PPSS Planificación y Pruebas de Sistemas Software
Curso 2017-18

Sesión S08: Pruebas de aceptación (1)

- Pruebas de aceptación
- Propiedades emergentes
- Diseño de casos de prueba de aceptación
- Pruebas de propiedades emergentes funcionales con Katalon Recorder

María Isabel Alfonso Galipienso
Universidad de Alicante
eli@ua.es

Niveles de pruebas

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

VERIFICACIÓN
¿está implementado correctamente?

nivel de unidades nivel de integración nivel de sistema

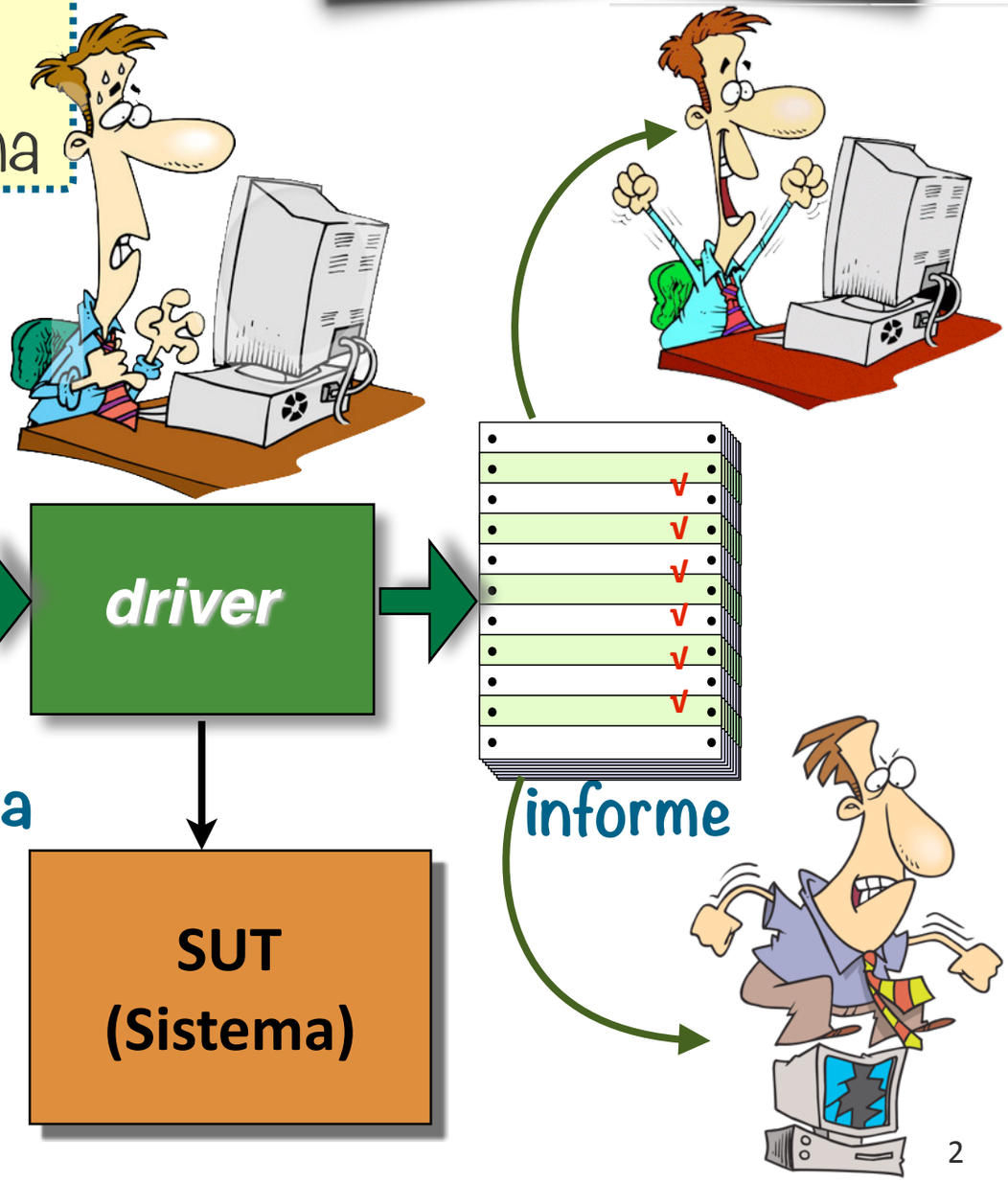
VALIDACIÓN
¿el producto es el correcto?

nivel de aceptación

Cuestión fundamental: ¿cuáles son los criterios bajo los cuales el sistema será aceptado por el usuario?

(ACCEPTANCE CRITERIA)

Objetivo: valorar en qué grado el software desarrollado satisface las expectativas del cliente



Acceptance testing

- Un producto está listo para ser entregado (*delivered*) al cliente después de que se hayan realizado las pruebas del sistema
  - A continuación, los clientes ejecutan los tests de **aceptación** basándose en sus expectativas sobre el producto. Finalmente, si los tests de aceptación son "aceptados" por el cliente, el sistema será puesto en PRODUCCIÓN.
- Las pruebas de aceptación son pruebas formales orientadas a determinar si el sistema satisface los criterios de aceptación (***acceptance criteria***)
  - Los criterios de aceptación de un sistema deben satisfacerse para ser aceptados por el cliente (éste generalmente se reserva el derecho de rechazar la entrega del producto si los tests de aceptación no "pasan")
- Hay dos categorías de pruebas de aceptación:
  - **User acceptance testing (UAT)**
    - Son **dirigidas por el cliente** para asegurar que el sistema satisface los criterios de aceptación contractuales
  - **Business acceptance testing**
    - Son **dirigidas por la organización** que desarrolla el producto para asegurar que el sistema eventualmente "pasará" las UAT. Son un ensayo de las UAT en el lugar de desarrollo

