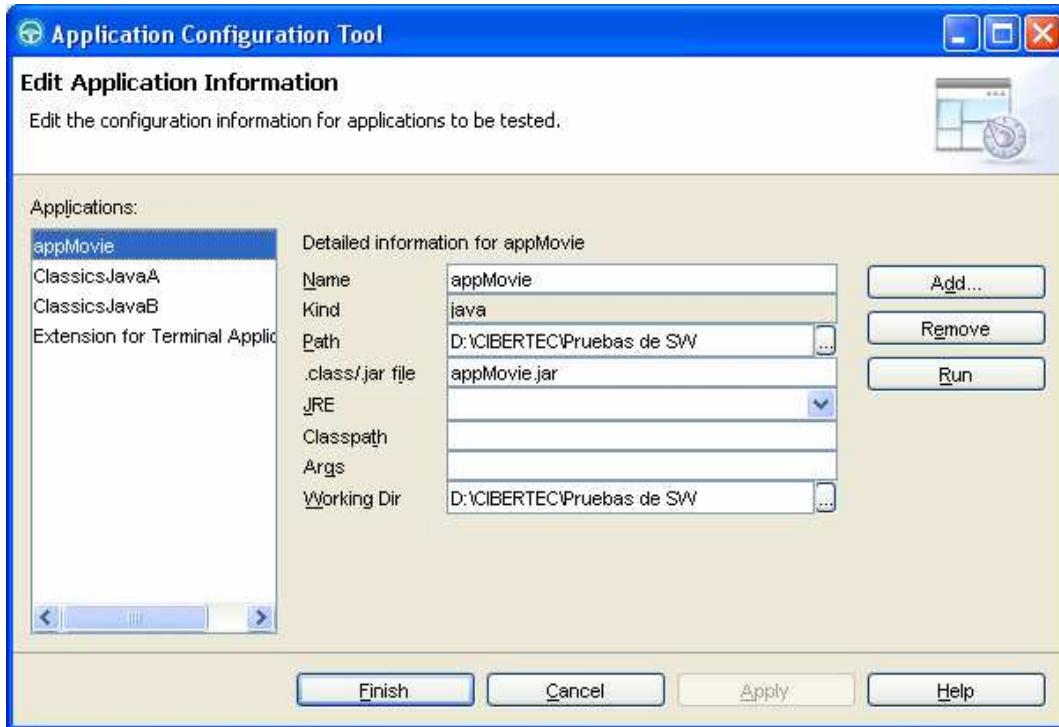


4. A continuación, la nueva aplicación aparecerá en la lista "Aplicaciones" de la ventana "Herramienta de Configuración de Aplicación" y su información detallada en el panel izquierdo (Nombre, Tipo, Vía de acceso, Archivo .class/.jar y Directorio de trabajo). Ahora, pulse **Run** para probar que la aplicación esté bien configurada. Luego, seleccione el botón **Apply**.

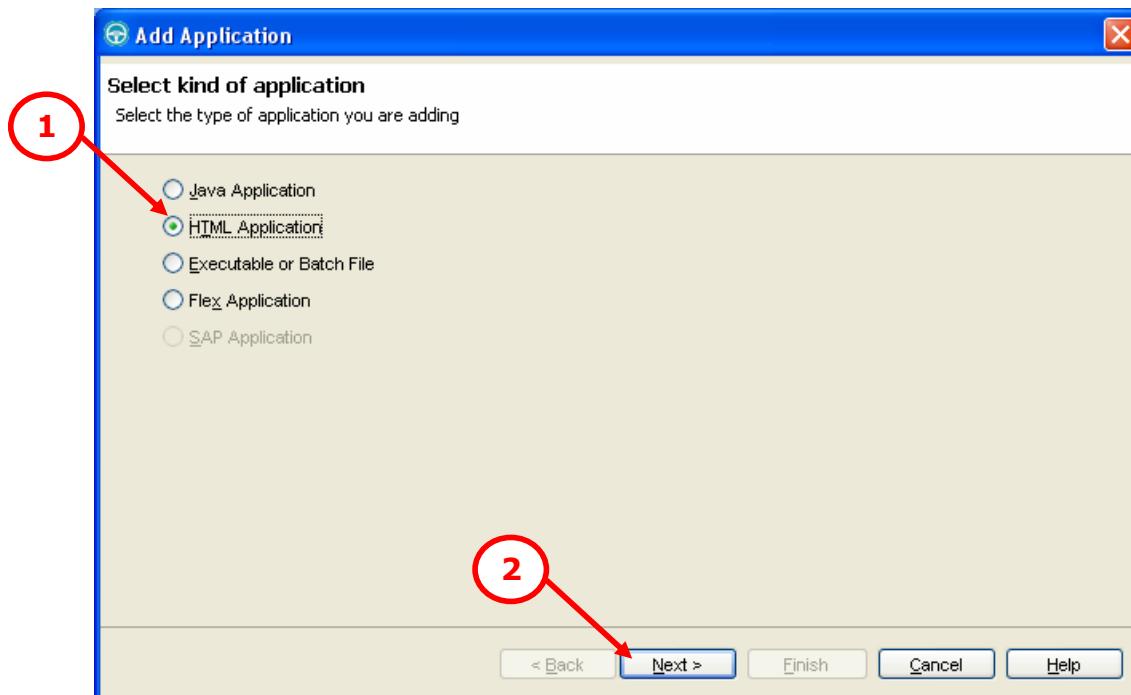


### 2.1.3.2. Agregar una aplicación HTML.

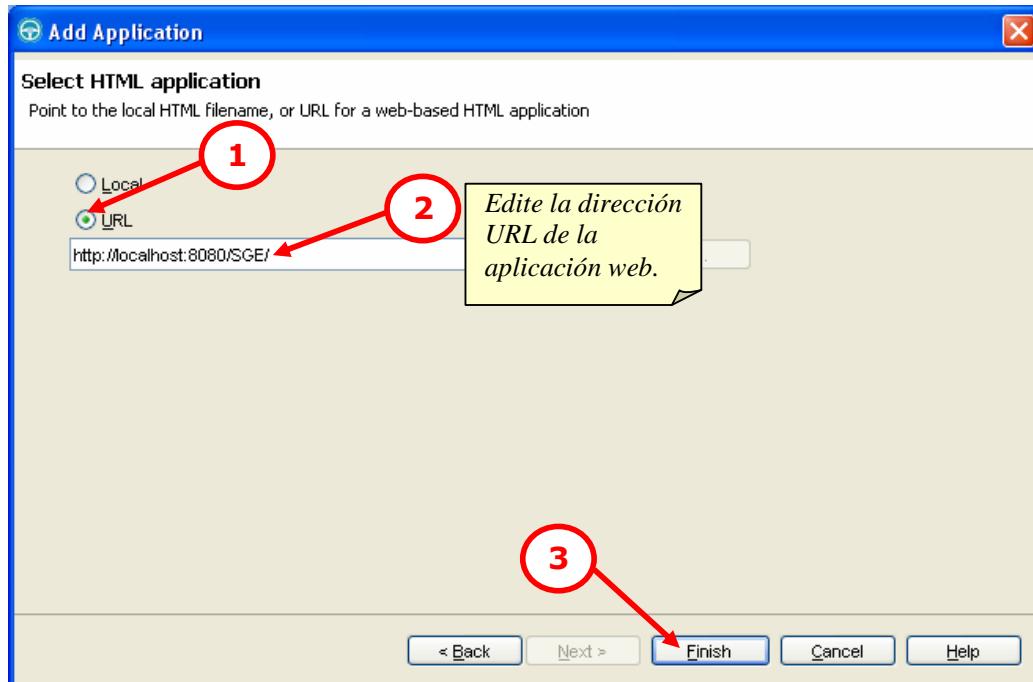
1. Desde la ventana “Herramienta de Configuración de Aplicación” pulse el botón **Add** para añadir una nueva aplicación HTML.



2. En el cuadro de diálogo “Añadir aplicación”, seleccione el tipo de aplicación. En este caso, seleccione **HTML Application** y luego pulse **Next**.

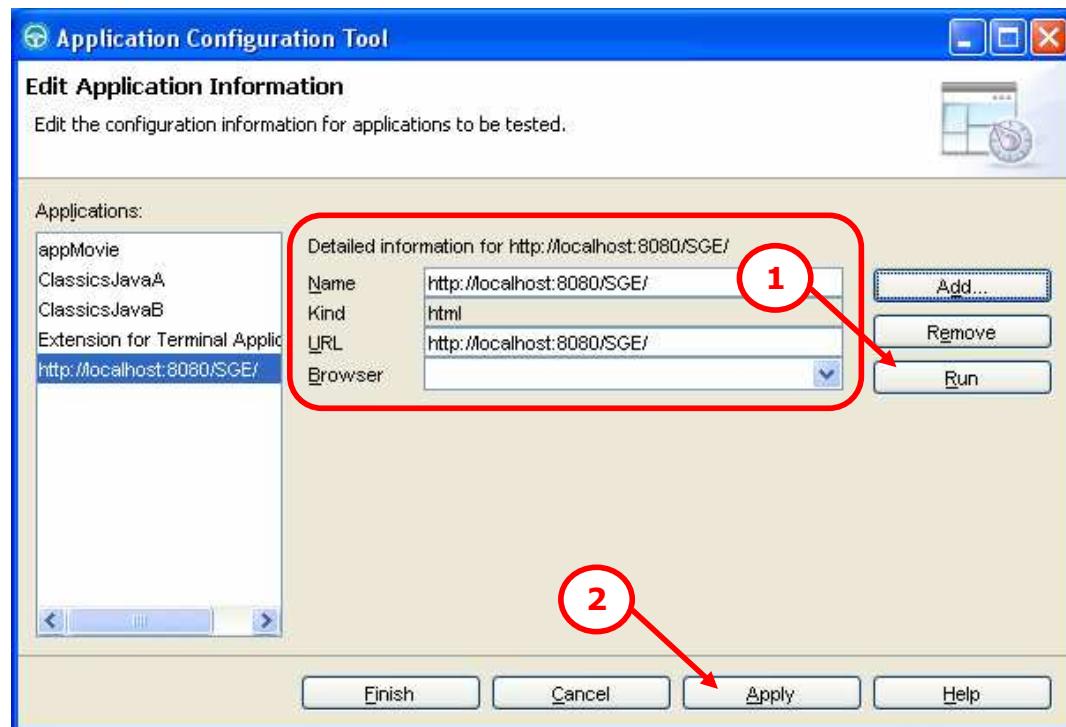


3. Ahora, especifique la página principal o el URL de la aplicación. Para ello, si elige **Local**, busque un archivo .htm o .html. Si elige **URL**, introduzca una dirección URL. Para el ejemplo, seleccione **URL**. Luego, pulse **Finish**.



4. La nueva aplicación aparecerá en la lista “Aplicaciones” de la ventana “Herramienta de Configuración de Aplicación” y su información detallada en el panel izquierdo (Nombre, Tipo y URL). Ahora, pulse **Run** para probar que la aplicación esté bien configurada. Luego, seleccione el botón **Apply**.

**Nota:** Asegúrese de Iniciar el contenedor web **Tomcat** que gestionará la ejecución de la aplicación.



## 2.1.4. Configuración de las propiedades del reconocimiento de objetos

A partir de la versión 8.0, el RFT introduce una nueva característica que permite configurar las propiedades de reconocimiento de objetos. Esta función utiliza la Herramienta de configuración de propiedades de objetos para configurar las propiedades de reconocimiento de objetos en la biblioteca de objetos personalizados. Durante la grabación de scripts de prueba, el archivo de biblioteca de objetos personalizados se utiliza como referencia para establecer las propiedades de reconocimiento de objetos y los pesos de las propiedades en el mapa de objetos, tal como se muestra en la figura 2.2.

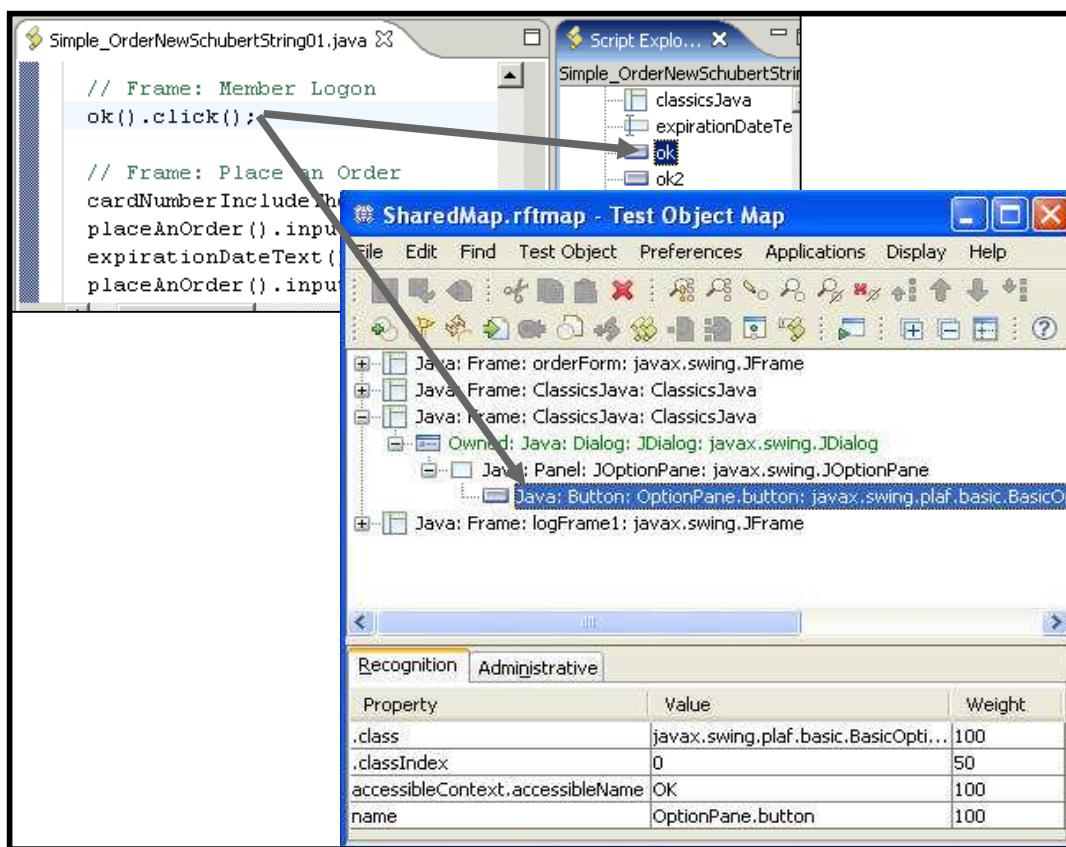


Figura 2.2. La información que describe los objetos de prueba es almacenada en un mapa de objetos de prueba

Un mapa de objetos de prueba es un catálogo jerárquico de descripciones de objetos de prueba. En la figura anterior se muestra dos pestañas que contienen las propiedades de un objeto de prueba: la pestaña “Reconocimiento” muestra los datos de reconocimiento que utiliza RFT para encontrar los objetos de prueba durante la reproducción, y la pestaña “Administrativo” muestra los datos administrativos internos del objeto.

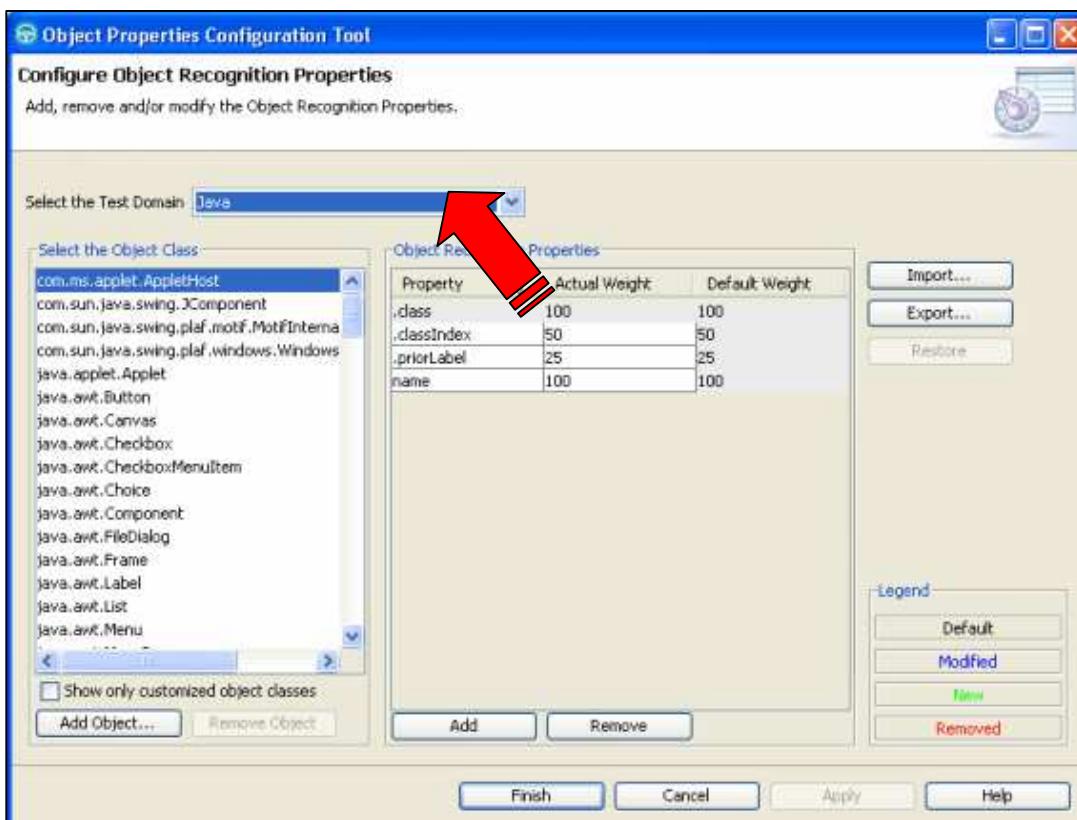
La Herramienta de configuración de propiedades de objetos por defecto muestra un catálogo de objetos para un dominio específico de prueba y objetos personalizados por el usuario que son utilizados por RFT. Personalizar las propiedades de reconocimiento de objeto ayuda a hacer que el script sea flexible a cambios en la aplicación de prueba.

En seguida, se muestra los pasos para ingresar a la herramienta de configuración de propiedades de reconocimiento de objetos que pertenecen al dominio de pruebas Java y se reflejará en todos los scripts a crear. El ejemplo muestra cómo eliminar el ancho y alto de un objeto.

1. Desde la barra de menú principal, seleccione **Configure > Configure Object Recognition Properties**.



2. En la siguiente ventana, seleccione Java como dominio de prueba.

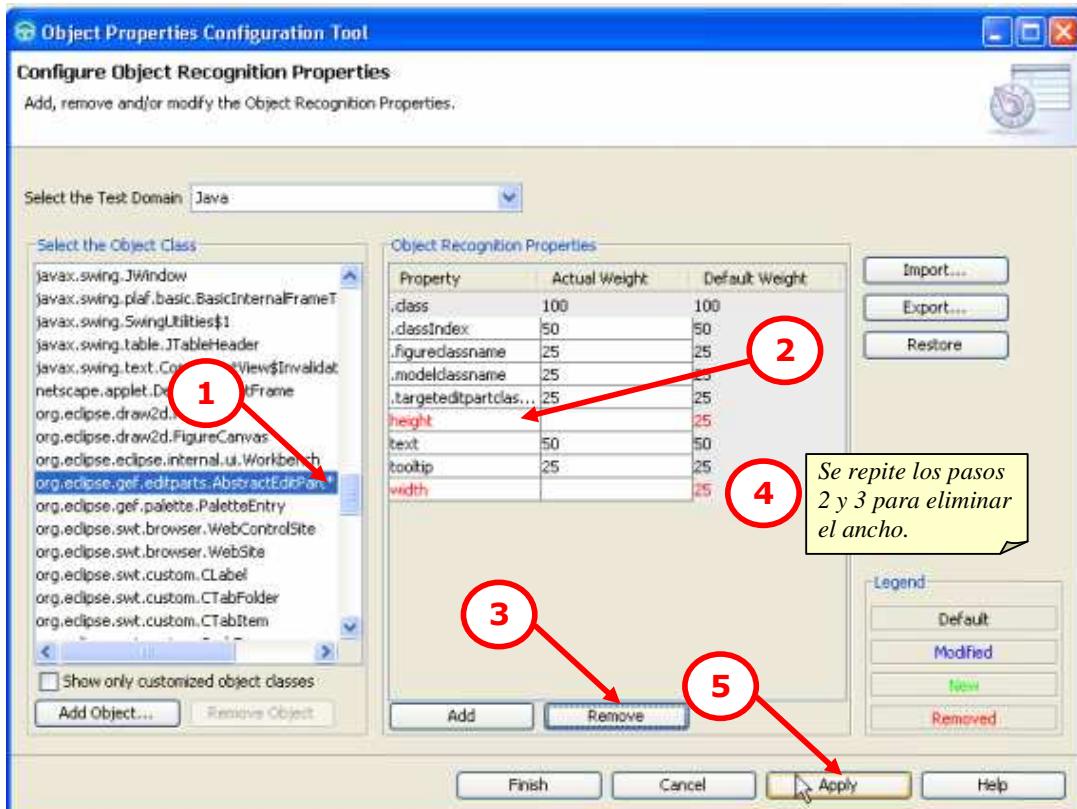


Los colores que se muestran en la leyenda hacen referencia al estado de las propiedades de un objeto:

- Default (color negro), es una propiedad por defecto
- Modified (color azul), si la propiedad se modificó
- New (color verde), si es una nueva propiedad
- Removed (color rojo), si la propiedad se eliminó

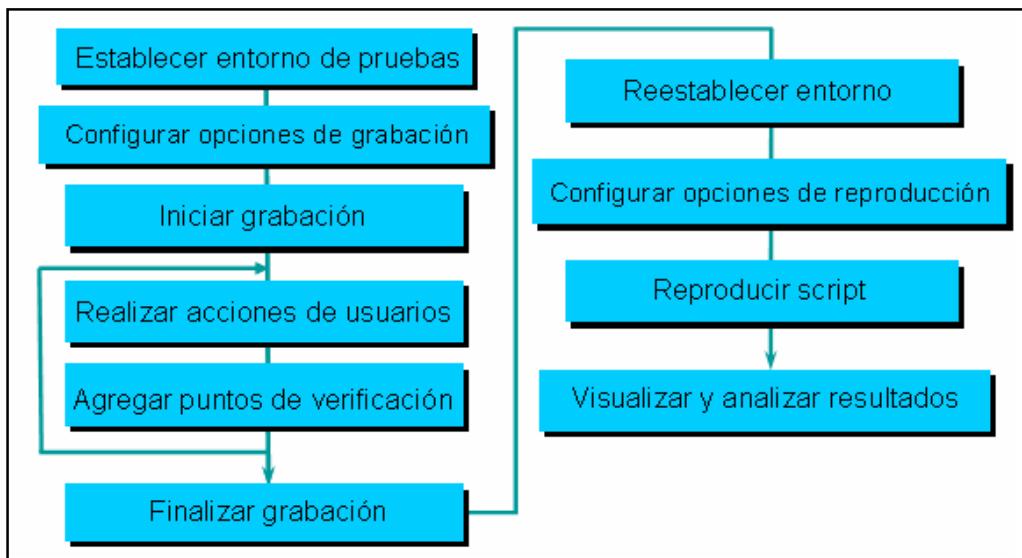
**Nota:** Si el objeto de prueba requerida de una aplicación bajo prueba no está en la lista, puede agregarlo a la biblioteca de objetos y personalizar sus propiedades de reconocimiento y pesos desde el botón **Add Object**.

3. A continuación, seleccione el objeto y las propiedades a eliminar. La eliminación de estas propiedades hacen que los scripts sean más resistentes a los cambios.



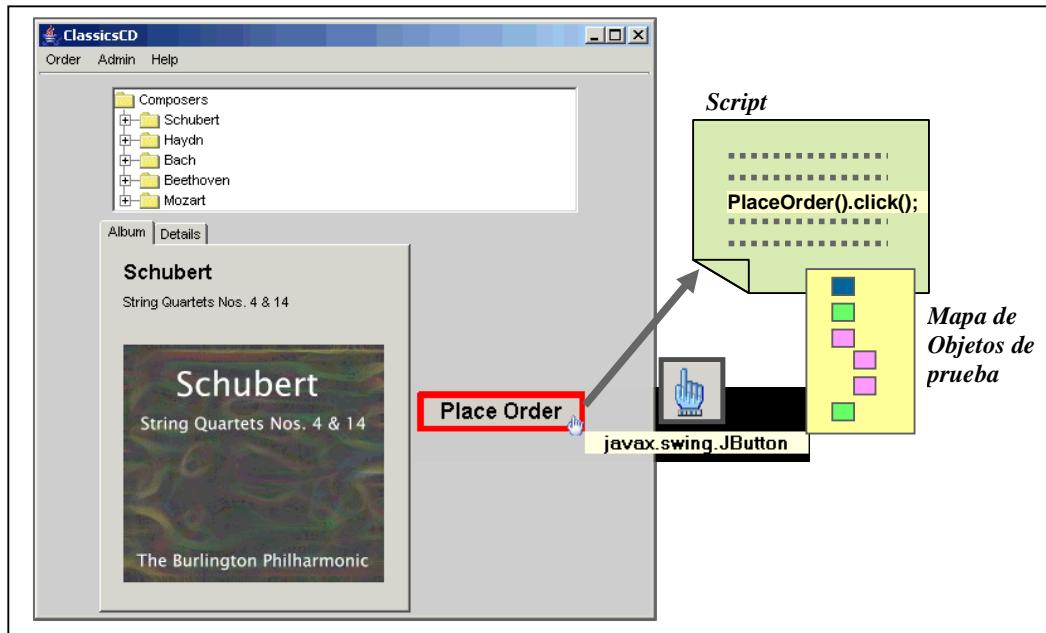
## 2.2. SCRIPT DE PRUEBAS FUNCIONALES

Un script de prueba en RFT puede ser simplificado cuando se crea a partir de un proceso de grabación, o de Java si se crean directamente con código Java. El primer tipo de script genera un archivo que contiene las acciones del usuario en la aplicación bajo prueba y otro archivo con el script en código Java. La siguiente figura muestra los pasos de alto nivel para grabar, reproducir y visualizar los resultados de un script simplificado de pruebas funcionales en RFT:



**Figura 2.3 Proceso de grabación, reproducción y visualización de resultados de un script de pruebas funcionales**

Como se explicó en el apartado anterior, el proceso de grabación siempre genera un script, el cual se asocia con un mapa de objetos de prueba.



**Figura 2.4. El proceso de grabación produce un script y un mapa de objetos de prueba**

Un mapa de objetos de prueba puede ser privado o compartido.

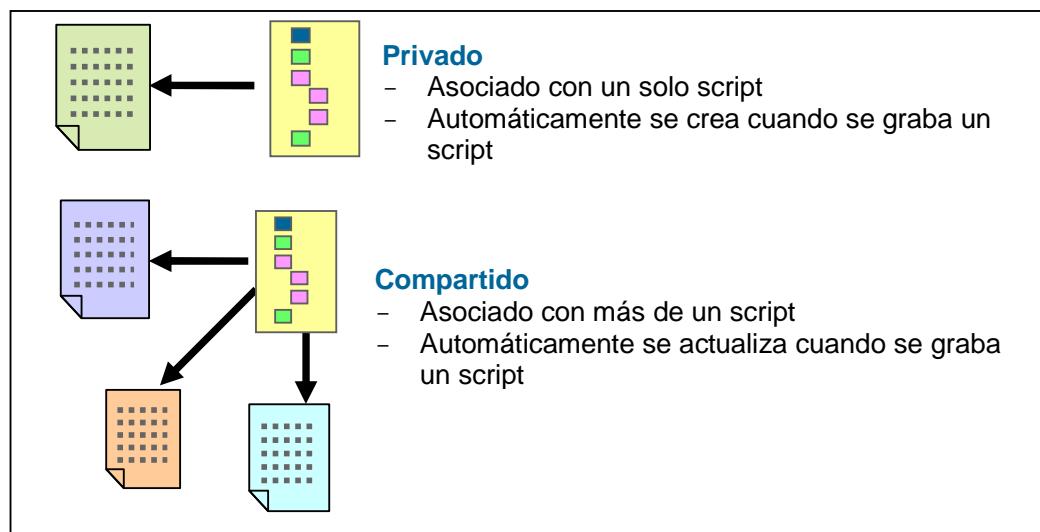


Figura 2.5. Mapa de objetos de prueba

En los siguientes puntos, se detallará el proceso de grabación, el proceso de reproducción y visualización de resultados.

### 2.2.1 Grabación de un script

Antes de iniciar el proceso de grabación, asegúrese de haber configurado el entorno de pruebas y la aplicación a probar.

Durante el proceso de grabación, se puede agregar **comandos controlados por datos** a fin de crear un pool de datos para probar otros valores durante el proceso de reproducción, y **puntos de verificación** para probar el estado de un objeto de la GUI.

La siguiente figura muestra el monitor con una barra de herramientas que se activa durante el proceso de grabación:

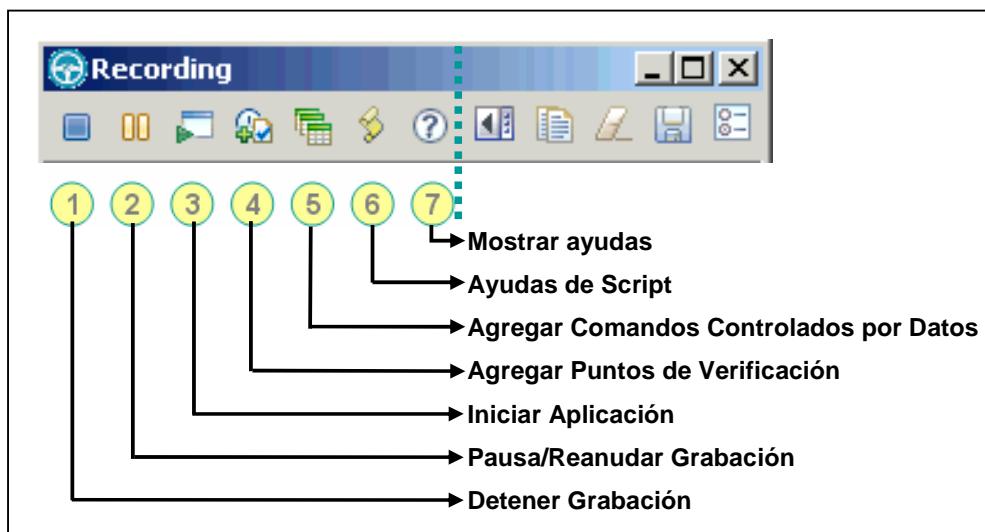


Figura 2.6. Barra de herramientas del monitor de grabación

#### 2.2.1.1 Comandos controlados por datos.

Este elemento creará un pool de datos con las variables que hacen referencia a un conjunto de objetos que contienen datos. Con RFT puede agregar comandos controlados por datos durante el proceso de grabación o desde la vista de aplicación una vez se haya generado el script.

#### 2.2.1.2 Puntos de verificación.

Un punto de verificación (PV) es la respuesta que esperamos del sistema. Esta respuesta muestra algún resultado sobre un objeto de la GUI, ejemplos:

- Sumas, totales, balances sobre una etiqueta o campos de texto
- Datos cargados sobre campos de texto de un formulario
- Mensajes de error sobre una etiqueta de un cuadro de diálogo
- Resultados de consultas sobre una tabla
- Imágenes

En RFT, un PV captura la información del objeto y los valores literales de la aplicación de prueba y la almacena, en la línea base, a efectos de comparación durante la reproducción. Cuando se reproduce un script, un punto de verificación vuelve a capturar la información del objeto para compararla con la línea base y para ver si se ha efectuado algún cambio, ya sea o no de forma intencionada. Comparar la información del objeto real de un script con la línea base resulta útil para identificar posibles defectos.

En RFT, se graba un PV mediante la selección de una de las 5 acciones siguientes sobre el objeto que deseamos evaluar:

- Para verificar datos.
- Para verificar propiedades del objeto
- Obtener el valor de una propiedad específica del objeto
- Esperar a seleccionar el objeto a probar. Esta opción se utiliza en caso sea necesario especificar un tiempo de espera para visualizar el resultado que se cargará sobre un objeto. Esto es para evitar problemas en la visualización de resultados.
- Visualizar cambios sobre una imagen. Al grabar este PV, se crea un archivo de imagen de línea base. Cada vez que reproduce el script, la imagen se comparará para ver si se han producido cambios, ya sea de forma intencionada o no intencionada. **La verificación de la imagen se hace estrictamente píxel a píxel.** Esto resulta útil para identificar errores posibles.

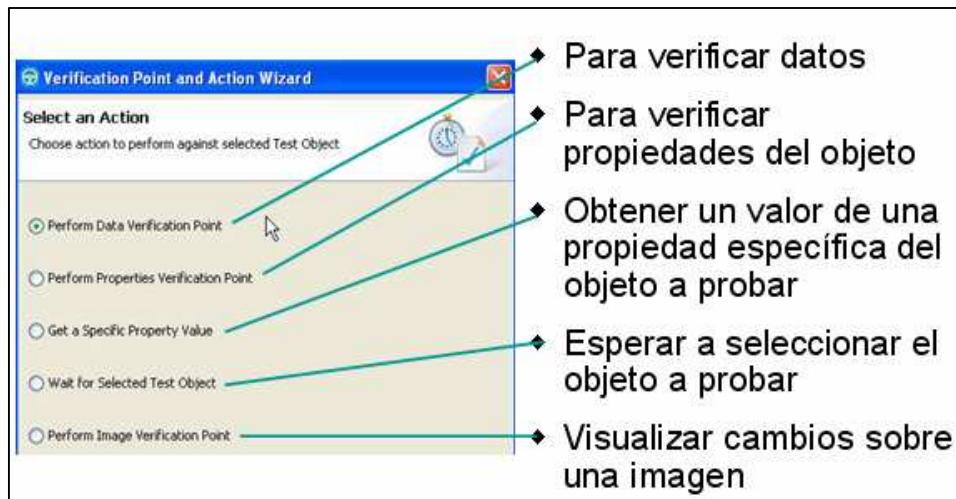


Figura 2.7. Tipos de Puntos de Verificación

A continuación, se muestran algunos ejemplos de PV:

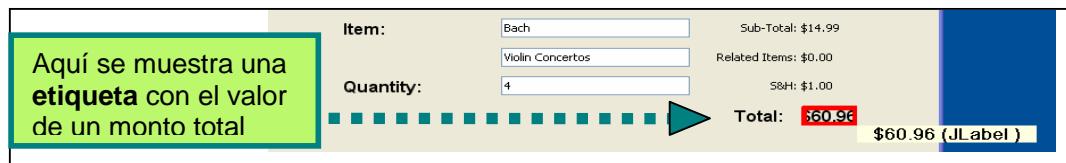


Figura 2.8. PV para verificar datos de una etiqueta



Figura 2.9. PV para verificar propiedades de una etiqueta

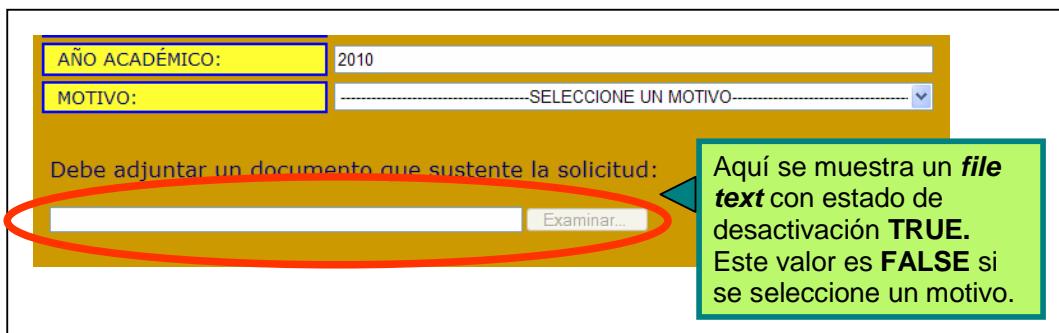


Figura 2.10. PV para verificar propiedades de un file text (Componente HTML)

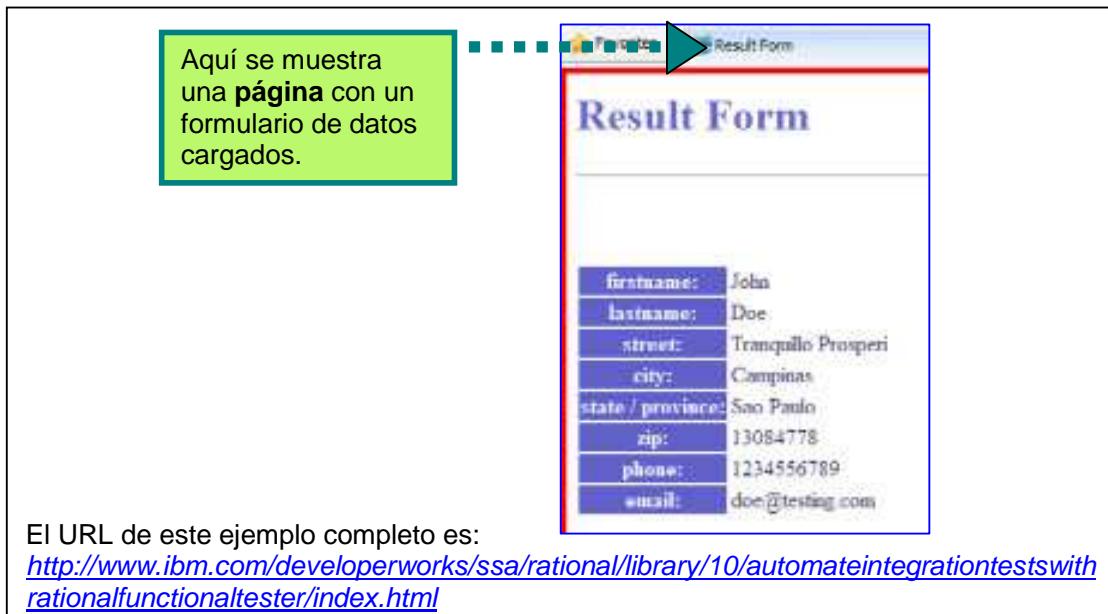


Figura 2.11. PV para verificar el título de una página (valor de una propiedad específica) que tiene el formulario con datos cargados

#### 2.2.1.3 Pasos para grabar un script.

Los pasos para grabar un script son los siguientes:

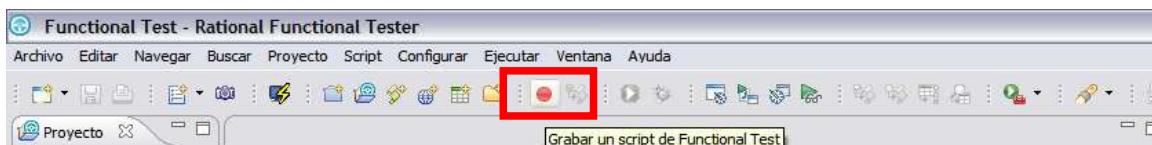
- Inicio de grabación** {
1. Crear un proyecto
  2. Crear el script
  3. Seleccionar la aplicación bajo prueba
  4. Realizar acciones de usuarios
  5. Añadir comandos controlados por datos
  6. Agregar puntos de verificación
  7. Finalizar proceso de grabación

**Nota:** Es normal, por primera vez, que los usuarios realicen acciones incorrectas durante el proceso de grabación. Cualquier error del usuario podrá ser corregido después, así es que no será necesario detener la grabación si se comete algún error.

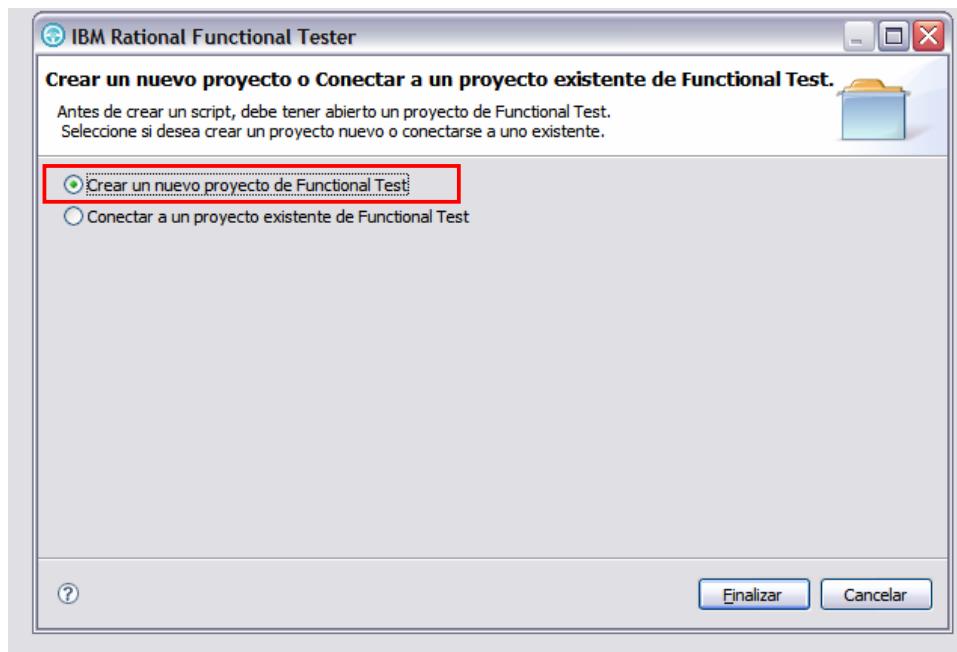
De inmediato, realizará los pasos citados grabará para garbar un script de prueba simplificado para probar una aplicación de escritorio Java.

### **Inicio de grabación**

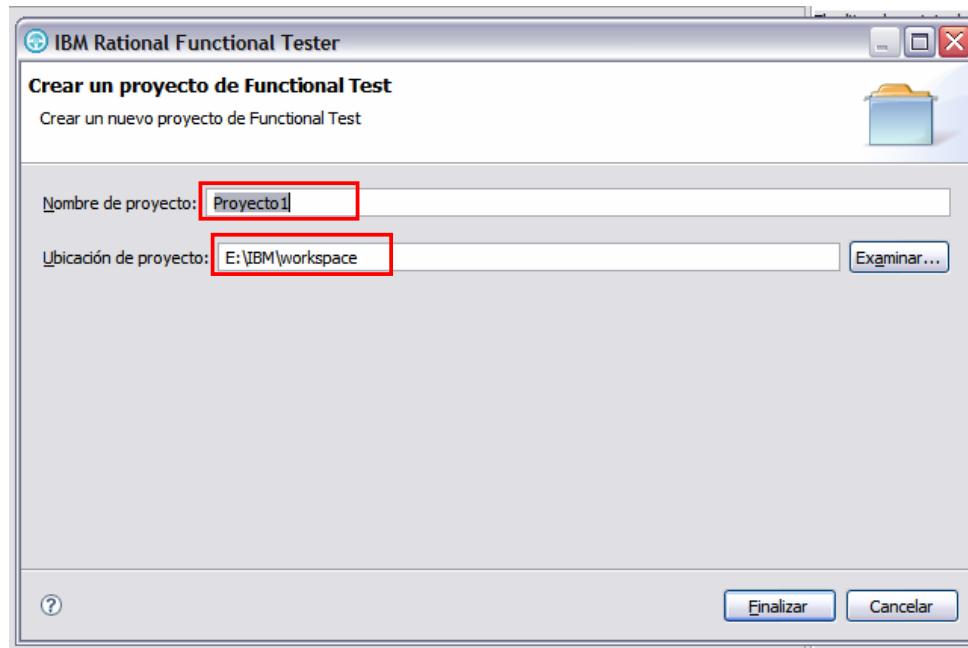
1. Para comenzar el proceso de grabación, pulse el ícono Grabar un script de *Functional Test* (●) desde la barra de herramientas.



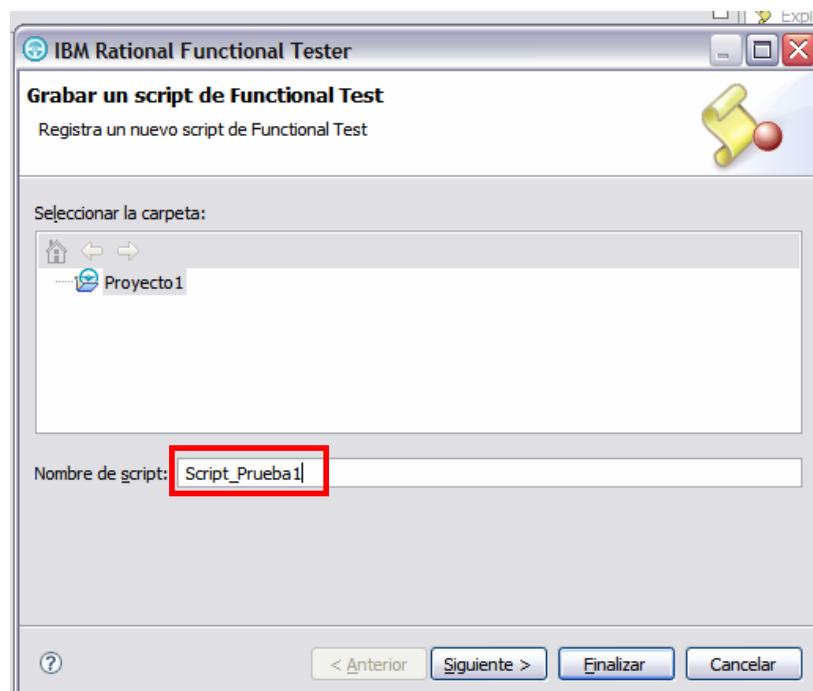
2. Al hacer click a “Grabar un Script” deberá seleccionar el proyecto a grabar. Seleccione “Crear un nuevo proyecto de *Functional Test*” y pulse Finalizar.



3. Edite el nombre del Proyecto, la ubicación de Proyecto y pulse Finalizar.



4. En la ventana “Grabar un script de *Functional Test*”, coloque el nombre del Script y pulse Finalizar.



5. La ventana de Functional Tester se minimiza automáticamente, y se muestra el Monitor de grabación.



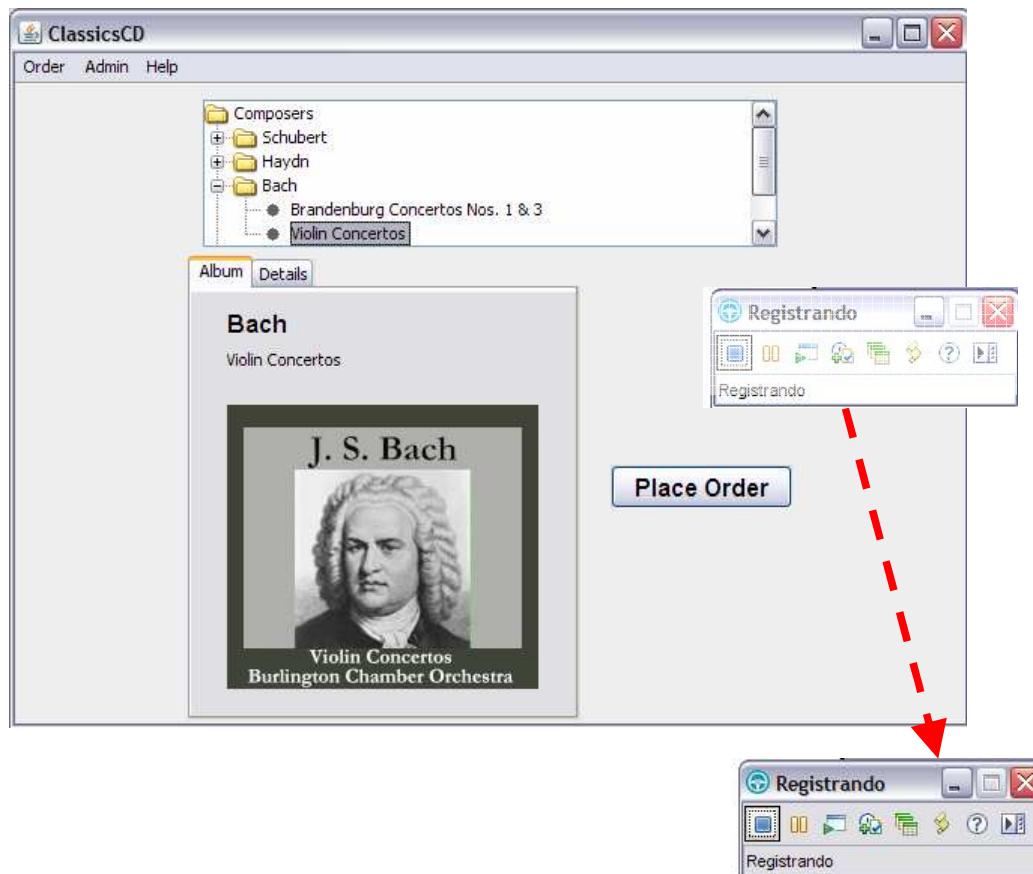
6. Para iniciar la aplicación de prueba, pulse el ícono Iniciar aplicación (►).



7. En la ventana “Iniciar aplicación”, seleccione **ClassicsJavaA - Java** y a continuación pulse Aceptar.



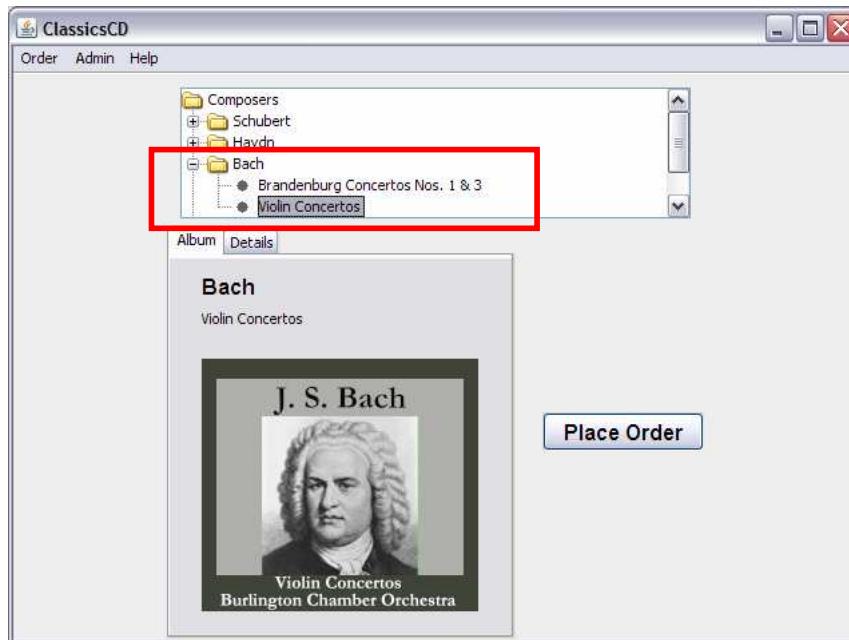
8. Se abre la aplicación de ejemplo, ClassicsCD. Si el monitor de grabación aparece delante de la aplicación, puede arrastrarlo hasta el ángulo inferior derecho de la pantalla.



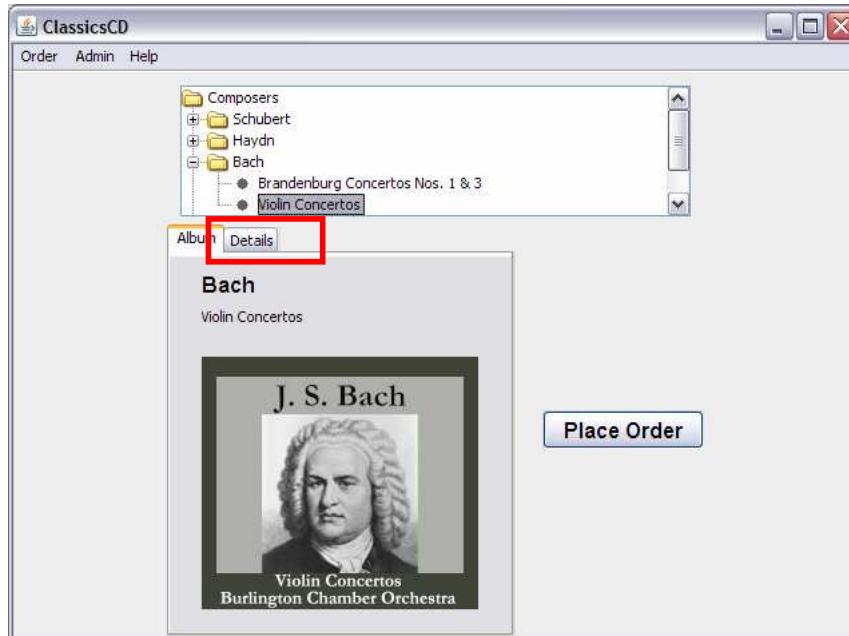
## Grabación de acciones

Ahora realizará las acciones de usuario sobre la aplicación **ClassicsJavaA** para generar un pedido.

1. Pulse el signo (+) situado al lado de Bach para abrir la lista de CD que estén en venta de dicho compositor y, a continuación, pulse Violin Concertos.



2. Ahora, pulse la página "Detalles" para ver la descripción del álbum.



3. Pulse Hacer pedido.



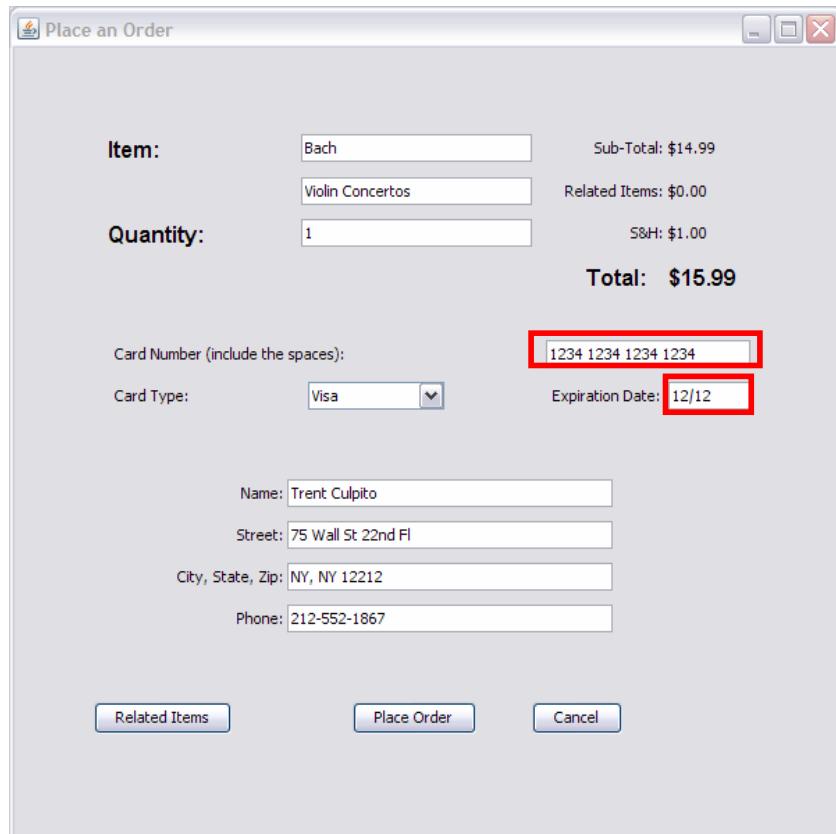
4. En la ventana “Member Logon”, conserve los valores predeterminados del Cliente existente Trent Culpito



5. Escriba xxxx en el campo Contraseña y pulse OK



6. En la ventana “Hacer Pedido”, escriba 1234 1234 1234 1234 en el campo número de tarjeta y, a continuación, escriba 12/12 en el campo fecha de expiración.

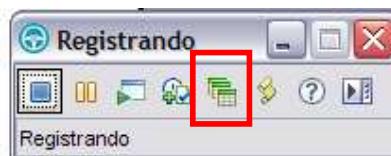


## Comandos controlados por datos

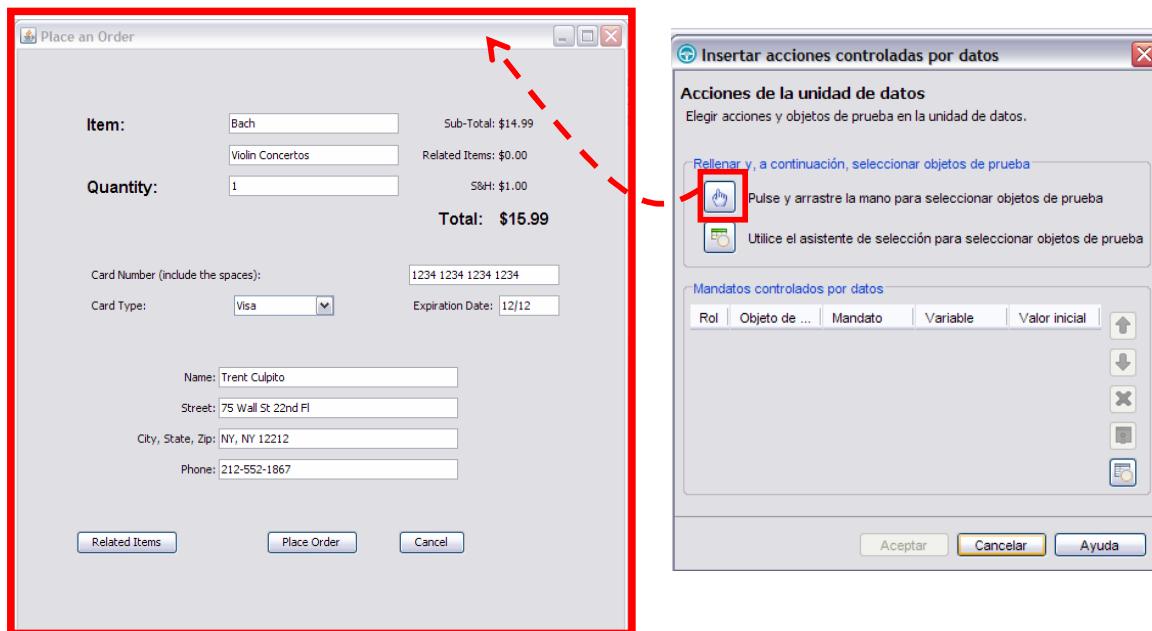
Ahora, agregue comandos controladas por datos a fin de crear un pool de datos con los datos procedentes de los objetos de la GUI.

**Nota:** Un pool de datos es una recopilación de registros de datos relacionados. Proporciona valores de datos a las variables de un script de prueba durante la reproducción de éste.

1. En la barra de herramientas del monitor de grabación, pulse Insertar comandos controlados por datos ().

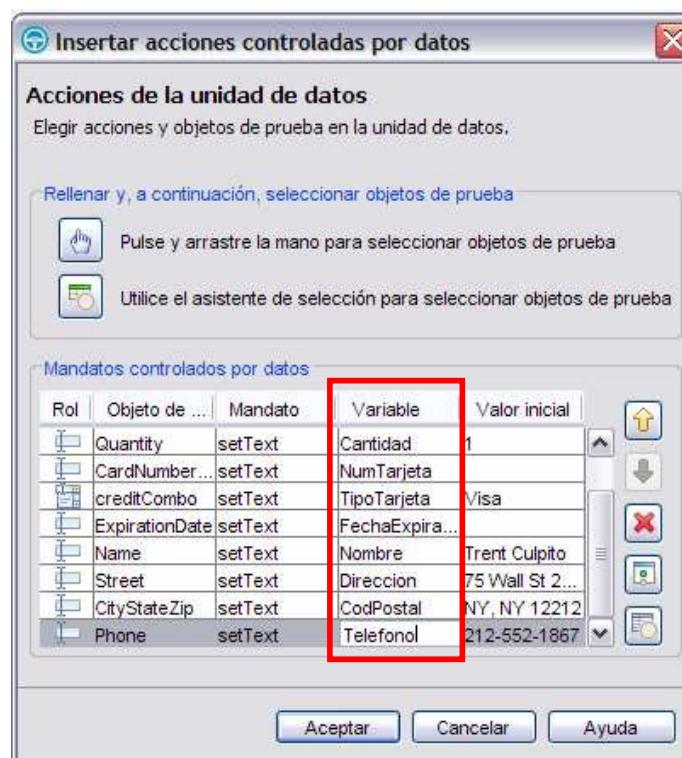


2. En la página Insertar “Acciones controladas por datos”, arrastre el buscador de objetos ( ) a la barra de título de la ventana “Hacer Pedido” de la aplicación. Aparecerá un contorno de color rojo alrededor de la ventana “Hacer Pedido”.

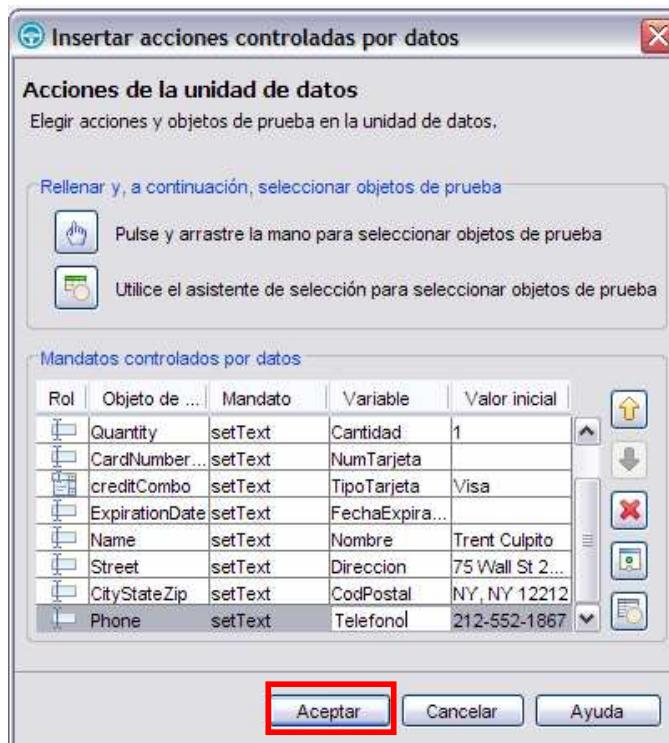


3. Suelte el botón del ratón. En la ventana “Acciones controladas por datos”, se mostrará la información sobre los controles de la GUI seleccionados. Reemplace los valores de la columna Variable por otros textos en español, pues esta columna será la cabecera del pool de datos que se creará. Utilice los siguientes nombres para la lista de variables; no utilice espacios en los nombres:

- Composer
- Artículo
- Cantidad
- NumTarjeta
- TipoTarjeta
- FechaExpiracion
- Nombre
- Direccion
- CodPostal
- Telefono



4. Ahora, pulse Aceptar.



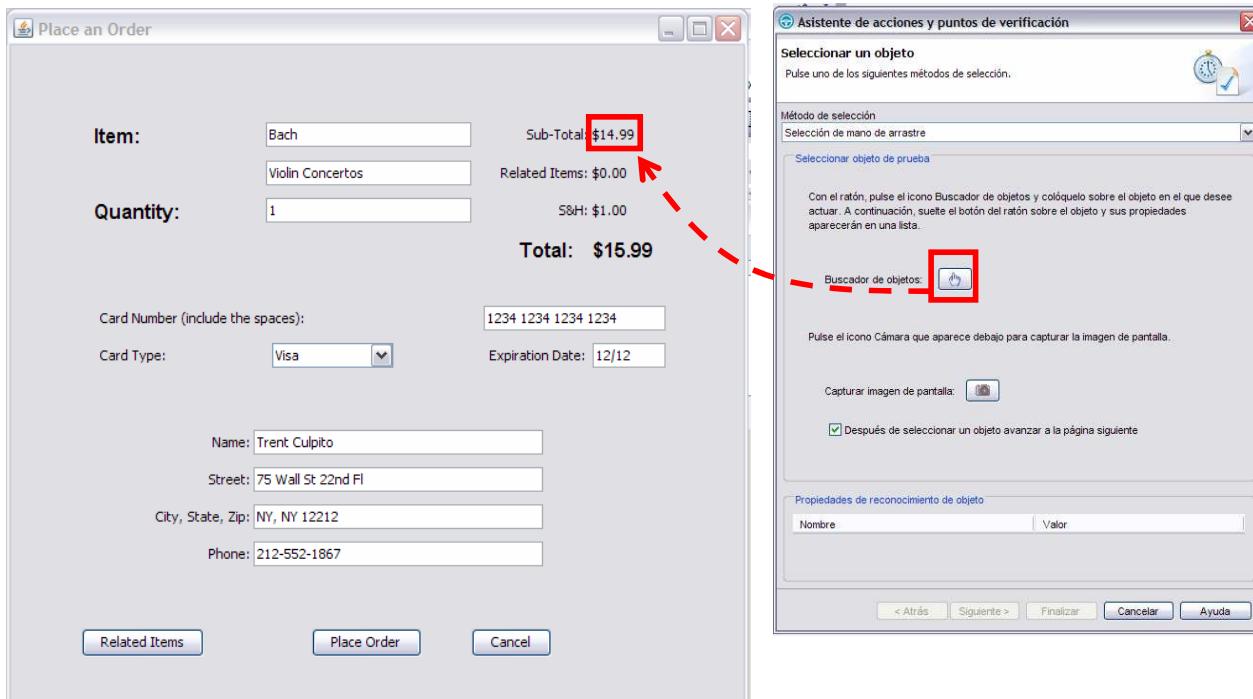
## Puntos de verificación

En esta sección, creará un punto de verificación con una referencia de agrupación de datos para comprobar que el importe del CD a comprar es correcto.

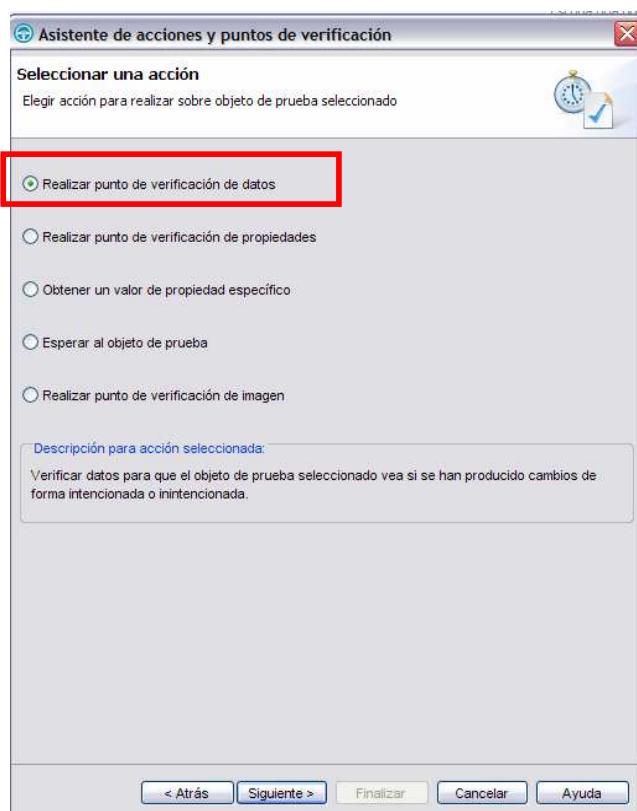
1. Desde el monitor de grabación, seleccione Insertar punto de verificación y mandato de acción (); se mostrará un asistente.



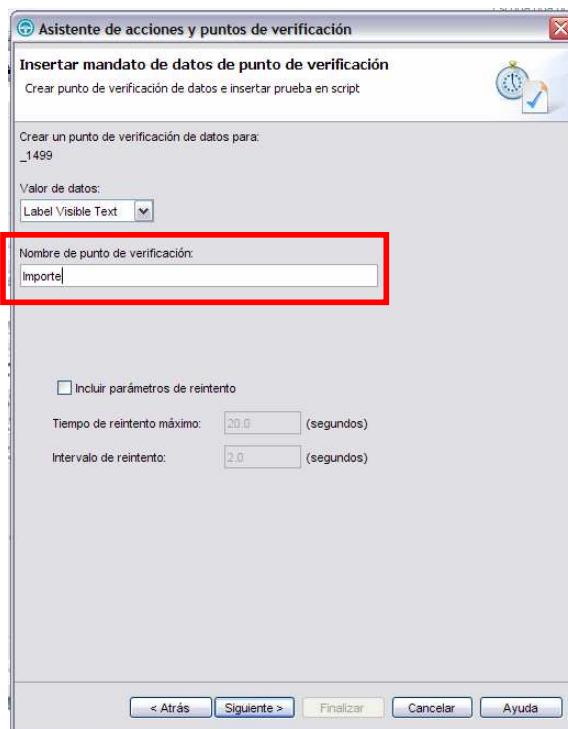
2. En el “Asistente de Acciones y puntos de verificación”, arrastre el buscador de objetos (🔍) hasta el valor \$14.99, que aparece junto a Sub-Total en la aplicación ClassicsCD. Aparece un contorno de color rojo en la cantidad, \$14.99.



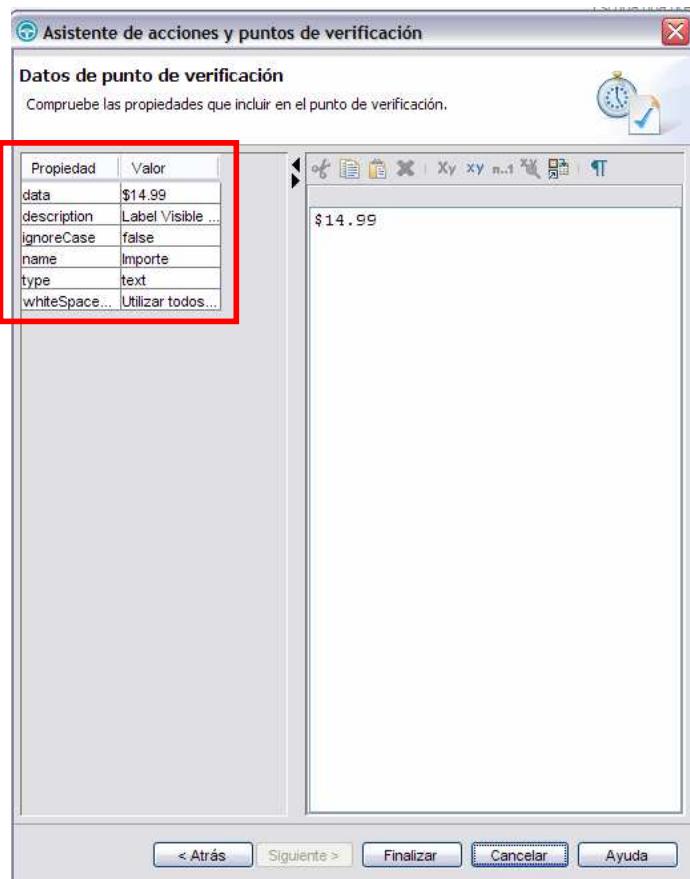
3. A continuación, pulse “Realizar punto de verificación de datos” para probar si el importe del CD cambia según la cantidad y pulse Siguiente.



4. Ahora, escriba Importe en Nombre de puntos de verificación y pulse Siguiente.



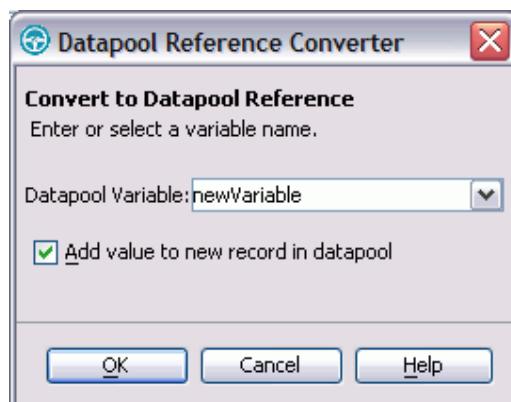
5. A continuación, se muestra la información del objeto que se le aplicó el PV. Luego, pulse Finalizar.



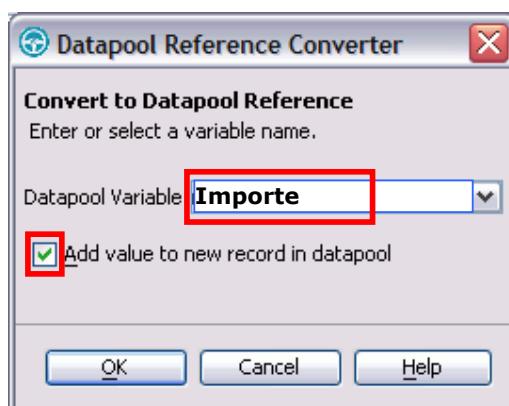
6. En la barra de herramientas de la página “Datos de punto de verificación”, pulse Convertir valor en referencia de agrupación de datos (Convertir valor en referencia de agrupación de datos) para utilizar un pool de datos en lugar de un valor literal en un punto de verificación. (Si no puede ver la opción Convertir valor en referencia de agrupación de datos en la barra de herramientas, aumente el tamaño de la página arrastrándolo hacia la derecha).



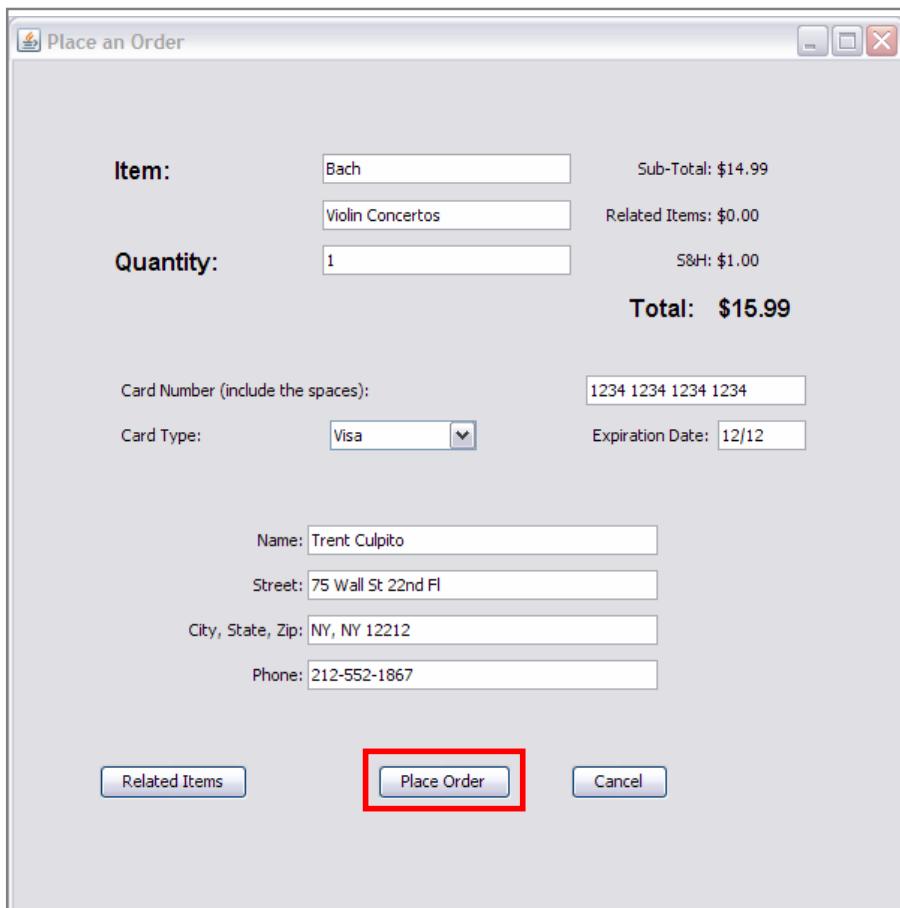
7. Se abre el cuadro de diálogo Convertidor de referencias de agrupación de datos.



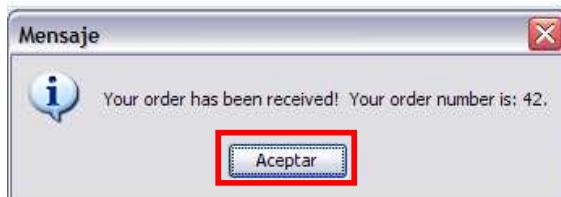
8. En Variable de agrupación de datos, escriba **Importe** para sustituir el valor **newVariable** como cabecera del pool de datos. Luego, asegúrese de seleccionar **Añadir valor al nuevo registro de la agrupación de datos** para añadir la variable Importe al registro del pool de datos existente que se ha creado en el apartado anterior. Por último, pulse OK.



9. Para hacer un pedido, en la aplicación ClassicsCD, pulse Hacer pedido.



10. A continuación, pulse Aceptar para cerrar el mensaje que confirma el pedido.



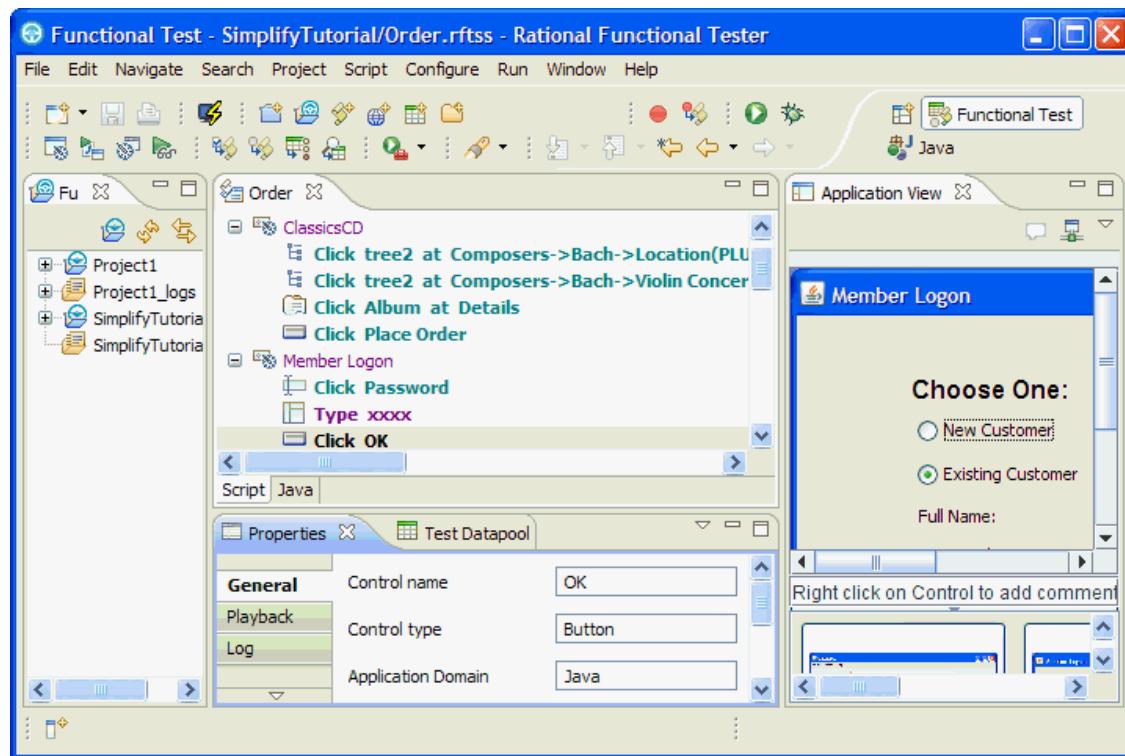
11. Para cerrar la aplicación, pulse el símbolo X que aparece en el ángulo superior derecho de la aplicación ClassicsCD.

## Finalización de la grabación

En la barra de herramientas Grabación, pulse Detener grabación (■) para escribir toda la información grabada en el script de prueba.



El script de prueba se muestra en la ventana del editor de script.



## Adición de datos al pool de datos

Antes de iniciar el proceso de reproducción, agregue más registros al pool de datos para probar la aplicación de ejemplo ClassicsCD efectuando más pedidos del CD.

El editor del pool de datos tiene el aspecto de esta tabla:

	Item::java.lang.String	_1499::java.lang.String	Cantidad::java.lang.Integer	NumTarjeta::java.lang.String	TipoTarjeta::java.lang.String	FechaExpiración::java.util.Date	Nombre::java.lang.String
0	Bach	Violin Concertos	1	9999 9999 999...	Visa	10/11	Trent Culp

1. Coloque el puntero del ratón en el editor de agrupaciones de datos y, a continuación, pulse Intro para añadir una fila después de la primera.
2. Para ahorrar tiempo, copie los datos de la fila 0 de la agrupación de datos en las filas vacías que haya creado.
  - a. Coloque el puntero del ratón en la celda de la fila 0, pulse el botón derecho del ratón y, a continuación, pulse Copiar.
  - b. Coloque el puntero del ratón en la celda de la fila 1, pulse el botón derecho del ratón y, a continuación, pulse Pegar.
  - c. Pulse Sí para pegar los datos en la fila vacía.
3. Cambie el valor de los datos que se encuentran en las columnas **Compositor**, **Artículo**, **Cantidad** e **Importe**.
4. En el editor Agrupación de datos de prueba, pulse X para cerrar el editor de agrupaciones de datos y, a continuación, pulse Sí para guardar los cambios que haya efectuado en la agrupación de datos de prueba.

## 2.2.2 Reproducción de un script

Al reproducir un script de prueba funcional se repite las acciones registradas y automatiza el proceso de pruebas de software. Con la automatización, puede probar cada nueva construcción de su aplicación más rápido y más bien que por las pruebas de forma manual. Esto disminuye el tiempo de prueba y aumenta la cobertura y consistencia.

Existen dos fases generales en la reproducción de un script:

- En la fase de desarrollo de pruebas, se reproducen scripts para verificar que funcionan tal como se pretendía, utilizando la misma versión de la aplicación que se somete a prueba y que se utiliza para grabar. Esta fase valida el comportamiento que se espera de la aplicación.
- En la fase de pruebas de regresión se reproducen scripts para comparar la última compilación de la aplicación que se somete a prueba con la línea base establecida durante la fase de desarrollo de prueba. La prueba de regresión identifica las diferencias que puedan haber surgido desde la última compilación. Puede evaluar estas diferencias para determinar si se trata de defectos o de cambios.

Para reconocer los objetos de prueba durante la reproducción, RFT utiliza el reconocimiento de pesos de las propiedades del objeto. La siguiente figura explica que durante el proceso de reproducción, RFT puede reconocer un objeto aún si tiene dos propiedades diferentes con respecto a los encontrados en la línea base.

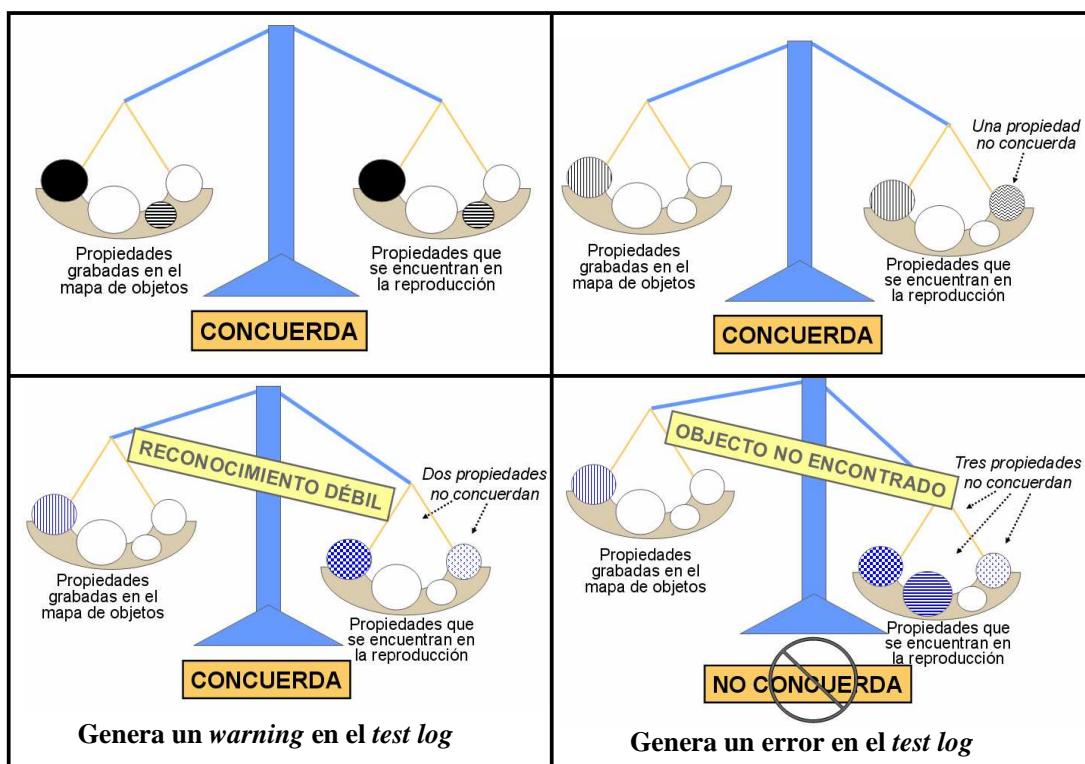


Figura 2.12. Reconocimiento de objetos

Cada propiedad de un objeto en el mapa de objetos de prueba tiene un valor de peso que se asignaron al grabar el script. Este valor de peso es un número de 0 a 100. A cada propiedad de un objeto encontrado durante la reproducción,

RFT le asigna una puntuación de reconocimiento. La mayor puntuación de reconocimiento significa una menor concordancia con la del objeto de la línea base. El objetivo de este mecanismo es permitir la reproducción de scripts a pesar de cambios significativos en los objetos de la aplicación bajo prueba.

### 2.2.3 Revisión de los resultados

En esta etapa, se analiza el resultado de la prueba a partir del *test log* que se generó al finalizar el proceso de reproducción. Durante la revisión, se determinará la causa de los *warnings* y errores encontrados. Por defecto, el archivo del *test log* se mostrará en formato HTML con 3 secciones en el panel izquierdo: resultado de errores, resultado de *warnings* y puntos de verificación. En la sección de la derecha, se describen con más detalle los errores y *warnings* generados, y un enlace para visualizar la página de Comparación de Puntos de Verificación.



Figura 2.13. Test Log

### 2.2.4 Características avanzadas de script de pruebas

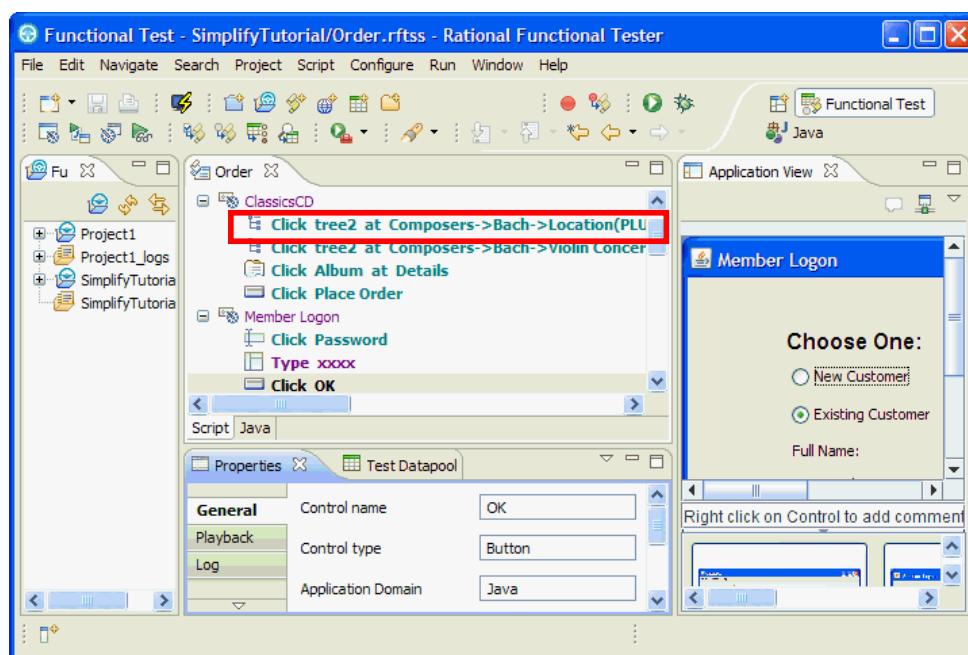
En este apartado, aprenderá a editar un script simplificado mediante la vista de Aplicación.

Los controles de la aplicación, sus datos y sus detalles de propiedades se capturan durante la grabación. Los detalles capturados muestran efectos visuales de la aplicación en la **vista Aplicación**. Puede modificar el script de prueba para probar controles adicionales, o para crear o editar puntos de verificación seleccionando los controles de la aplicación a partir de los efectos visuales de la aplicación sin tener que volver a ejecutar la aplicación bajo prueba.

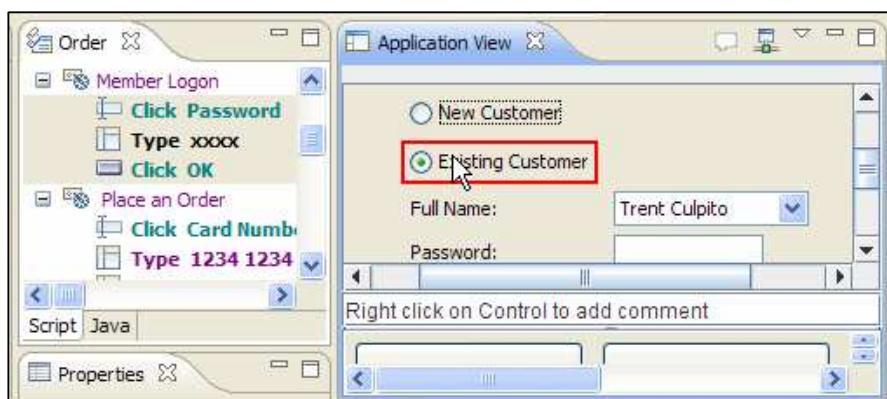
#### 2.2.4.1 Insertar un punto de verificación mediante el efecto visual de la aplicación

Se creará un punto de verificación en el árbol Compositores para comprobar si todos los compositores y sus CD's están listados. Este punto de verificación no se ha insertado cuando se grabó el script de prueba. En lugar de grabar el script o abrir la aplicación para volver a ejecutarla, añada un punto de verificación en el script de prueba desde los efectos visuales de la aplicación. Para ello, realice los siguientes pasos:

1. Desde la pestaña Script seleccione la línea de prueba **Click tree2 at Composers->Bach->Location(Plus\_Minus)** (la segunda línea del script de prueba). En la vista Aplicación, el efecto visual de la aplicación resalta la región del árbol Compositores en color azul.



2. Desde la vista de Aplicación, sitúe el ratón encima de la región del árbol de Compositores. La región se resalta en rojo.



- Pulse el botón derecho del ratón y seleccione "Insertar punto de verificación > Punto de verificación de datos".

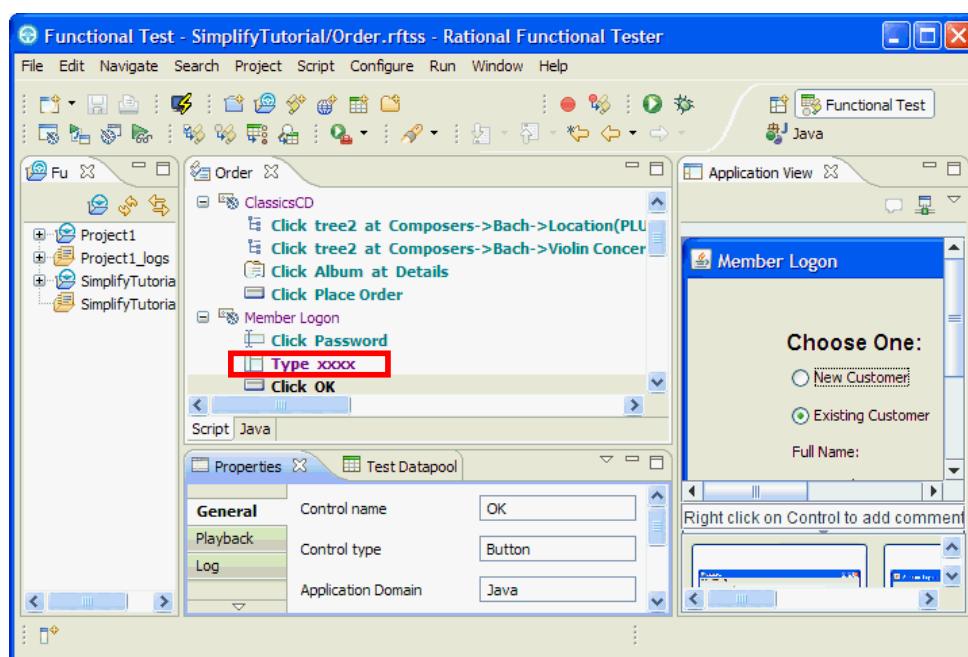
### ¿Qué ha sucedido en el script de prueba?

La línea de prueba **Verify data in tree2** se añade al editor del script (después de la segunda línea de prueba).

#### 2.2.4.2 Añadir un control adicional a la prueba.

En la ventana Member Logon de la aplicación ClassicsJavaA, no se ha grabado el campo Recordar contraseña para pruebas. Añada el control Recordar contraseña inexistente en el script de prueba:

- Seleccione la línea de prueba Type xxxx en el editor de script. Los efectos visuales de la aplicación Member Logon se muestran en la vista Aplicación.



- Sitúe el ratón encima del control Recordar contraseña en el efecto visual de la aplicación. El control Recordar contraseña se resalta en rojo.
- Pulse el botón derecho del ratón y seleccione el control "Insertar recordar contraseña > seleccionar".

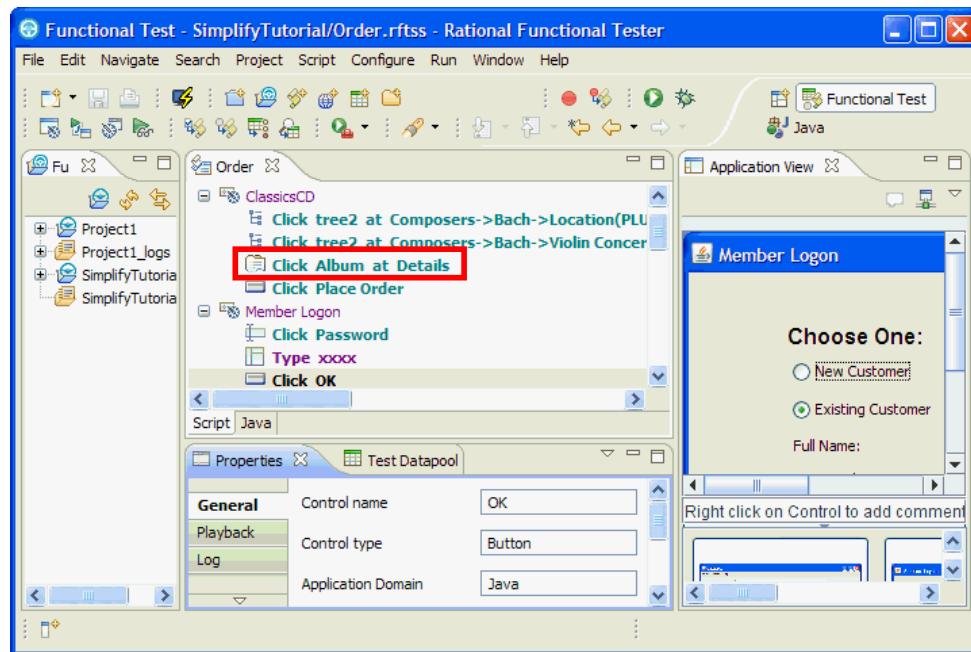
### ¿Qué ha sucedido en el script de prueba?

La línea de prueba Seleccionar recordar contraseña se inserta en el script de prueba.

#### 2.2.4.3 Cómo modificar la línea de prueba en el editor de script y las propiedades.

Puede modificar la línea de prueba en el editor de script y también especificar detalles tales como los parámetros de reproducción e información de registro para la ejecución de la línea de prueba en la vista Propiedades. Ahora, inhabilite la línea de prueba para ver los detalles del álbum y especificar la información que se debe mostrar en el registro durante la ejecución de la verificación de datos para la lista de control Compositores.

1. Seleccione la línea de prueba Click Album at Details en el editor de script.



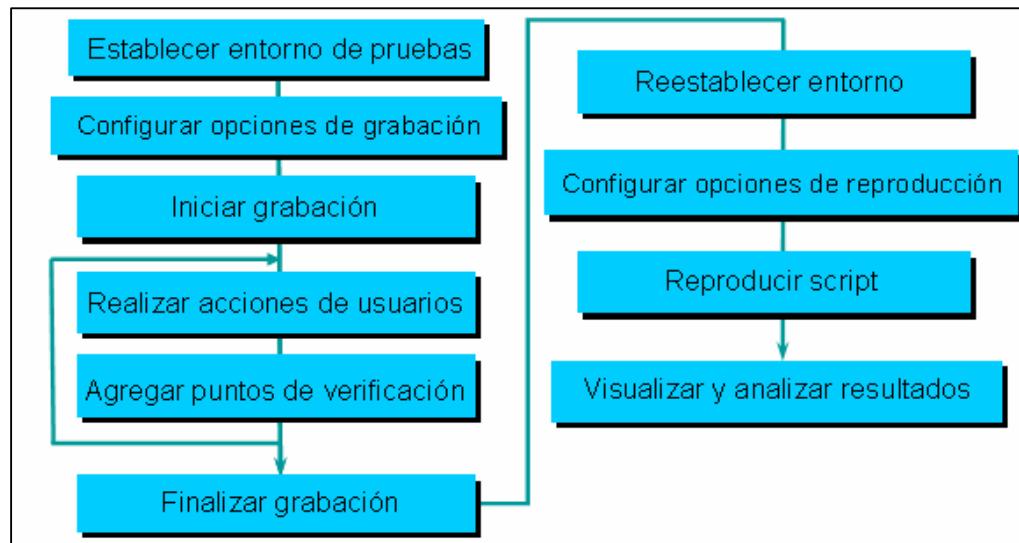
2. Pulse con el botón derecho del ratón y seleccione la acción Habilitar/inhabilitar para inhabilitar la línea de prueba. La línea de prueba no se ejecuta durante la siguiente reproducción.
3. Seleccione la línea de prueba **Verify data in tree2** que se ha insertado en el script mediante el efecto visual de la aplicación.
4. Pulse la página de Registro en la vista Propiedades y seleccione Instantánea de control para ver el estado del control durante la ejecución. Esta instantánea se muestra en el registro de reproducción.

## Actividades

1. A partir de la aplicación de escritorio Java entregada en clase, realice los siguientes procesos para un caso de uso:
  - a. Grabación de un script
  - b. Reproducción del script
  - c. Análisis de resultados
2. A partir de la aplicación web Java entregada en clase, realice los siguientes procesos para un caso de uso:
  - a. Grabación de un script
  - b. Reproducción del script
  - c. Análisis de resultados
    - ¿Qué *warnings* se generaron? Comente
    - ¿Qué errores se generaron? Comente
3. A partir de la aplicación web Java de la pregunta 2, realice los siguientes procesos para el mismo caso de uso:
  - a. Modifique la aplicación según indicaciones del docente
  - b. Reproducción del script grabado en la pregunta 2
  - c. Análisis de resultados
    - ¿Qué *warnings* se generaron? Comente
    - ¿Qué errores se generaron? Comente

## Resumen

- IBM *Rational Functional Tester* (RFT) es una herramienta automatizada para la realización de pruebas funcionales y de regresión. Permite probar aplicaciones Java, .NET y basadas en Web.
- Con la **tecnología de grabación** del RFT, se puede generar scripts mediante la ejecución y el uso de la aplicación bajo prueba.
- Gracias a la **tecnología ScriptAssure** del RFT se puede crear scripts de prueba más resistentes a los cambios en los objetos de las aplicaciones.
- Un script simplificado de prueba funcional en RFT genera dos archivos: un script con las acciones del usuario y un script con código java.
- Los pasos de alto nivel del proceso de grabación, reproducción y visualización de resultados de un script de prueba funcional se muestra en la siguiente figura:



- En RFT, un punto de verificación captura la información del objeto y los valores literales de la aplicación de prueba y la almacena, en la línea base, a efectos de comparación durante la reproducción. Cuando se reproduce un script, un punto de verificación vuelve a capturar la información del objeto para compararla con la línea base y para ver si se ha efectuado algún cambio, ya sea o no de forma intencionada. Comparar la información del objeto real de un script con la línea base resulta útil para identificar posibles defectos.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

☞ <http://publib.boulder.ibm.com/infocenter/rfthelp/v8r1/index.jsp>

En esta página encontrará información del entorno IBM RFT: guía de instalación, guía de aprendizaje, entre otros temas.

☞ <http://www.ibm.com/developerworks/ssa/rational/library/09/functionaltestertableobjects/#download>

Este artículo muestra cómo usar IBM RFT para automatizar objetos ajustados dentro de las celdas de una tabla HTML manteniendo las celdas especificadas como referencia. Lo ayudará a automatizar operaciones con estos objetos.

☞ <http://www-01.ibm.com/support/docview.wss?uid=swg21213488>

En esta página se describe la gestión de los objetos de prueba en el IBM RFT.



## FUNDAMENTOS RATIONAL PERFORMANCE TESTER

### LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno realiza pruebas de rendimiento para su proyecto final, el cual permite resolver los desafíos de pruebas de rendimiento más comunes.

### TEMARIO

#### 3.1. Tema 5: Introducción al Rational Performance Tester

- 3.1.1. Arquitectura de Rational Performance Tester
- 3.1.2. Características y beneficios de Rational Performance Tester

#### 3.2. Tema 6: Pruebas de rendimiento en RPT

- 3.2.1. Grabación de pruebas
- 3.2.2. Programación de pruebas
- 3.2.3. Ejecución de pruebas de rendimiento

### ACTIVIDADES PROPUESTAS

- Los alumnos realizan pruebas de rendimiento para un proyecto web Java.

### 3.1. INTRODUCCIÓN AL RATIONAL PERFORMANCE TESTER

Las pruebas de rendimiento tienen un significado diferente para diferentes personas, pero en general, estas pruebas se realizan para:

- Determinar el tiempo de respuesta de una solicitud
- Determinar el número de usuarios que el sistema soportará
- Determinar la configuración óptima del sistema
- Averiguar qué sucede cuando un sistema está sometido a carga pesada

Una prueba de rendimiento es el proceso de ejecución de un sistema mediante la actividad de emulación con el fin de:

- Observar el comportamiento del sistema
- Recolectar datos y métricas

Más allá de la observación y la recolección, los ingenieros utilizan los datos y métricas para mejorar el rendimiento de la aplicación objetivo. Este descubrimiento y la resolución de los cuellos de botella (los casos de bajo rendimiento o inaceptable) es el retorno real de la inversión (ROI) de una herramienta de pruebas de rendimiento.

Las actividades típicas de emulación son:

- Transacciones del usuario y escenarios de los CU
- Períodos de mayor actividad
- Trabajos por lotes y otros procesos internos del sistema
- Administrador de tareas (por ejemplo, copias de seguridad del sistema)

Los objetivos de las pruebas de rendimiento son:

- Determinar los tiempos de respuesta del sistema
- Determinar el número máximo de usuarios de un sistema (componentes, transacción, o configuración)
- Descubrir las configuraciones óptimas o mínima del sistema

Las pruebas de rendimiento pueden ayudar a reducir los costes del sistema, determinando qué recursos del sistema, tales como los re cursos de memoria y disco, son necesarios para ofrecer un rendimiento aceptable. Comprender los requisitos mínimos permite tomar mejores decisiones sobre qué hardware y software debe ser comprado para el despliegue de la aplicación objetivo.



**Figura 3.1. Costo por Defecto en Diseño, Pruebas y Producción**

Existen muchas herramientas para realizar pruebas de rendimiento, Rational Performance Tester (RPT) es una de ellas. RPT es una solución de verificación de cargas y rendimiento para equipos que se ocupen de la capacidad de ampliación de sus aplicaciones basadas en web. Gracias a la combinación de funciones de análisis detallados y fáciles de utilizar, Rational Performance Tester simplifica la creación de pruebas, la generación de cargas y la recopilación de datos para garantizar que las aplicaciones se amplíen hasta miles de usuarios concurrentes.

### 3.1.1 Arquitectura de Rational Performance Tester

#### 3.1.1.1. Relación entre el *workbench* y *workspace*

El *workbench* es la interfaz de usuario y el entorno de desarrollo integrado (IDE) en Eclipse y en los productos de la plataforma de desarrollo de software IBM Rational que se basan en Eclipse. Mientras que el *workspace* es la colección de proyectos y otros recursos que están actualmente en desarrollo en el *workbench*.

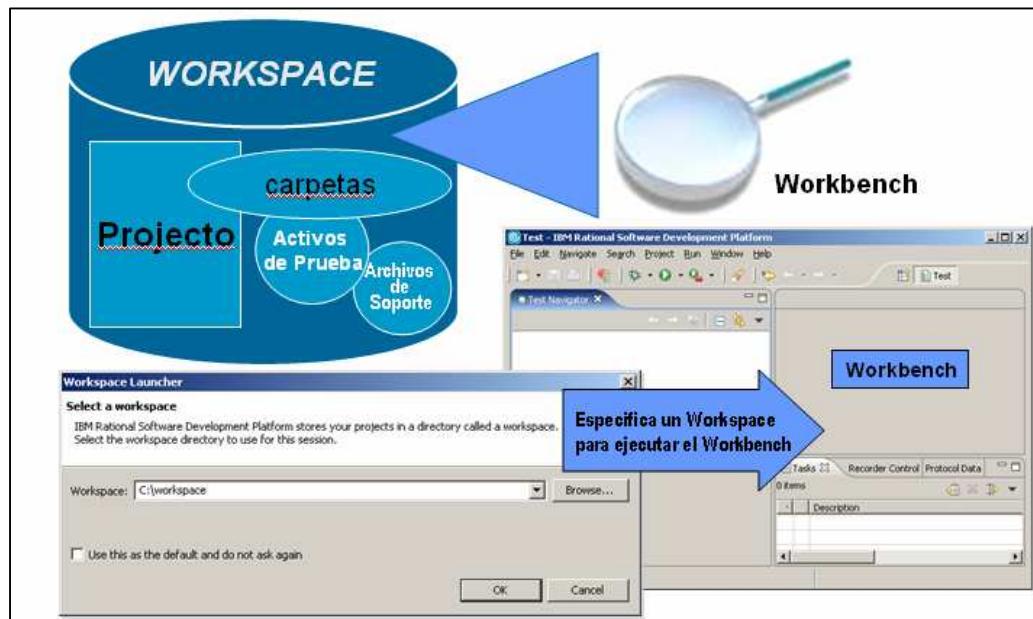


Figura 3.2. Relación entre el *workbench* y *workspace*

#### 3.1.1.2. Perspectiva de Prueba en RPT.

La siguiente figura muestra el entorno del RPT con la perspectiva **Performance Test** activa.

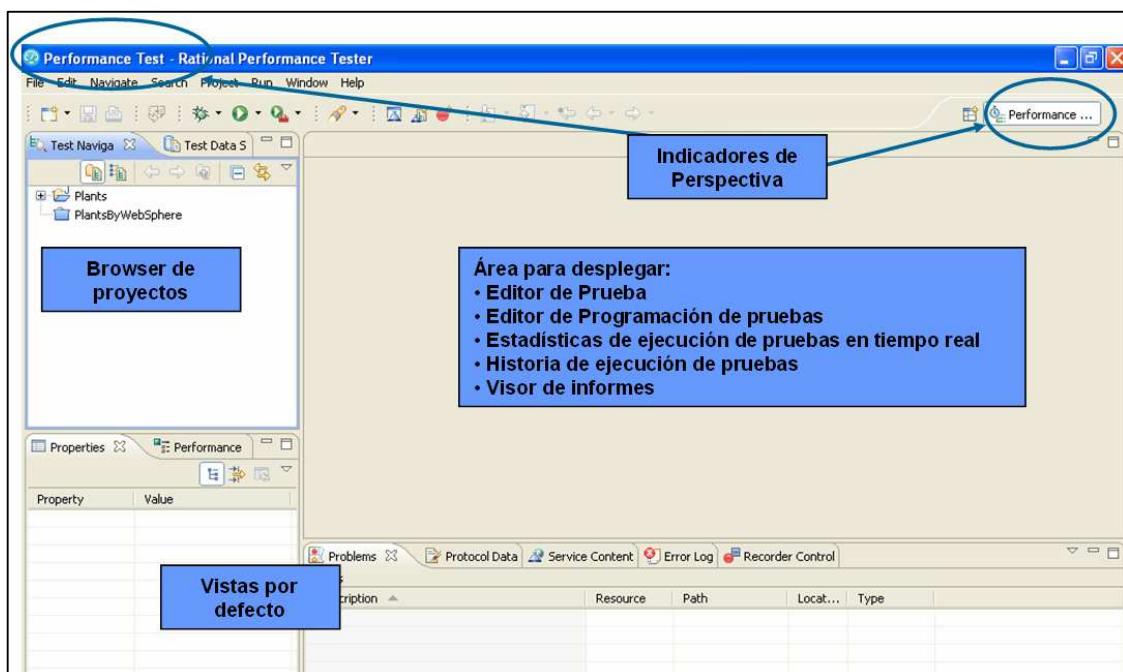
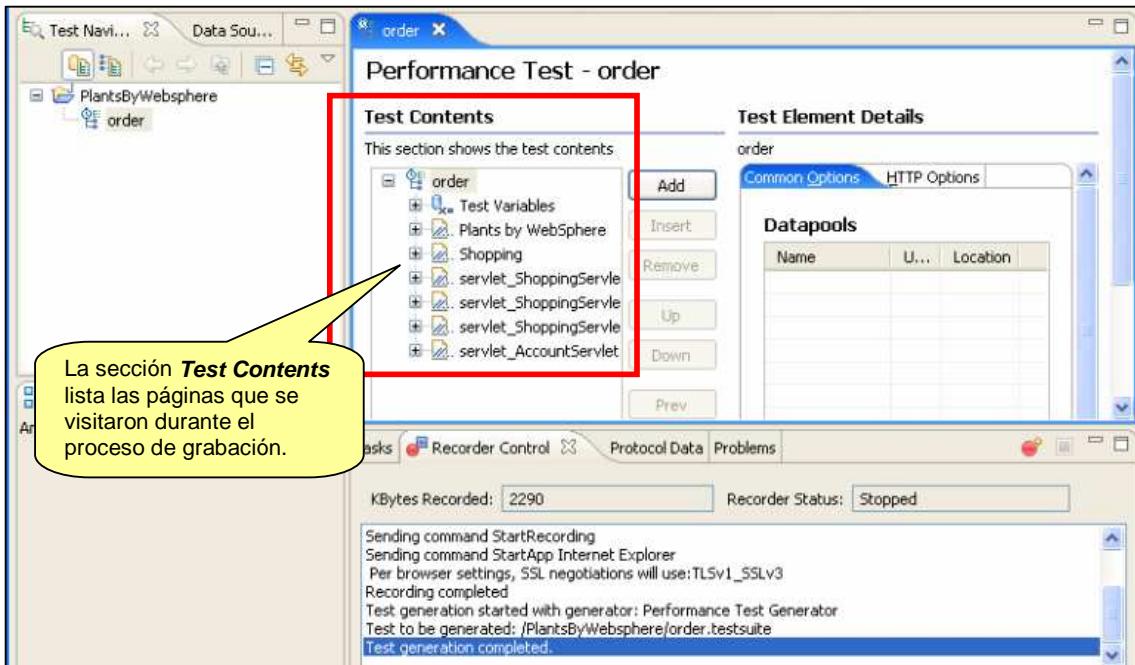


Figura 3.3. Perspectiva **Performance Test**

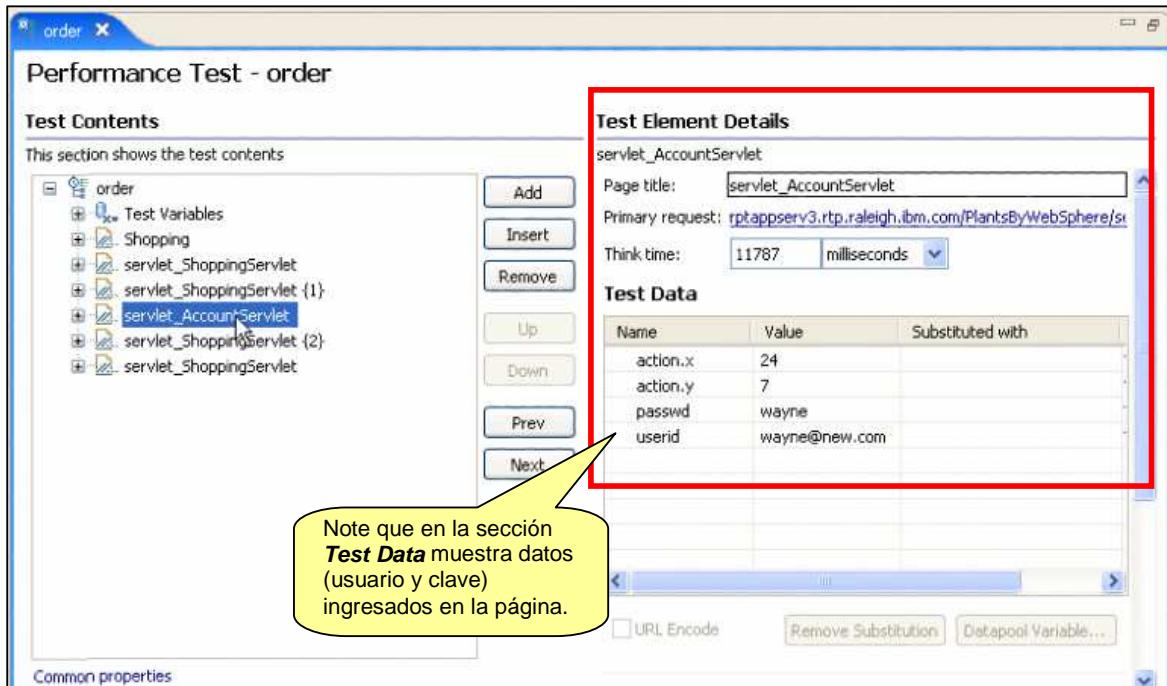
## Editor de Prueba

Al finalizar la sesión de grabación de un escenario de caso de uso se muestra el editor de pruebas con la información de las páginas visitadas.



**Figura 3.4. Vista de la prueba al finalizar el proceso de grabación**

Al seleccionar una página de la sección **TestContents**, en la sección **Test Elements Details** se muestra su información detallada.



**Figura 3.5. Información detallada de una página de prueba**

Al desplegar una página de prueba, se visualiza las solicitudes que se generaron desde esta página.

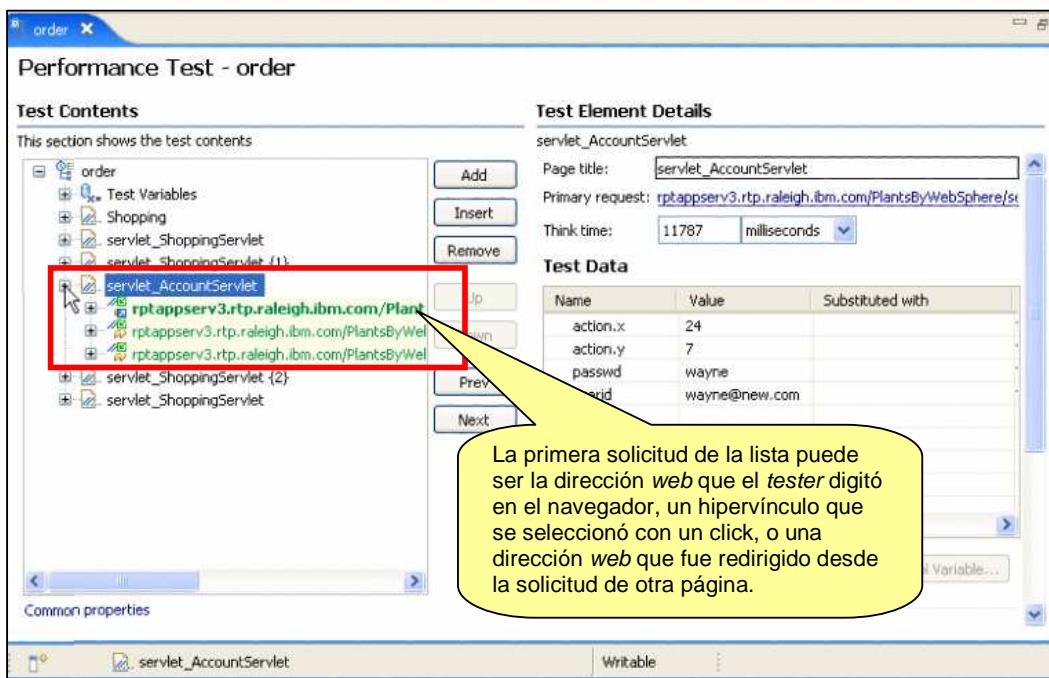


Figura 3.6. Solicitudes de la página de prueba

También se puede visualizar la respuesta generada para la solicitud de una página.

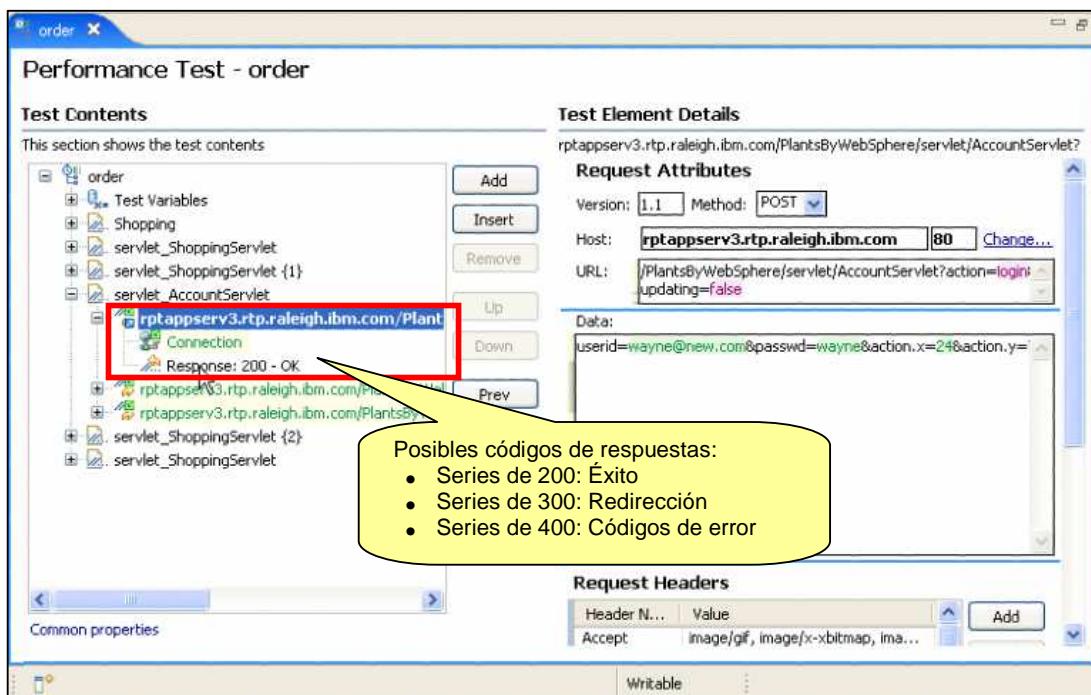


Figura 3.7. Respuesta de la solicitud seleccionada

## Editor de Programación de Pruebas

En el editor de programación de pruebas se diseña la carga de trabajo a fin de emular con precisión las acciones de los usuarios individuales.

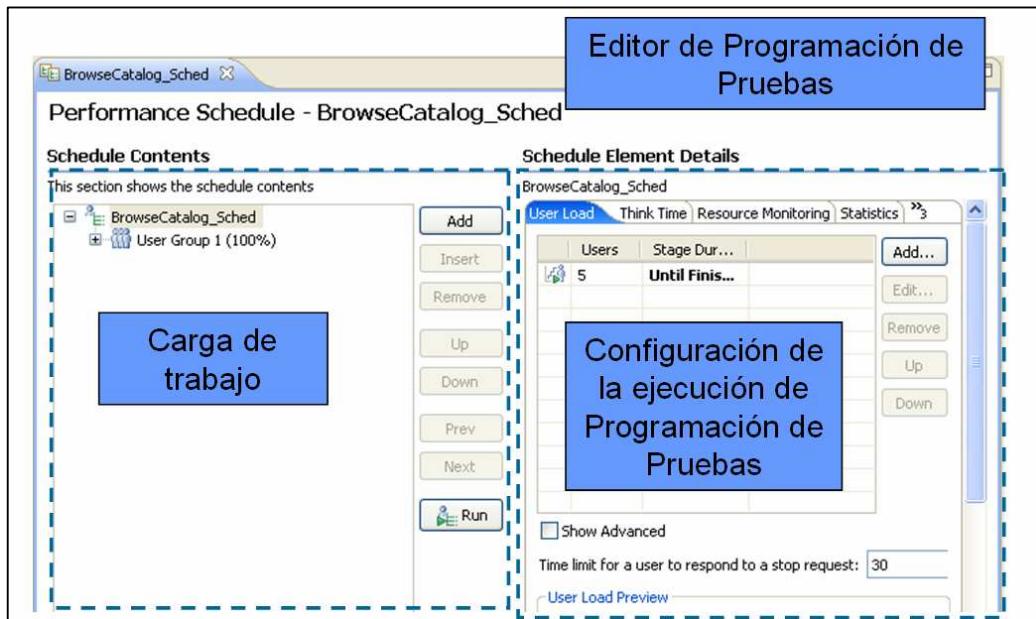


Figura 3.8. Editor de programación de Pruebas

## Estadísticas de Ejecución de Pruebas en Tiempo Real

La vista **Performance Test Runs** muestra los resultados de la prueba y es donde se puede acceder para visualizar los informes. Esta vista también controla los siguientes elementos durante la ejecución de la prueba:

- Nivel de Log: Es un indicador de la cantidad de registros por hacer. Cuanto más alto sea el nivel, mayor información será registrada.
- Usuarios virtuales: Agregar o restar el número de usuarios virtuales durante la ejecución de la prueba.
- Detener ejecución de la prueba: Detener la prueba.

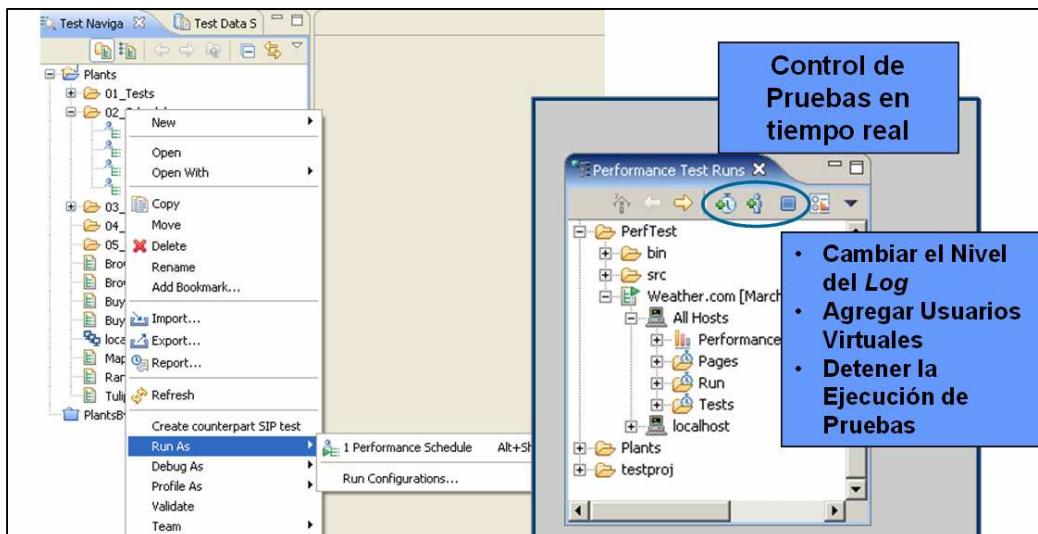


Figura 3.9. Vista Performance Test Runs

Los informes en tiempo real están disponibles durante la ejecución de la prueba. Son idénticos a los informes por defecto que se pueden acceder después de la ejecución de pruebas. Puede seleccionar una de las pestañas, que contienen diversos informes, desde la parte inferior del monitor de pruebas. Tal como se indica en la siguiente figura.

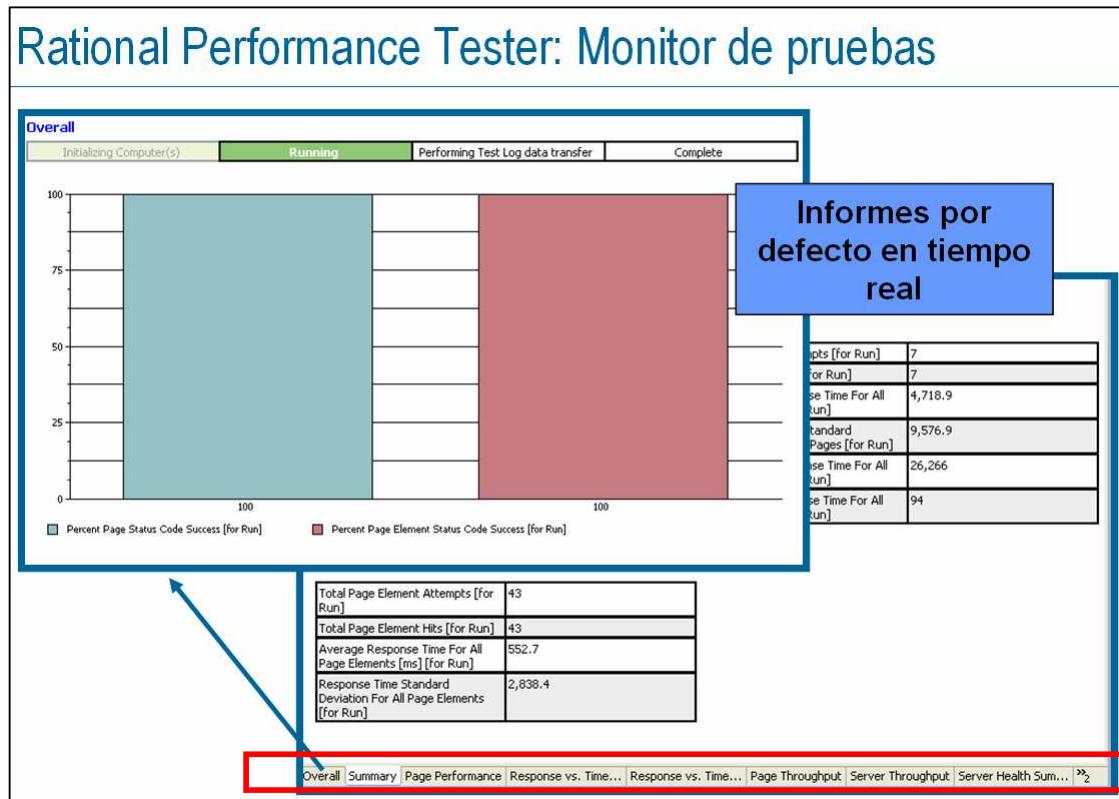


Figura 3.10. Monitor de pruebas durante el proceso de ejecución

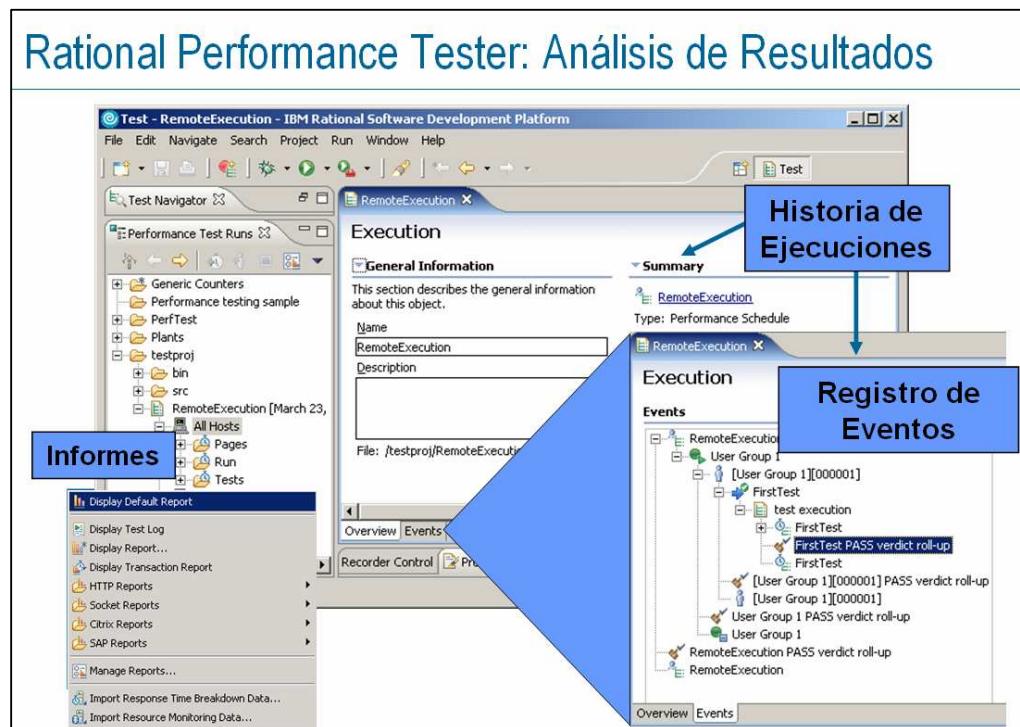


Figura 3.11. Informe de resultados después del proceso de ejecución

### 3.1.2 Características y beneficios de Rational Performance Tester

A continuación se listan las características y los beneficios conseguidos gracias a las características del RPT.

- **Múltiples técnicas, poco intrusivas de grabación capturan la comunicación cliente servidor involucrando los protocolos tanto HTTP/HTTPS como SQL:** La grabación del script ocurre en forma no obstructiva mientras el *tester* interactúa con la interfaz de usuario de la aplicación cliente. La grabación de la comunicación resultante permite la coexistencia de múltiples protocolos en un único script - en lugar de tener que fragmentarlo en grabaciones de cada protocolo en scripts separados - simplificando la documentación y modificación del test.
- **Los filtros incluidos de correlación de datos detectan datos variables y preparan los tests para la generación de de carga test basados en los datos:** Para asegurar la equivalencia de las cargas de usuario simulados a las del mundo real, RPT contiene un utilitario de correlación de datos que detecta automáticamente los datos variables usados durante el proceso de grabación. Mediante su "datapool wizard", los *testers* de la performance pueden generar datos variables (como nombres, números de tarjetas de crédito, direcciones, etc) los mismos serán accedidos durante la ejecución del test, asegurando que cada usuario simulado presente datos únicos el ambiente servidor. Sin este paso, los tests de performance no reproducirían el ambiente de producción fallando en consecuencia al tratar de exponer los verdaderos problemas de escalabilidad.
- **Un rico lenguaje y una librería de funciones de test permiten a los testers expertos una extensa customización de los scripts que requieren de algoritmos avanzados de análisis:** El test script del **Performance Tester** usa un lenguaje poderoso, similar al C que permite a los *testers* avanzados de performance el implementar una variedad ilimitada de técnicas de personalización para facilitar análisis y manipulación de los datos durante la ejecución del test. Tiene incluidos un conjunto de funciones predefinidas particularmente útiles para técnicas avanzadas de *testing* de carga.
- **Tiene un scheduler de carga de test que puede ser personalizado completamente para crear modelos de perfiles reales de carga de usuarios altamente precisos:** Antes que se pueda ejecutar el test, los *testers* de performance deben determinar con precisión el modelo de los diversos grupos de usuarios que usarán el sistema en la realidad. **Rational Performance Tester** contiene un *scheduler* avanzado de carga de trabajo para permitir este modelado, facilitando que los *testers* alcancen a replicar el mundo real de los usuarios finales. Están disponibles variadas construcciones como loops y saltos condicionales y los múltiples perfiles de usuario resultantes se pueden ejecutar en paralelo.
- **Reportes en tiempo real muestran segundo a segundo los tiempos de respuesta de usuarios y grupos de usuarios exponiendo así los cuellos de botella del sistema tan pronto**

**como ocurren:** Durante la ejecución del test, RPT muestra gráficamente vistas consolidadas de los tiempos promedio de respuesta - a lo largo del tiempo - para cada perfil de usuario simulado. Además, los *testers* tienen la opción de profundizar en la conversación en tiempo real entre cualquier instancia de usuario individual y el sistema bajo test para ver los datos intercambiados, permitiendo un análisis rápido y temprano de la degradación sospechada de la respuesta. Se dispone de reportes adicionales pos-ejecución del test para un posterior análisis de los datos respondidos - todos exportables a archivos .CSV y HTML.

- **Se pueden adquirir y correlacionar los datos detallados de los recursos servidor con los tiempos de respuesta del mismo para exponer limitaciones de performance relacionadas al hardware:** Una variedad de métricas de recursos del servidor son reunidas por el RPT durante la ejecución del test. Estas métricas se pueden mostrar lado a lado con los datos de respuesta reunidos por el servidor para analizar si la pobre performance del sistema se puede atribuir a las características del hardware (tales como el tiempo de encolado y memoria del sistema) más que al software de aplicación.

## 3.2. PRUEBAS DE RENDIMIENTO EN RPT

Existen muchos tipos de pruebas de rendimiento, por ejemplo:

- De carga: Determina los tiempos de respuesta de todos los procesos de negocio críticos y las transacciones importantes de la aplicación. Esta prueba puede mostrar el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga.
- De volumen: Determina el rendimiento y capacidad de *drivers* asociados a un proceso o transacción específica.
- De stress: Determina la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.

Las pruebas de rendimiento con RPT pueden ser automatizadas. Para ello, debe grabar los escenarios a probar, programar las pruebas y ejecutarlas a fin de analizar los informes obtenidos según las necesidades del equipo. Estos procesos se detallan a continuación con un caso práctico.

## CASO PRÁCTICO: DESARROLLO DE UNA PRUEBA DE RENDIMIENTO

La aplicación sobre la cual se realizarán las pruebas de rendimiento es PLANTS by WebSphere.

Plants by WebSphere, es una aplicación e-commerce basada en Web que da soporte a la venta de semillas, plantas y herramientas de jardinería.

### 1. Arquitectura y Background

#### Funcionalidad del Cliente:

- Validar usuario (Login)
- Buscar en catálogo
- Registrar una orden

#### Funcionalidad del Administrador:

- Gestionar Órdenes

### 2. Tecnología

Desarrollado con herramientas IBM WebSphere

Front end, Frame-based application browser GUI

Back end, Base de Datos IBM DB2

### 3. Estructura de Datos

Tabla	Descripción y Contenido
<b>CUSTOMER</b>	Contiene información de envíos del cliente Data: name, address, phone number, password, customer ID(llave), etc.
<b>INVENTORY</b>	Contiene información de ítems en el catálogo de productos. Data: name, description, image, category, inventory ID(llave), etc.
<b>ORDER1</b>	Contiene información de la entrega y cuentas de los clientes. Data: sale date, credit card information, shipping information, order ID(llave), etc.
<b>ORDERITEM</b>	Contiene información de los productos. Data: item info, price, quantity, etc.
<b>IDGENERATOR</b>	Usado por el generador de ID secuenciales.
<b>BACKORDER</b>	Contiene información del estado de la orden, incluyendo la fecha y en cuando se reduce el inventario, y la fecha de la orden.

#### 4. Perfiles de usuario:

- **Browsers** navegan por el site viendo sus productos de interés pero se marchan antes de comprar
- **Shoppers** ubican y agregan productos a sus carritos, pero se van del site antes de comprar
- **Buyers** ubican y agregan productos al carrito y completan el proceso de compra
- **Inventory managers** verifican las órdenes de productos y colocan las órdenes de inventario para reestablecerlo

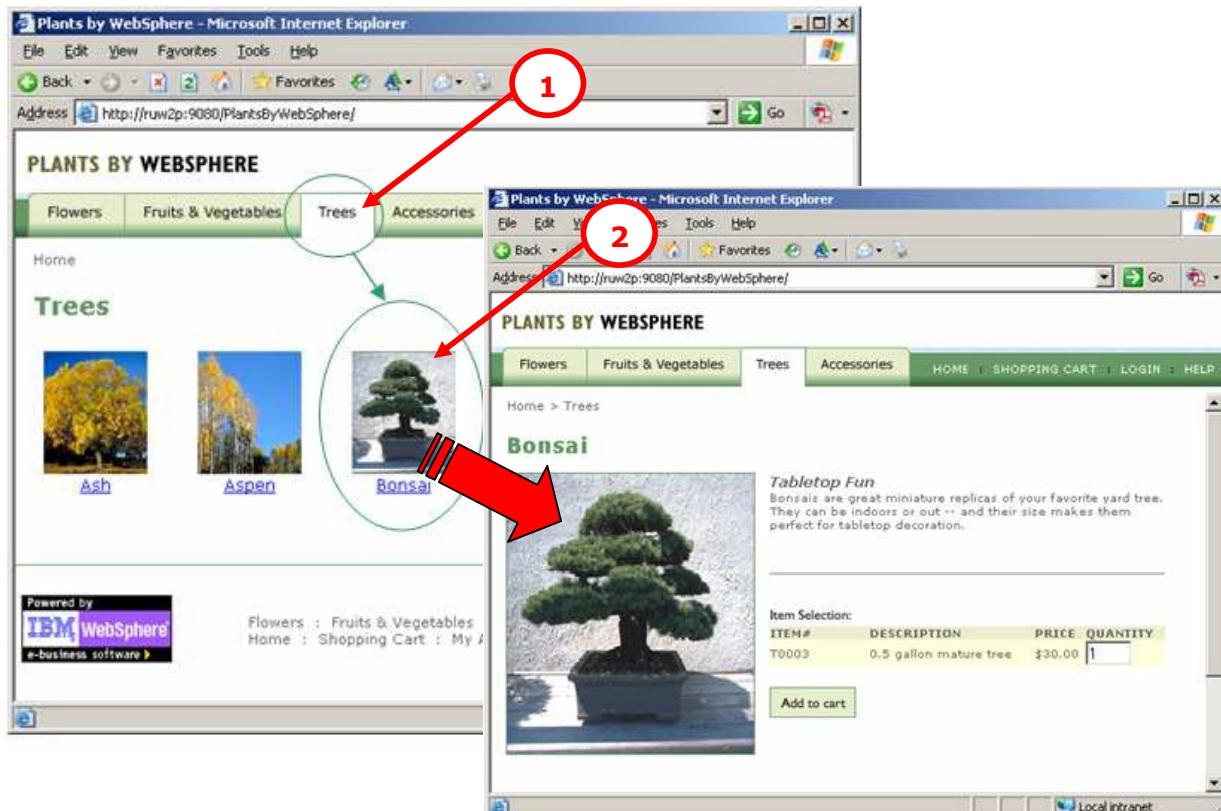


#### 5. Transacciones que se grabarán

- Buscar en catálogo
- Registrar una orden

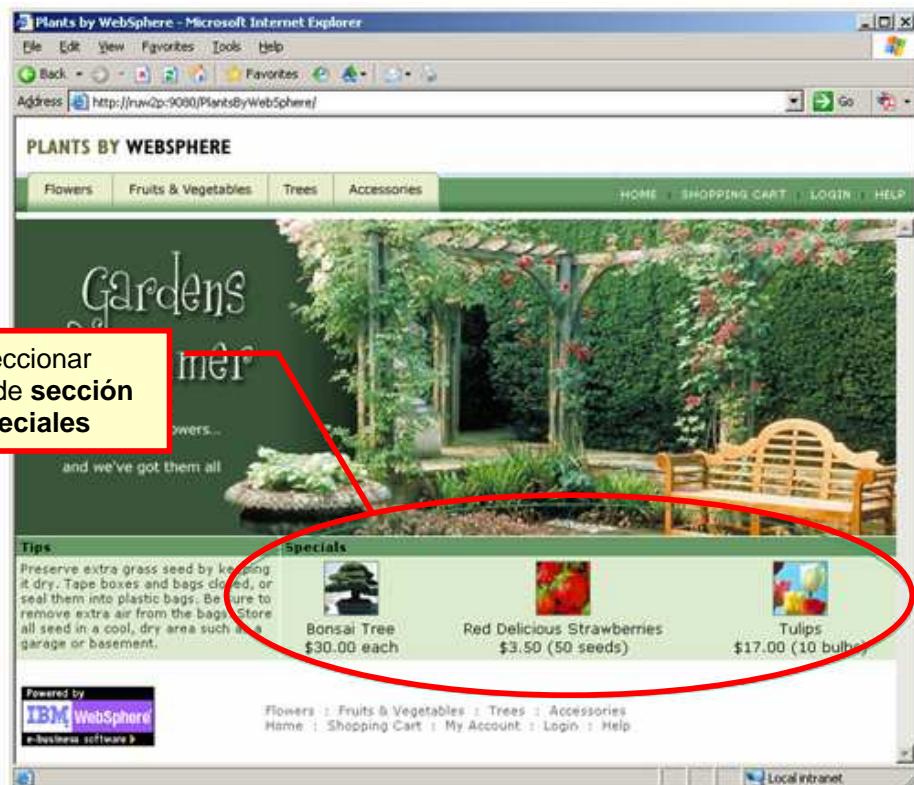
##### 5.1. Buscar en catálogo

- Seleccione una categoría para seleccionar un ítem y luego, seleccione su imagen o nombre para mostrar detalles.

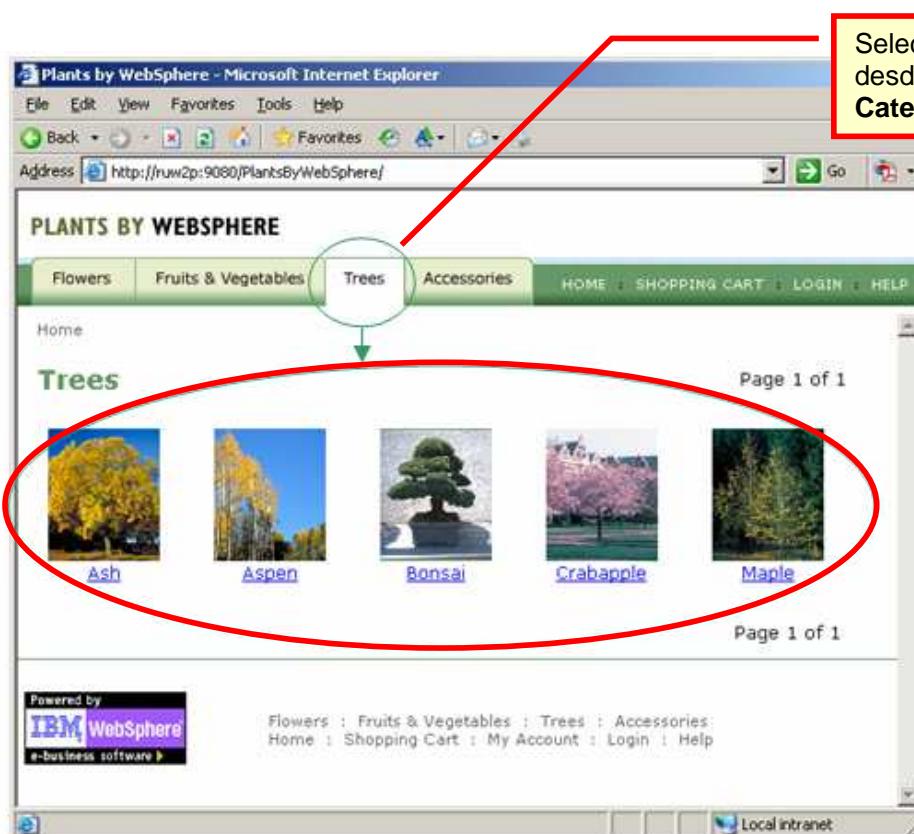


## 5.2. Registrar una orden

- Localice y seleccione un ítem de interés: desde la sección Especiales o desde alguna categoría.

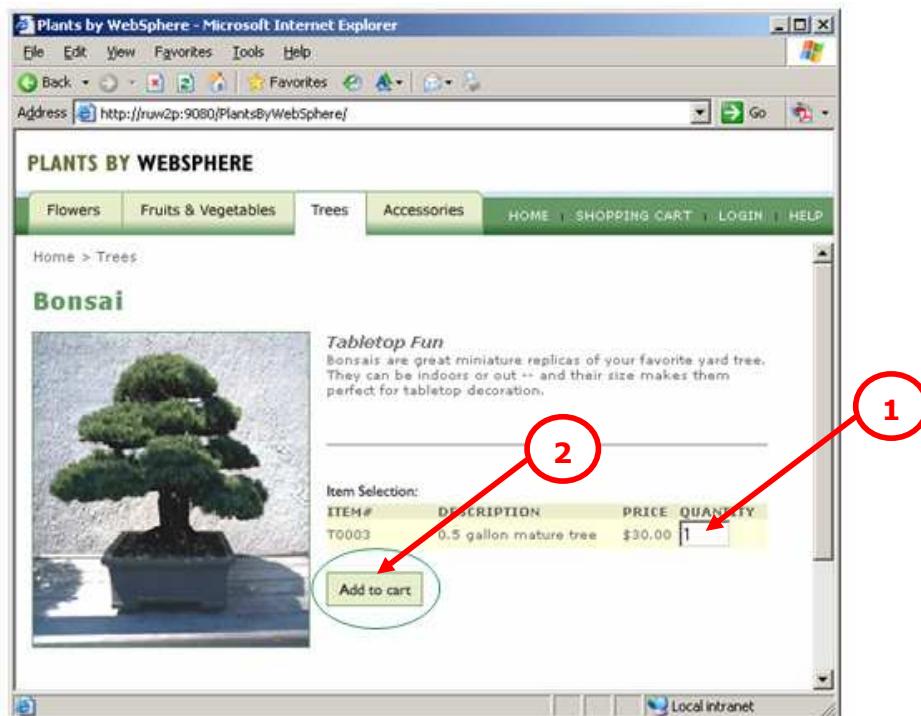


Seleccionar  
desde sección  
Especiales

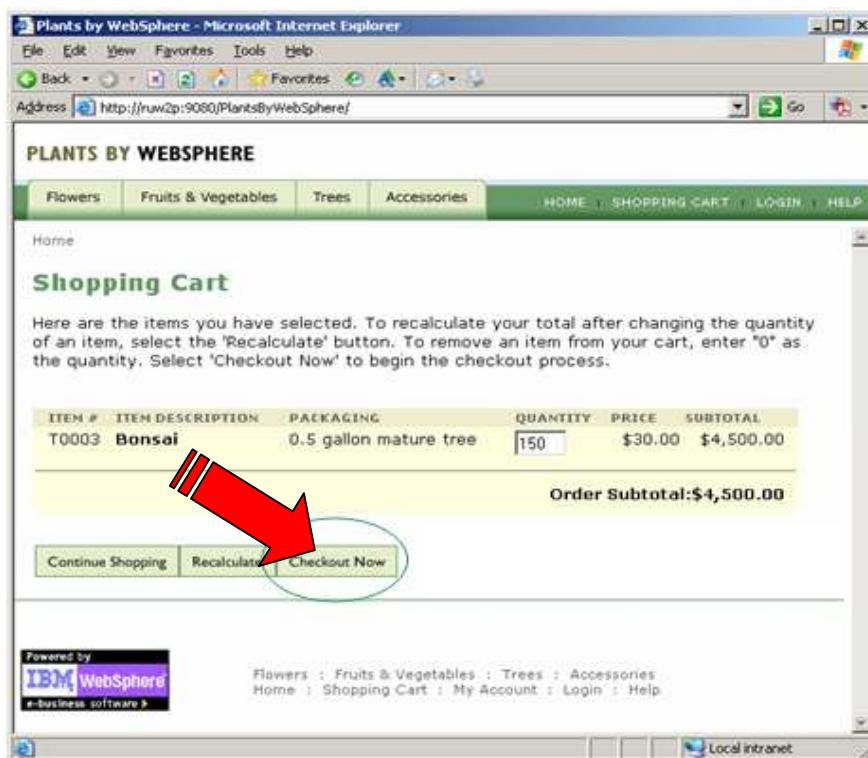


Seleccionar  
desde alguna  
Categoría

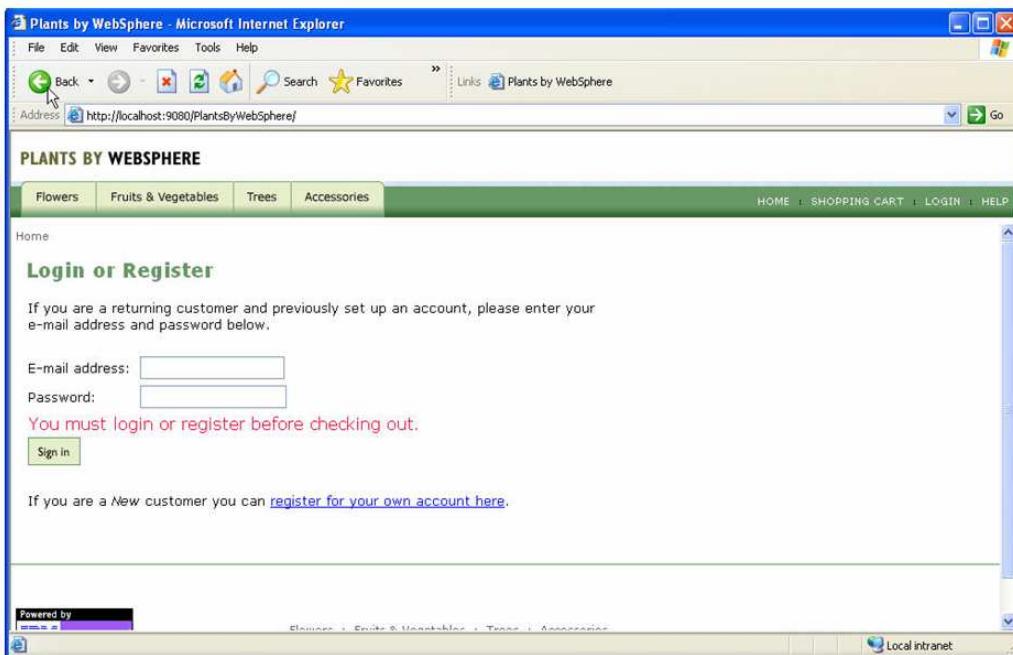
- De la página de detalle del ítem, especifique la cantidad y pulse ADD to cart.



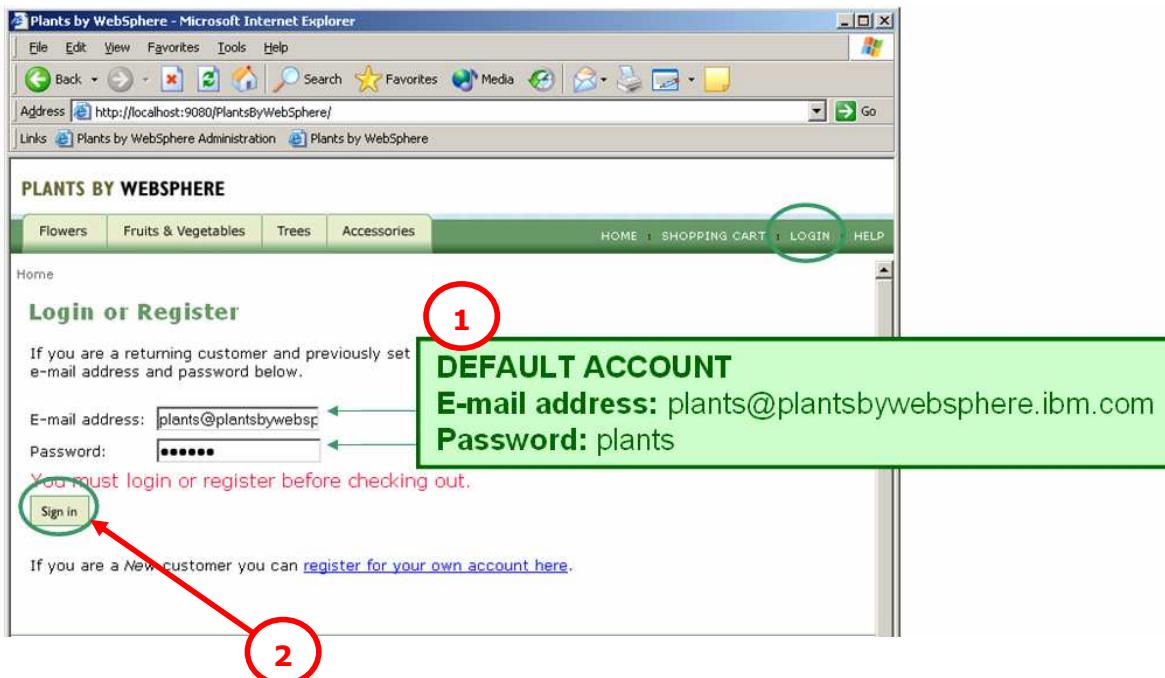
- Cuando la compra esté lista pulse **Checkout Now**



- La página Login o Register se muestra:



- Ingrese al sistema con la cuenta por default. Puede crear una nueva cuenta si lo desea, desde la opción **register for your own account here**.



- Complete el formulario de Checkout, luego pulse **Continue**.

**PLANTS BY WEBSPHERE**

Flowers Fruits & Vegetables Trees Accessories HOME SHOPPING CART LOGIN HELP

Home > Shopping Cart

### Checkout

Enter the billing and shipping information for your order below. Select 'Continue' to review and place your final order.

Billing Address	
Full Name	* student student
Address Line 1	* 1 Main St
Address Line 2	
City	* Somewhere
State	* MA
Zip Code	* 12345
Phone (daytime)	* 888-555-1212

Shipping Information

Check here if the shipping address is the same as the billing address.

Expiration Year	* 2002
Cardholder Name	* [empty]

**Continue**

- Revise el detalle de las órdenes, y luego pulse **Submit Order**

**PLANTS BY WEBSPHERE**

Flowers Fruits & Vegetables Trees Accessories HOME SHOPPING CART LOGIN HELP

Home > Shopping Cart > Checkout

### Review Your Order

Review your order below and select 'Submit Order' at the bottom to place your order. You can also add more items to your order by selecting 'Continue Shopping'.

Order Information		
ORDER TOTAL	SHIPPING ADDRESS	BILLING ADDRESS
\$4,504.99	student student 1 Main St  Somewhere, MA 12345 888-555-1212	student student 1 Main St  Somewhere, MA 12345 888-555-1212

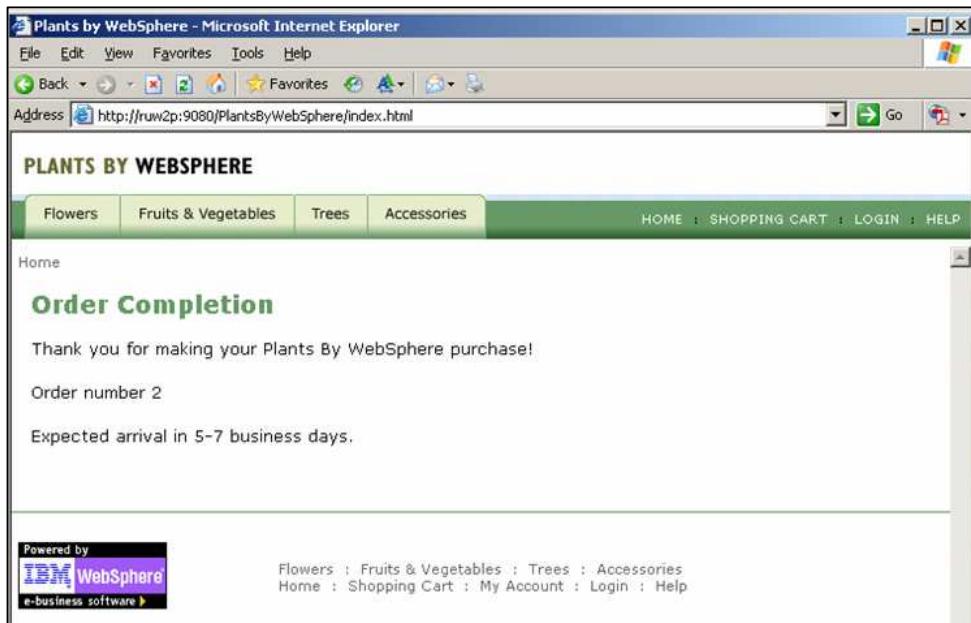
**Order Details**

ITEM #	ITEM DESCRIPTION	PACKAGING	QUANTITY	PRICE	SUBTOTAL
T0003	Bonsai	0.5 gallon mature tree	150	\$30.00	\$4,500.00

Order Subtotal: \$4,500.00  
Shipping, Standard Ground: \$4.99  
Order Total: \$4,504.99

**Continue Shopping** **Submit Order**

- Verifique la página de la Orden completa que garantiza que la transacción se ha completado.



## 6. Preparación para la grabación de las pruebas

### 1. Preparar el entorno de pruebas

- Configurar Herramientas de prueba
- Configurar entorno para la ejecución de la aplicación objetivo

### 2. Grabar pasos detallados del usuario para la prueba: Registrar una orden

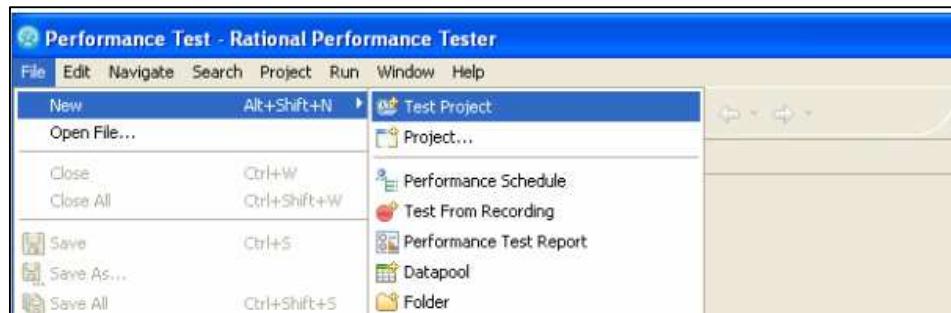
1. Página de Inicio
  - Seleccione Bonsai Tree
2. Página de Descripción del producto
  - Pulse ADD to CART
3. Página de Shopping cart
  - Confirme 1 árbol Bonsai
  - Pulse Checkout now
4. Página de Inicio de Sesión
  - Ingresar la cuenta de email por default y el password
  - Pulse en Sign in
5. Página de Checkout
  - Especifique el embarque y la información de facturación
  - Active el checkbox para especificar que se use la información de la cuenta por defecto y la dirección de facturación
  - Ingrese el número de la tarjeta de crédito 1234 1234 1234
  - Cambie el año de expiración a 2011
  - Pulse continue
6. Revise la página de órdenes
  - Pulse Submit Order
7. Página Order Confirmation
  - Confirme que el número de orden se ha incrementado

## 7. Grabación de las pruebas

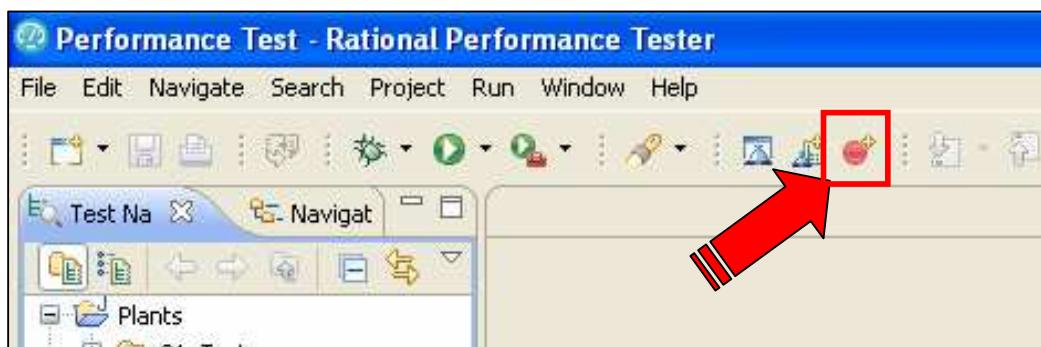
El siguiente gráfico muestra el proceso general:



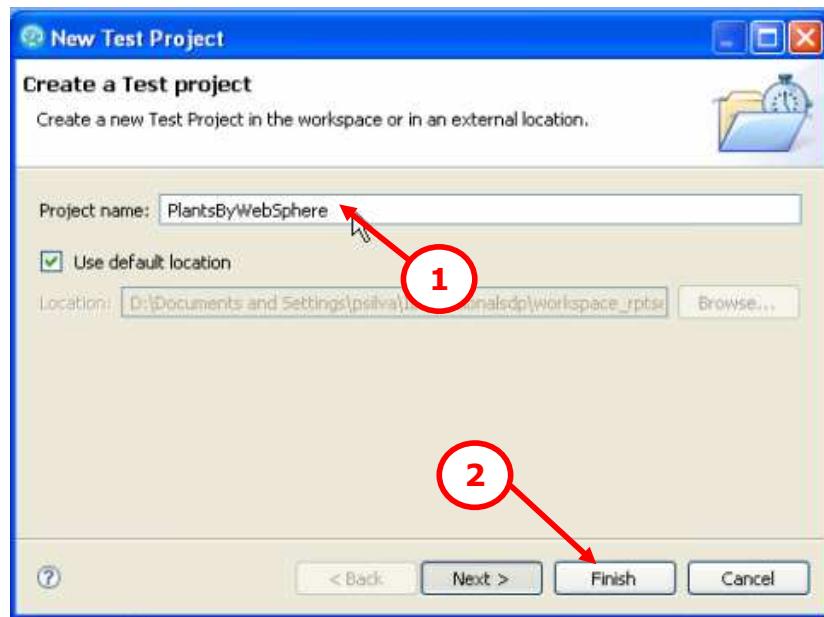
1. Seleccione **File > New > Test Project** para crear el proyecto que contendrá las pruebas de la aplicación.



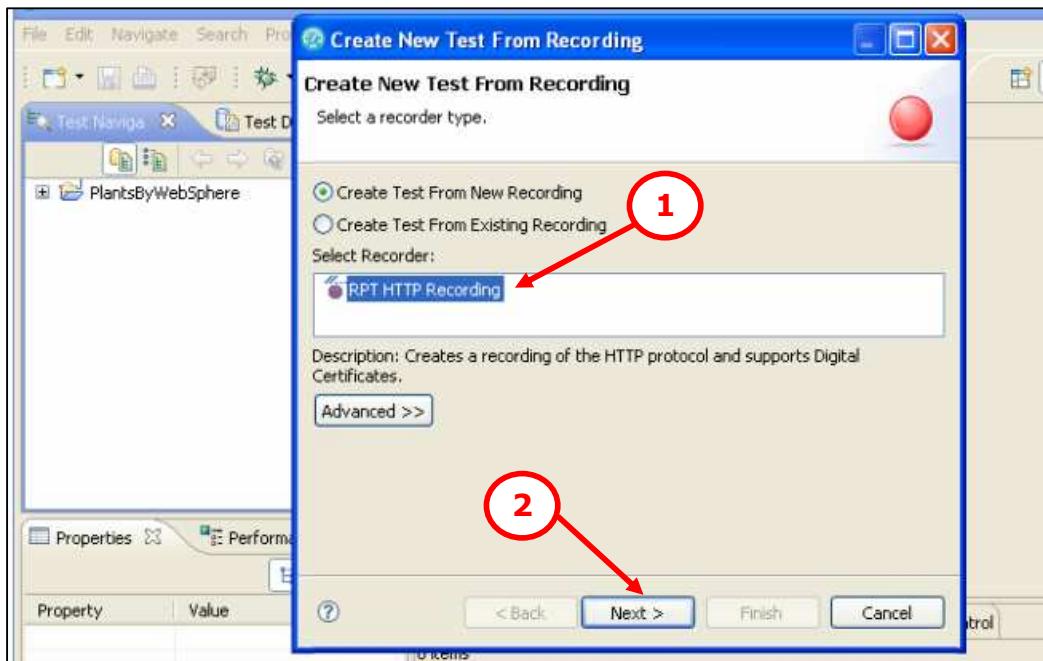
También puede crear el proyecto al seleccionar la opción de grabación desde la barra de herramientas:



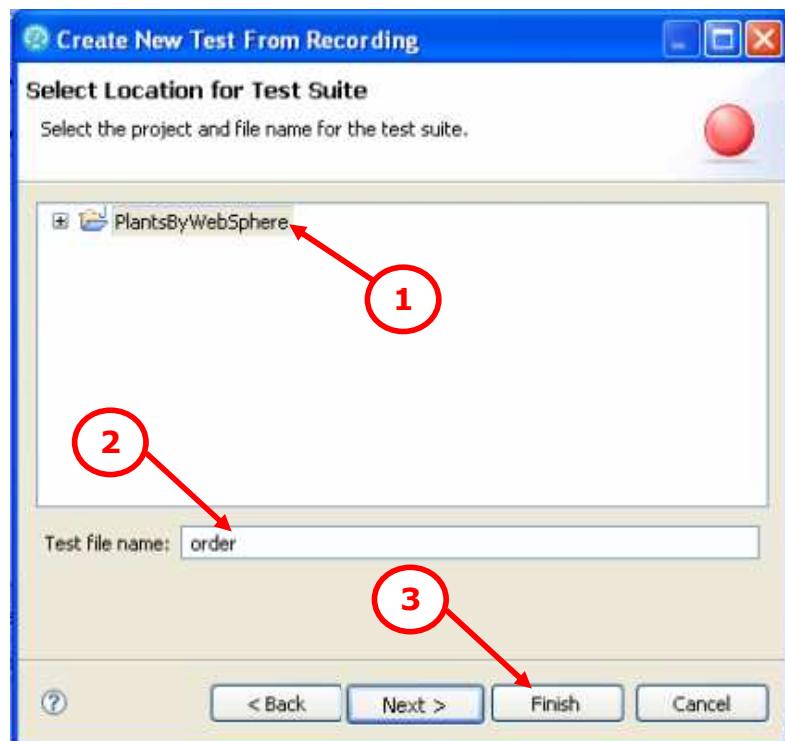
2. Desde la ventana “Nuevo Proyecto de Prueba” edite el nombre del proyecto y luego seleccione **Finish**.



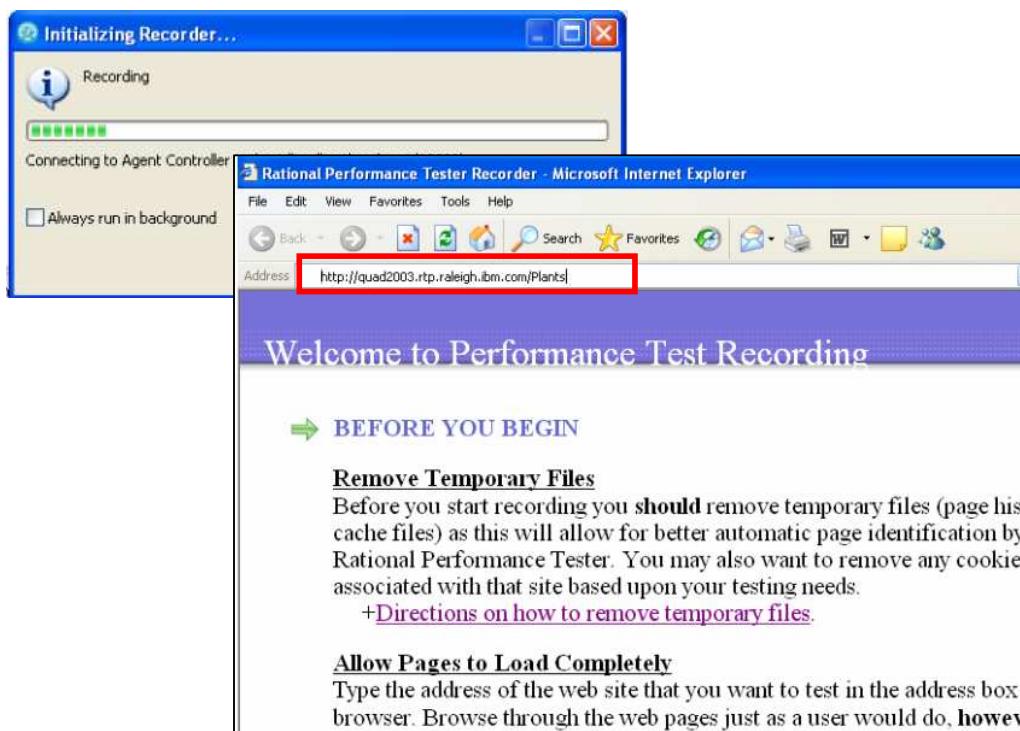
3. Note que en el explorador de pruebas se ha creado su proyecto. Seleccione **RPT http Recording** para iniciar la grabación, luego **Next**.



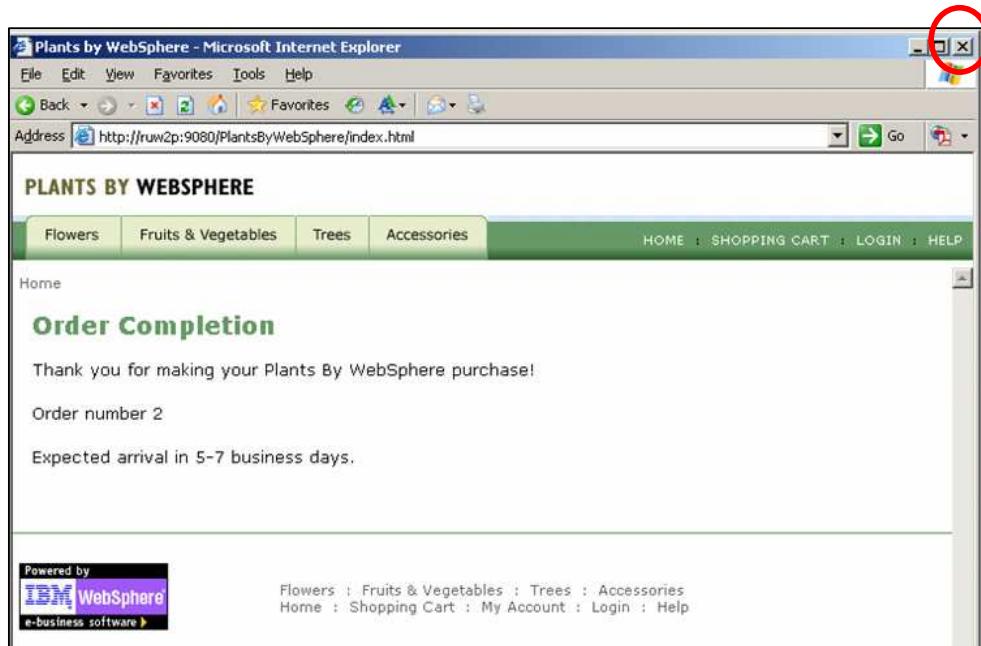
4. Ahora, seleccione el proyecto, edite el nombre de su primera prueba y seleccione **Finish**.



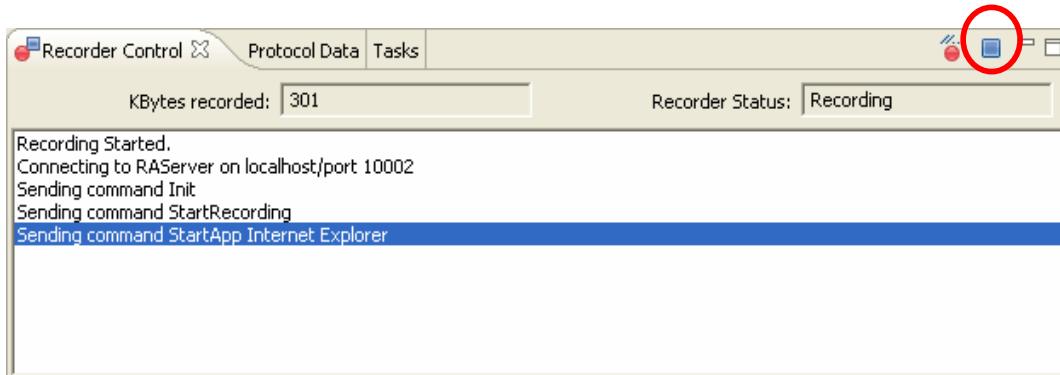
5. A continuación, se mostrará el indicador de inicio de grabación y luego el browser de Internet en donde escribirá el URL de su aplicación y **Enter** para empezar a grabar.



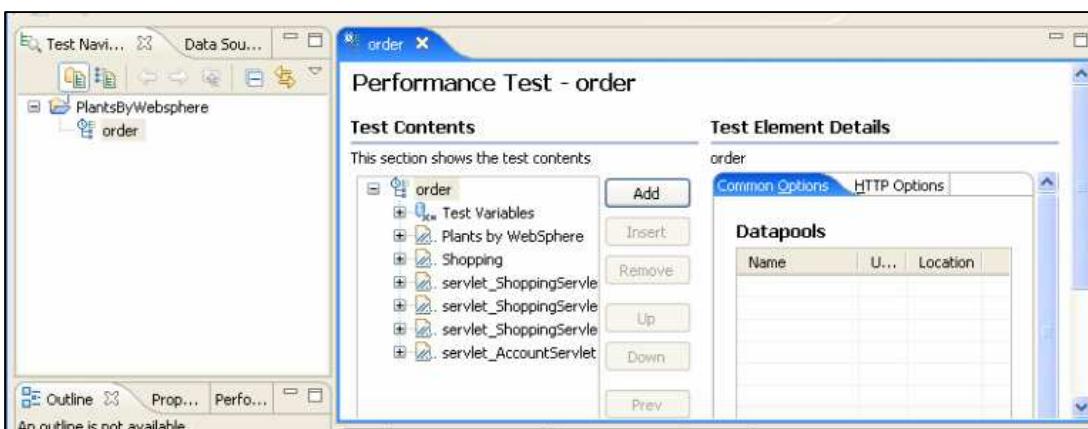
- Después de interactuar con su aplicación para grabar el escenario “**Registrar una orden**”, cierre el browser de Internet para detener el proceso de grabación.



También puede detener el proceso de grabación desde la opción **Stop Recording** de la vista **Recorder Control**.



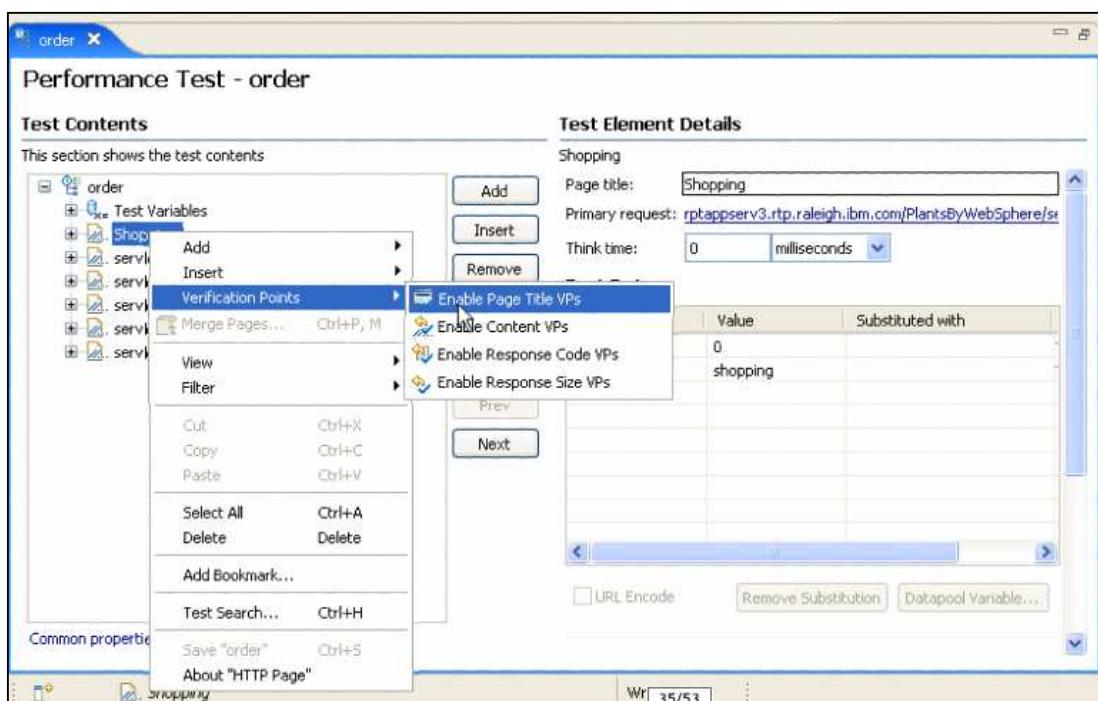
- Desde el editor de pruebas podrá visualizar la prueba generada.



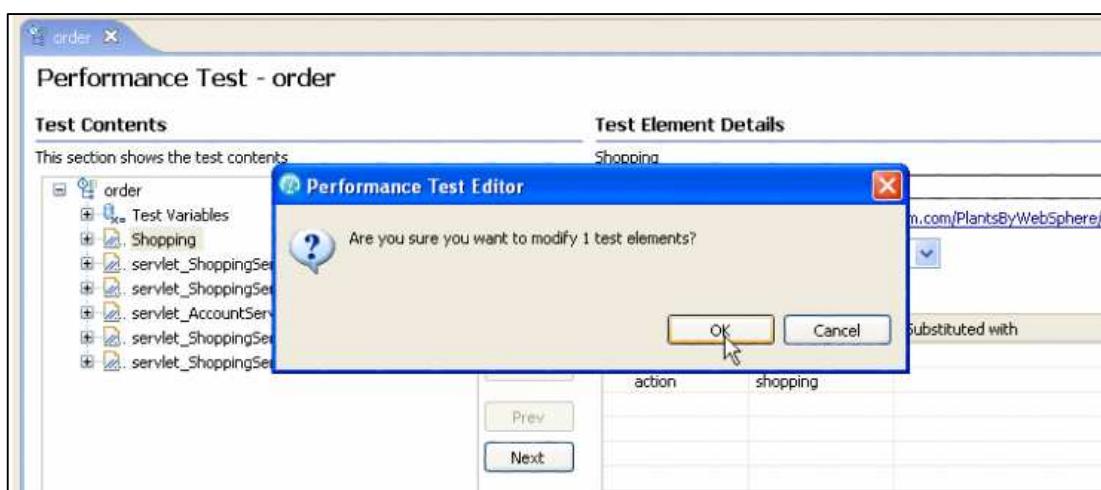
## 8. Edición de Pruebas

En este punto, aprenderá a añadir puntos de verificación y pool de datos a una prueba. Los puntos de verificación comprueban si se da el comportamiento esperado durante una ejecución. El pool de datos permite variar los valores de lo registrado en la prueba.

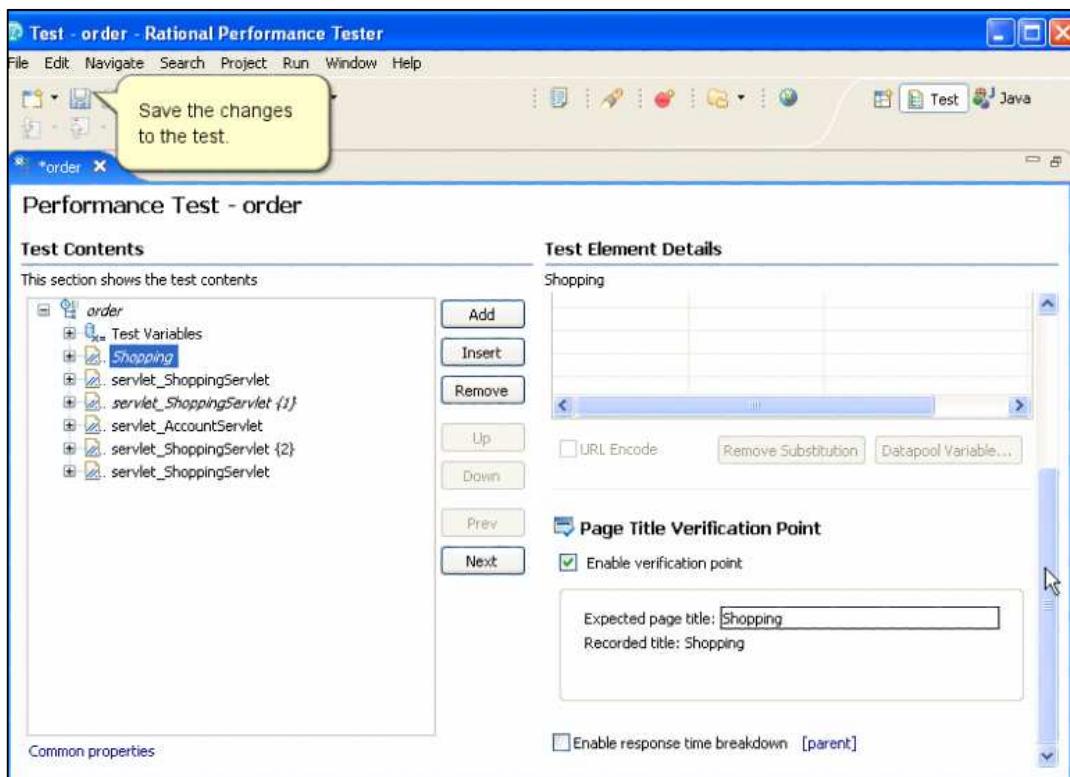
- Click derecho sobre la página a la cual agregará un punto de verificación (PV) > **Verification Points > Enable Page Title VPs.**



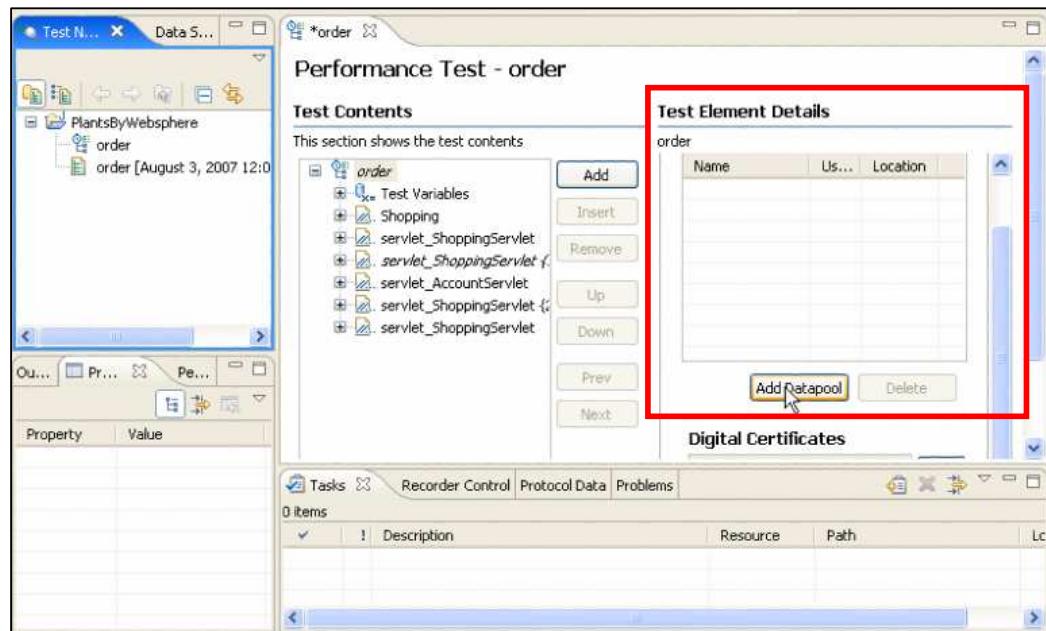
- Confirme la agregación del PV.



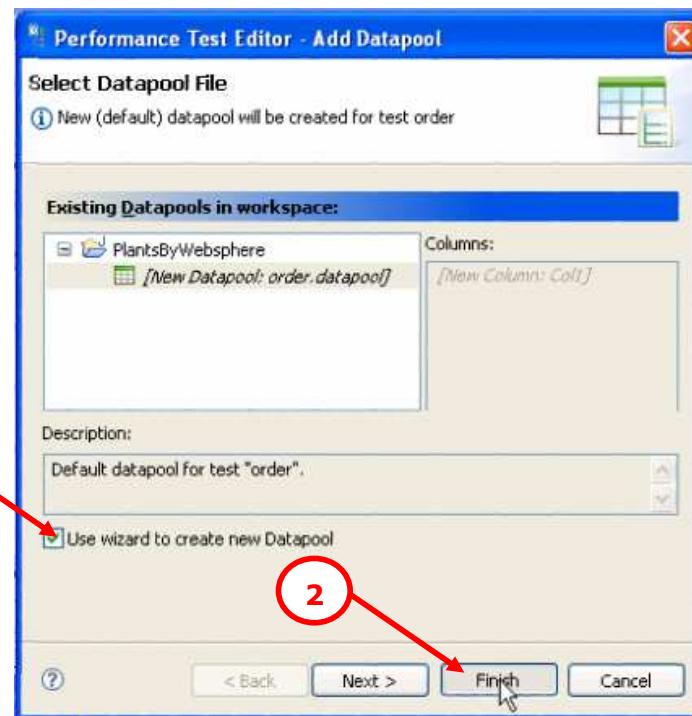
3. A continuación, se mostrará la sección del PV agregado; guarde los cambios.



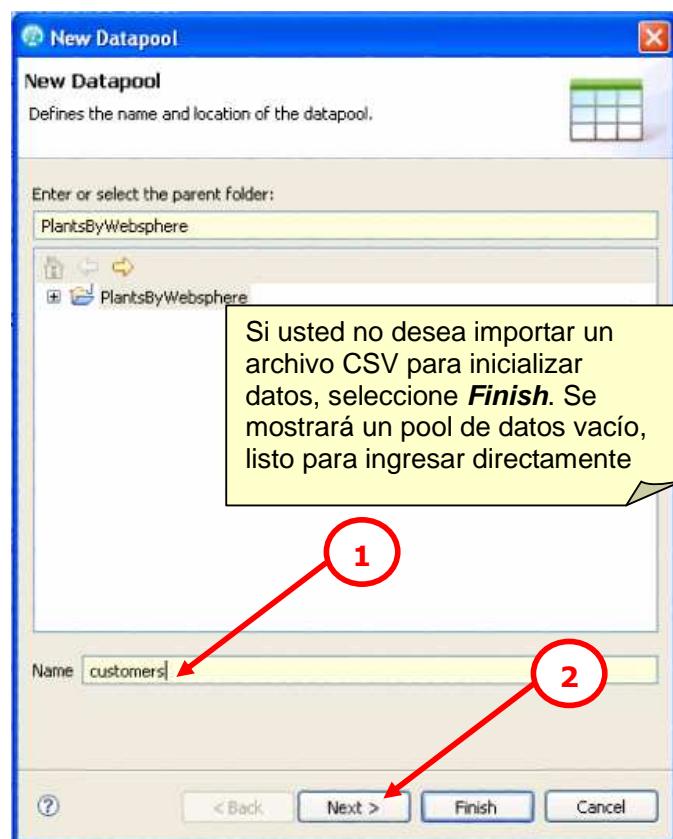
4. Ahora, importará un archivo CSV para crear un pool de datos. Para ello, desde la sección **Datapools** seleccione **Add Datapool**.



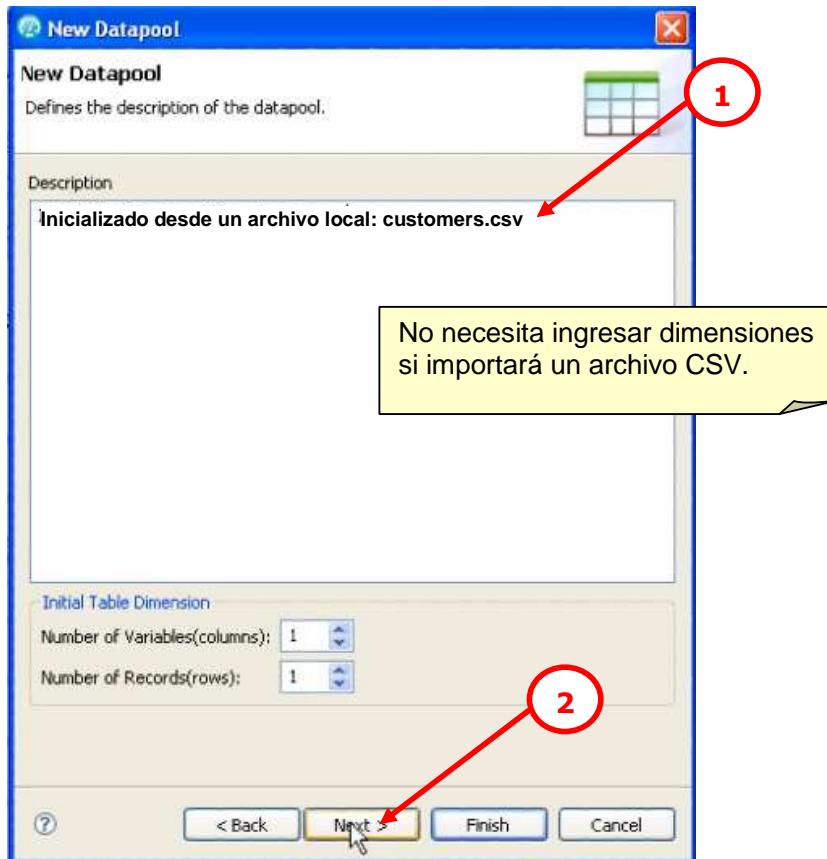
5. A continuación, active el *checkbox* para crear un nuevo pool de datos. Luego, pulse **Finish**.



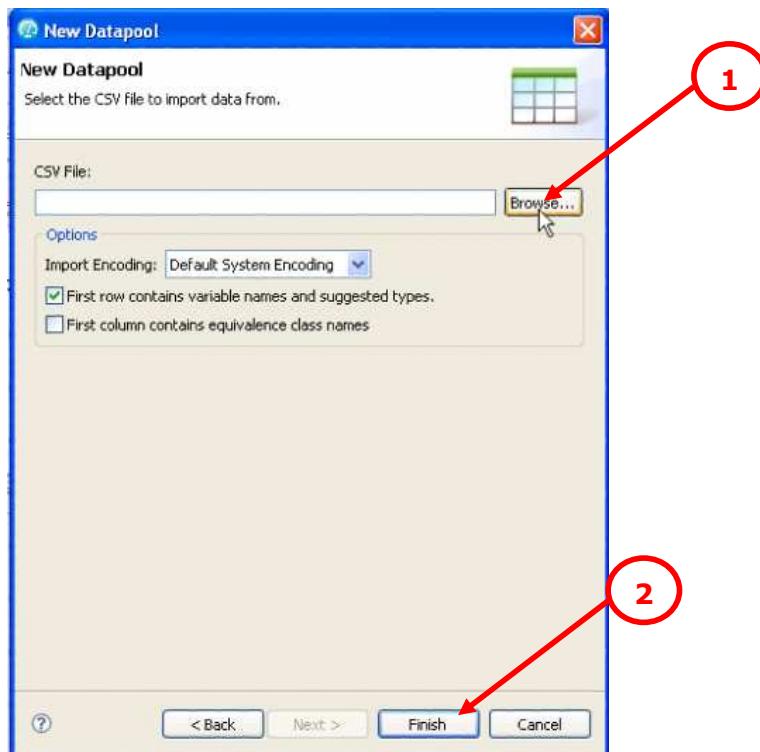
6. Desde la ventana **New Datapool** ingrese un nombre para el nuevo pool de datos. Luego, pulse **Next**.



7. A continuación, edite una descripción y luego, pulse **Next**.



8. Ubique el archivo CSV.



9. A continuación se mostrará el pool de datos importado.

The screenshot shows the Rational Performance Tester interface. In the center, there is a table titled "Datapool" with two columns: "userid:" and "passwd:". The data is as follows:

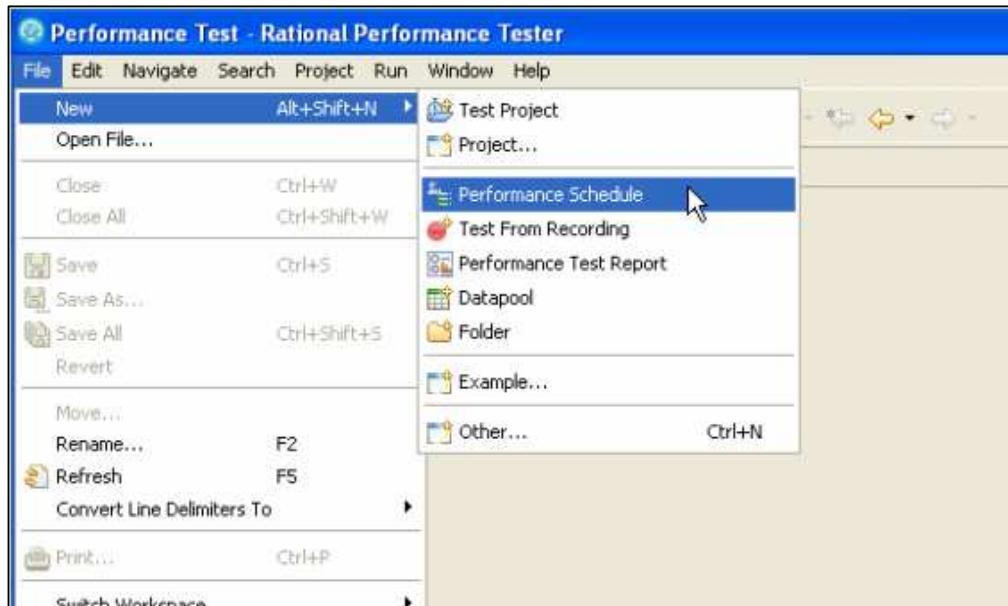
userid:	passwd:
jennifer@new.com	jennifer
1 john@new.com	john
2 melinda@new.com	melinda
3 nick@new.com	nick
4 pat@new.com	pat
5 phil@new.com	phil
6 richard@new.com	richard
7 wayne@new.com	wayne

A yellow callout box points to the table with the text: "El **datapool** importado contiene los valores para 8 usuarios: usuario y password."

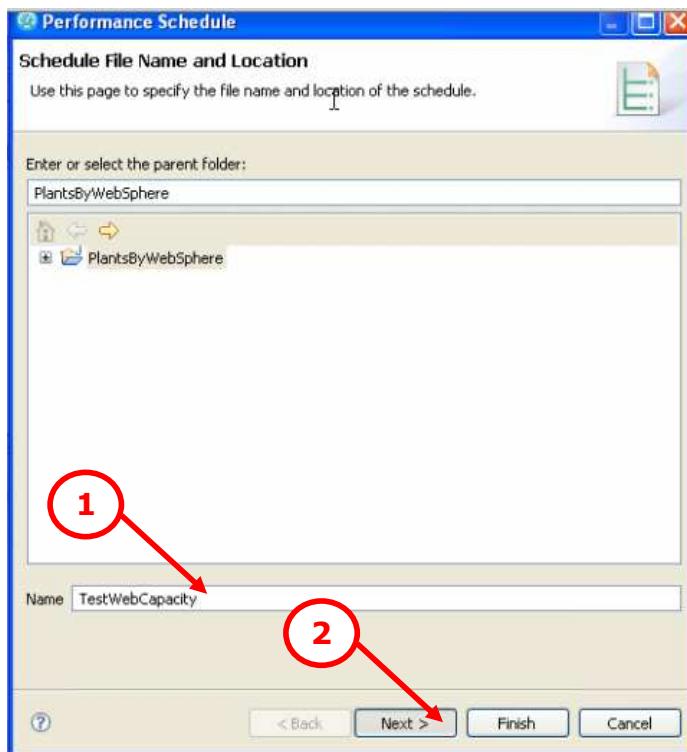
## 9. Programación de Pruebas

La programación de una prueba representa una carga de trabajo a fin de realizar las pruebas de rendimiento automatizadas. Los pasos se describen a continuación.

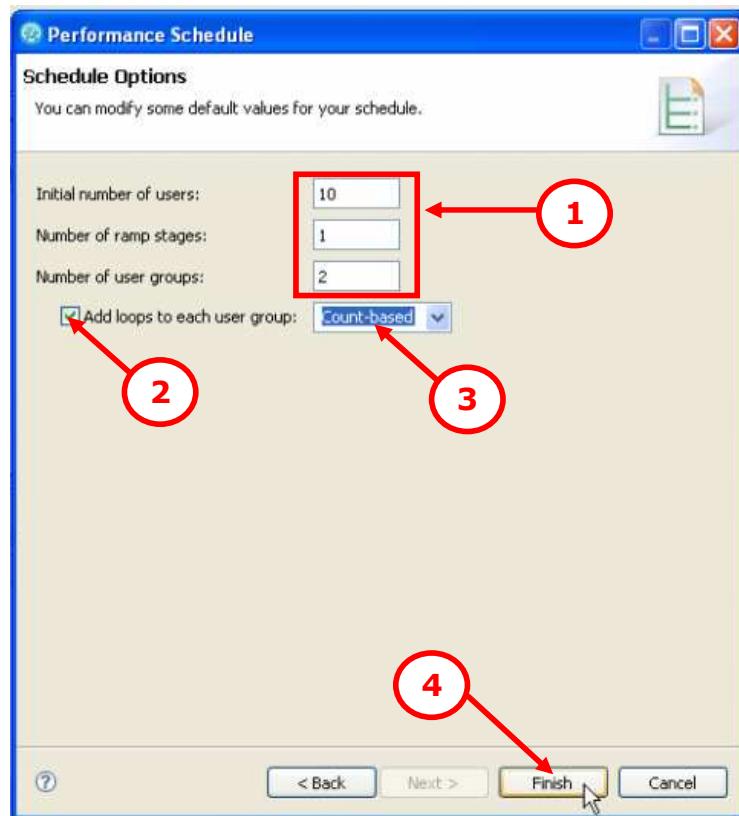
1. Seleccione **File > New > Performance Schedule** para programar una prueba.



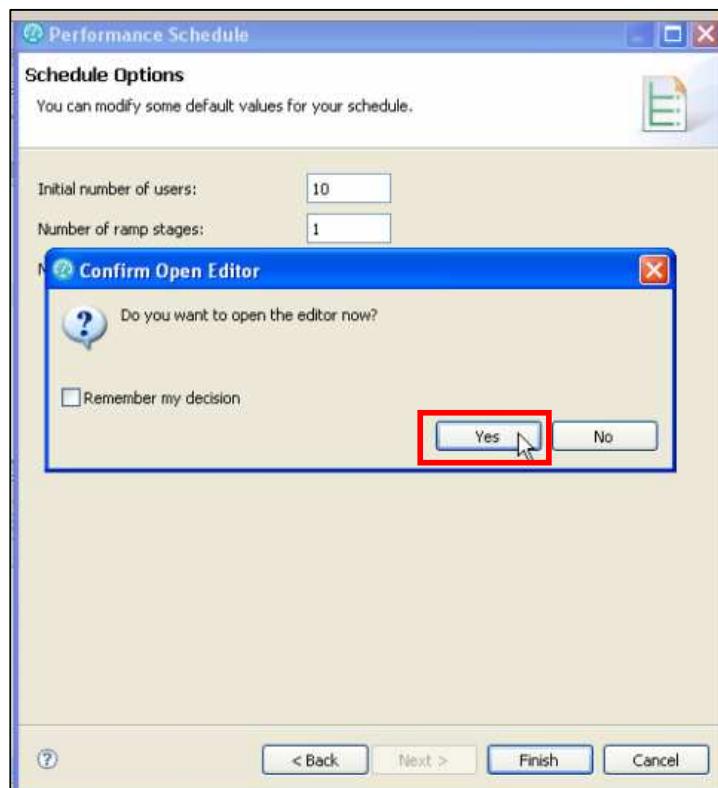
2. Ingrese el nombre de la programación y luego seleccione **Next**.



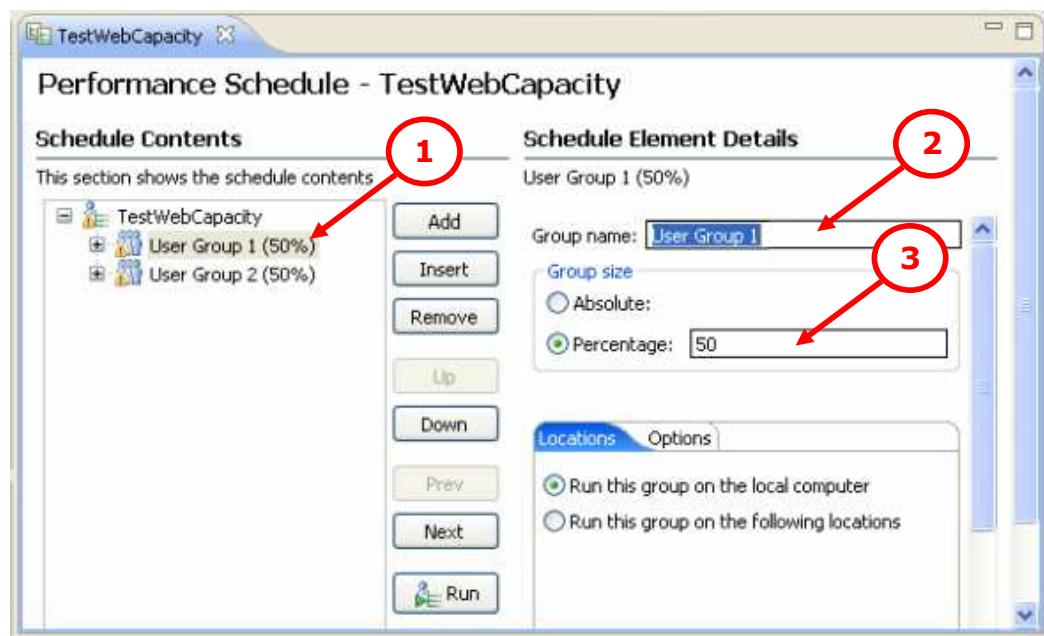
3. Ingrese los valores iniciales de la programación, seleccione el tipo de bucle para cada grupo de usuario y luego **Finish**.



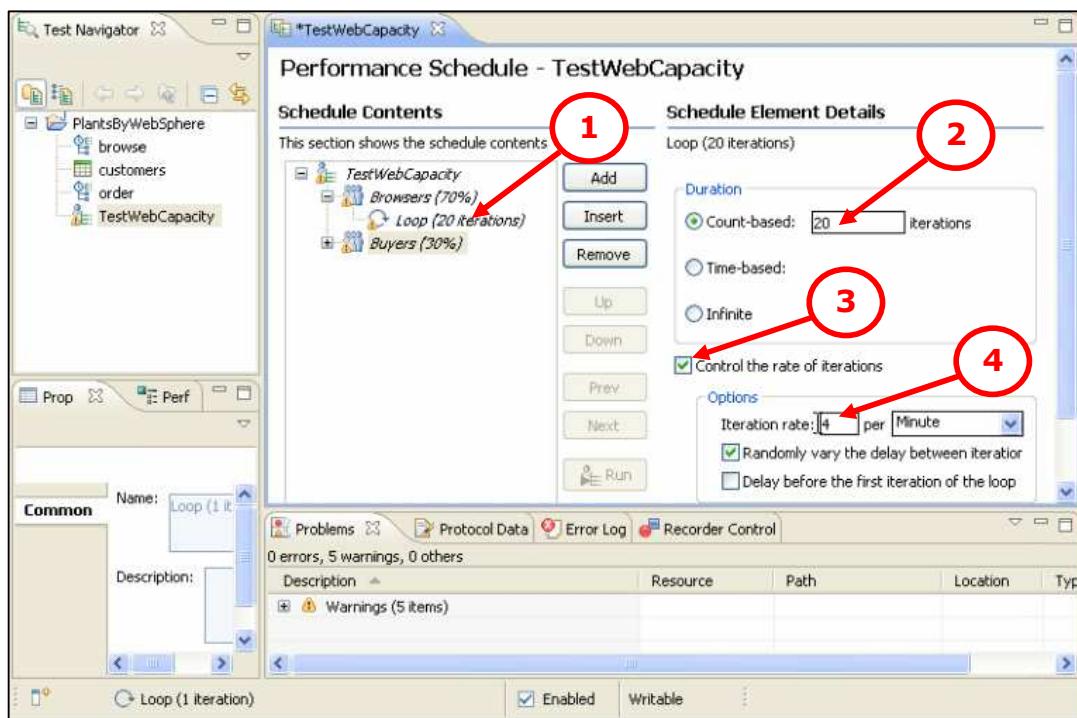
4. Ahora, confirme la visualización del editor de programación.



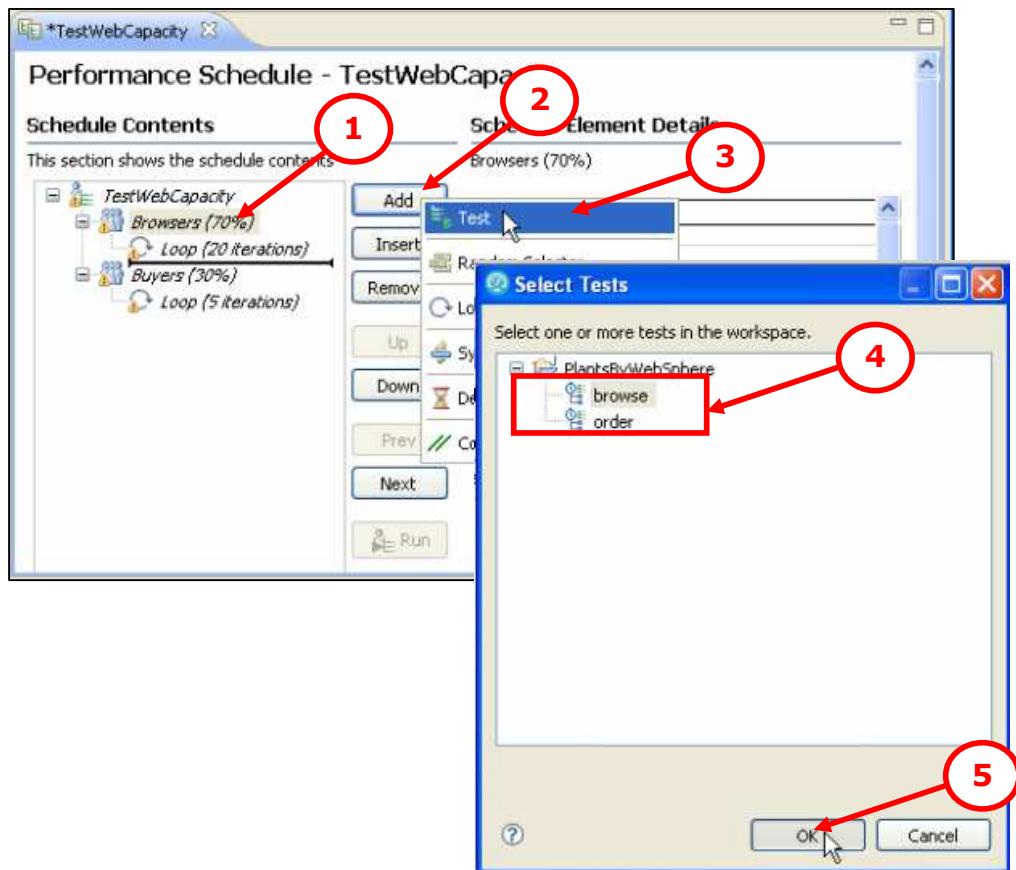
5. A continuación, modifique la información de cada grupo de usuario: nombre y porcentaje, según los usuarios que tendrá la aplicación a probar. Para ello, seleccione el grupo y realice los cambios desde la sección de detalles (panel izquierdo).



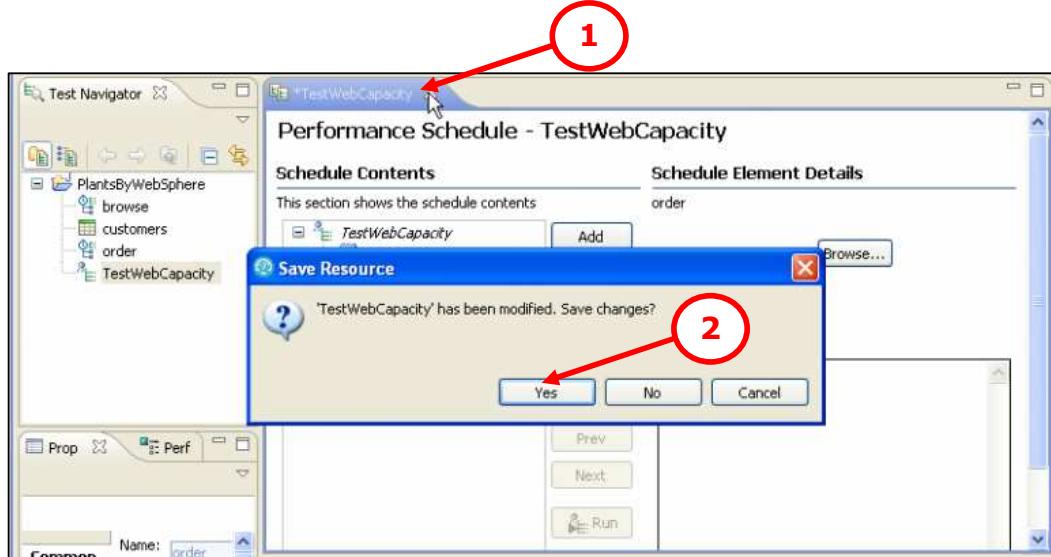
6. Ahora, agregue las características del bucle de cada grupo de usuario.



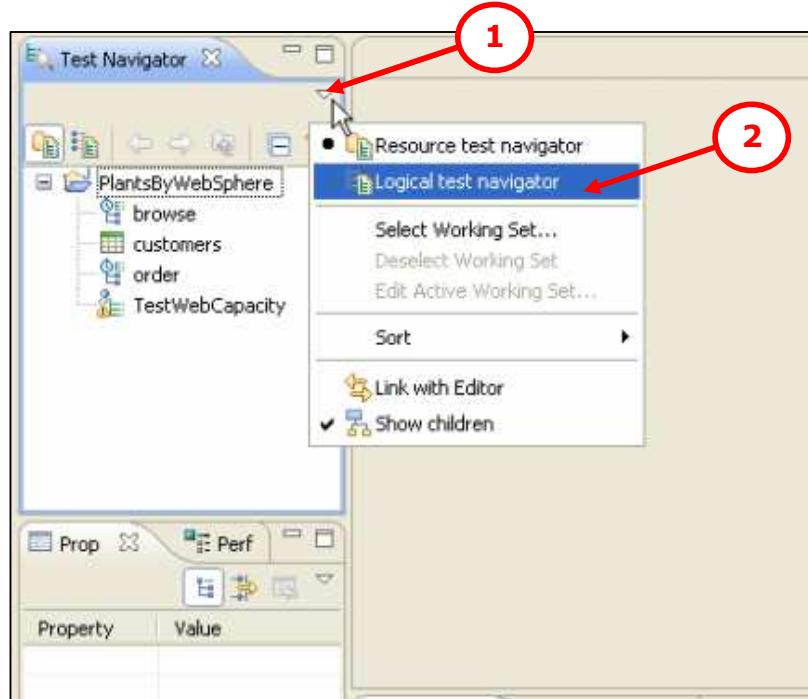
7. A continuación, agregue las pruebas que se requieran a cada grupo de usuario. Así:



8. Luego, cierre el editor de programación de pruebas y confirme para guardar los cambios.



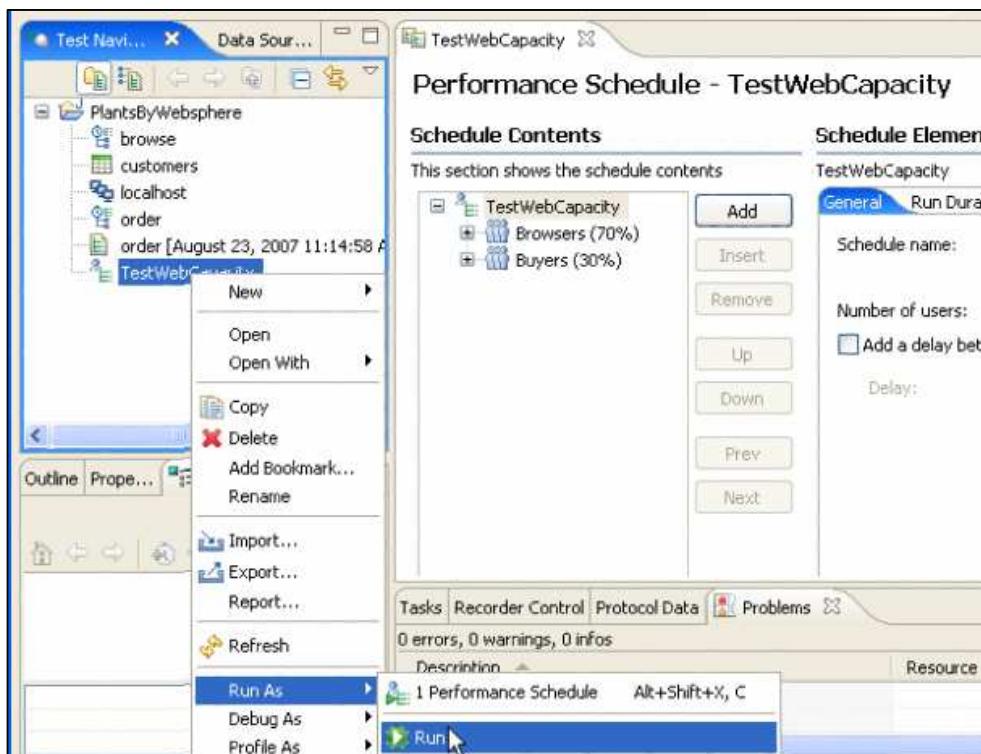
9. Por último, para agrupar los activos de prueba por tipo, seleccione el ícono del triángulo y seleccione **Logical test navigator**.



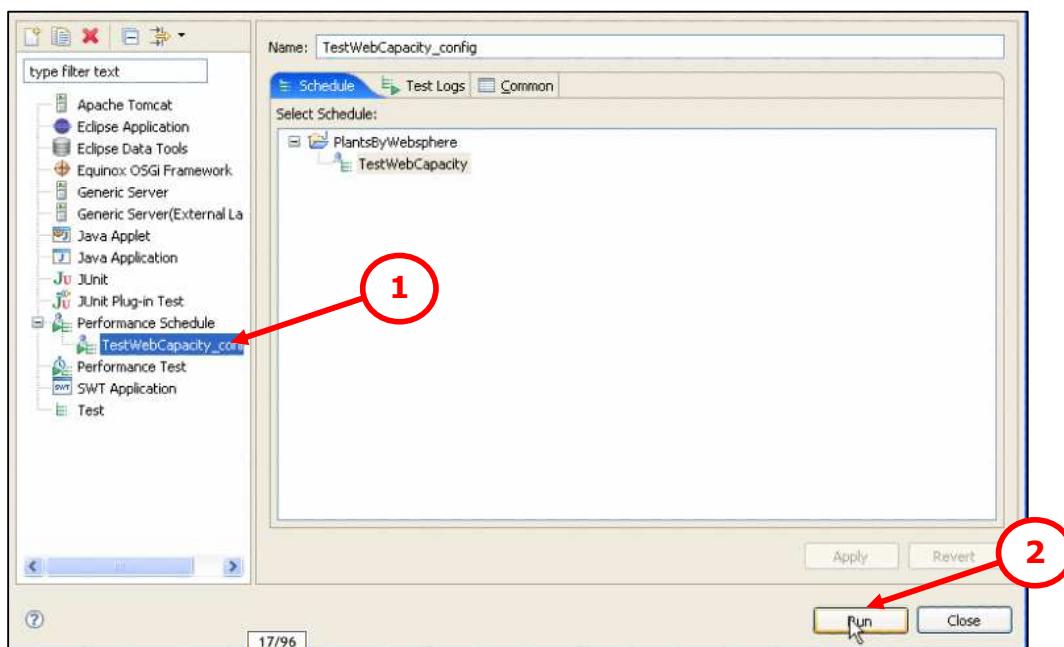
## 10. Ejecución de una Prueba

Para ejecutar la prueba con una programación previamente configurada realice los siguientes pasos:

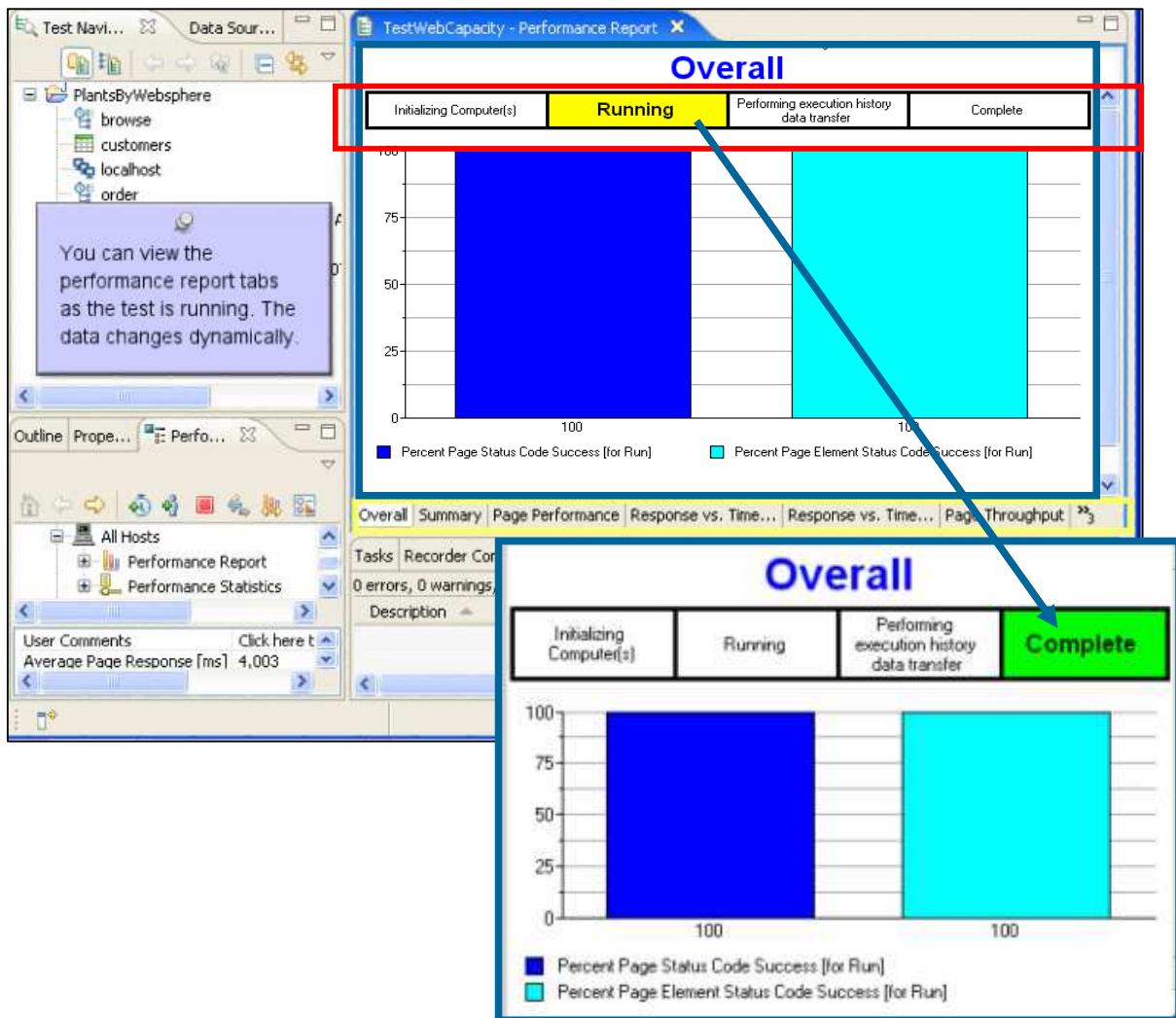
1. Seleccione ejecutar con la opción **Run**.



2. Seleccione una programación configurada localmente, de lo contrario se genera una programación por defecto.



3. Los informes de la programación, que se explicaron anteriormente, se visualizan e irán actualizándose en tiempo real **hasta que el indicador de progreso se complete**.



## Resumen

- ❑ Los objetivos de las pruebas de rendimiento son:
  - Determinar los tiempos de respuesta del sistema
  - Determinar el número máximo de usuarios de un sistema (componentes, transacción, o configuración)
  - Descubrir las configuraciones óptimas o mínima del sistema
- ❑ Rational Performance Tester es una solución de verificación de cargas y rendimiento para equipos que se ocupen de la capacidad de ampliación de sus aplicaciones basadas en web. Gracias a la combinación de funciones de análisis detallados y fáciles de utilizar, Rational Performance Tester simplifica la creación de pruebas, la generación de cargas y la recopilación de datos para garantizar que las aplicaciones se amplíen hasta miles de usuarios concurrentes.
- ❑ Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - ☞ <http://publib.boulder.ibm.com/infocenter/rpthelp/v8r1m0/index.jsp>  
En esta página encontrará información del entorno IBM RPT: guía de instalación, guía de aprendizaje, entre otros temas.
  - ☞ [http://agile.csc.ncsu.edu/SEMMaterials/tutorials/rpt/index.html#section8\\_0](http://agile.csc.ncsu.edu/SEMMaterials/tutorials/rpt/index.html#section8_0)  
En esta página, hallará un ejercicio sobre prueba de rendimiento con RPT.
  - ☞ <http://www.ibm.com/developerworks/ssa/rational/library/09/buildingscriptsinrationalperformancetester/index.html>  
En esta página, hallará un ejemplo de generación de scripts sólidos para pruebas de confiabilidad en RPT.