Acceptance criteria

- Los criterios de aceptación deben ser DEFINIDOS y ACORDADOS entre el proveedor (organización a cargo del desarrollo) y el cliente
  - Constituyen el "núcleo" de cualquier acuerdo contractual entre el proveedor y el cliente
- Una cuestión clave es: ¿Qué criterios debe satisfacer el sistema para ser aceptado por el cliente?
  - El principio básico para diseñar los criterios de aceptación es asegurar que la calidad del sistema es aceptable
  - Los criterios de aceptación deben ser medibles, y por lo tanto, cuantificables
- Algunos atributos de calidad que pueden formar parte de los criterios de aceptación son:
  - Corrección funcional y Completitud
    - ¿El sistema hace lo que se quiere que haga? ¿Todas las características especificadas están presentes?
  - Exactitud, integridad de datos, rendimiento, stress, fiabilidad y disponibilidad, mantenibilidad, robustez, confidencialidad, escalabilidad,...

Propiedades emergentes

- Cualquier atributo incluido en los criterios de aceptación es una **propiedad emergente**
- Las propiedades "emergentes" son aquellas que no pueden atribuirse a una parte específica del sistema, sino que "emergen" solamente cuando los componentes del sistema han sido integrados, ya que son el resultado de las complejas interacciones entre sus componentes. Por lo tanto, no pueden "calcularse" directamente a partir de las propiedades de sus componentes individuales
- Hay dos tipos de propiedades emergentes:
  - **Funcionales**: ponen de manifiesto el propósito del sistema después de integrar sus componentes
  - **No funcionales**: relacionadas con el comportamiento del sistema en su entorno habitual de producción (p.ej. fiabilidad, seguridad...)
- En esta sesión nos vamos a centrar en las pruebas de propiedades emergentes FUNCIONALES

Diseño de pruebas de aceptación

- Deberían ser responsabilidad de un grupo separado de pruebas que no esté implicado en el proceso de desarrollo. Se trata de determinar que el sistema es lo suficientemente "bueno" como para ser usado (entregado al cliente, o ser lanzado como producto): cumple los criterios de aceptación
- Normalmente se trata de un proceso *black-box* (functional testing) basado en la especificación del sistema. También reciben el nombre de "pruebas funcionales", para indicar que se centran en la "funcionalidad" y no en la implementación
- Ejemplos de métodos de diseño de pruebas emergentes funcionales:
  - Diseño de pruebas basado en **requerimientos**
    - son pruebas de validación (se trata de demostrar que el sistema ha implementado de forma adecuada los requerimientos y que está preparado para ser usado por el usuario). Cada requerimiento debe ser "testable"
  - Diseño de pruebas de **escenarios**
    - los escenarios describen la forma en la que el sistema debería usarse

Diseño de pruebas basado en requerimientos (I)

Capítulo 8.3.1 "Software Engineering" 9th. Ian Sommerville

- Un principio general de una buena especificación de un requerimiento es que debe escribirse de forma que se pueda diseñar una prueba a partir de él
  - [610-12-1990-IEEE Standard Glossary of Software Engineering Terminology]. A requirement is:
    1. A condition or capability needed by a user to solve a problem or achieve an objective
    2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
    3. A documented representation of a condition or capability as in 1 or 2.

- Ejemplo de requerimiento “testable”

> *If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.*
>
> *If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.*

- Ejemplo de casos de prueba que podemos derivar del requerimiento anterior:

1. Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system

Diseño de pruebas basado en requerimientos (II)

- Ejemplo de casos de prueba que podemos derivar del requerimiento anterior (continuación):

2. Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.

3. Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.

4. Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.

5. Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

- Fijaos que para un único requerimiento necesitaremos varios tests para asegurar que "cubrimos" todo el requerimiento

Diseño de pruebas basado en escenarios (I)

Capítulo 8.3.2 "Software Engineering" 9th. Ian Sommerville

- Un escenario es una descripción de un ejemplo de interacción del usuario/s con el sistema. Un escenario debería incluir:
  - Una descripción de las asunciones iniciales, una descripción del flujo normal de eventos, y de situaciones excepcionales; y una descripción del estado final del sistema cuando el escenario termine
- Ejemplo de escenario:

Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

Diseño de pruebas basado en escenarios (II)

- El escenario anterior describe las siguientes características (requisitos o requerimientos) de nuestro sistema:
  - Authentication by logging on to the system.
  - Downloading and uploading of specified patient records to a laptop
  - Home visit scheduling
  - Encryption and decryption of patient records on a mobile device
  - Record retrieval and modification
  - Links with the drugs database that maintains side-effect information
  - The system for call prompting
- Las pruebas basadas en escenarios normalmente prueban varios requerimientos en un mismo escenario. Por ello, además de probar los requerimientos individuales, también estamos probando la combinación de varios de ellos
  - El diseño de pruebas resultante se obtiene agregando los casos de prueba que tengan en cuenta los requerimientos anteriores: el tester, cuando ejecuta este escenario, adopta el rol de Kate, y debe contemplar situaciones como introducir credenciales erróneas, o información incorrecta sobre el paciente

Pruebas funcionales: Katalon

Podéis consultar "Katalon Automatio Recorder Quickstart" en https://www.katalon.com/resources-center/blog/katalon-automation-recorder/

- Katalon Automation Recorder (KAR) es una herramienta para automatizar pruebas funcionales sobre **aplicaciones Web**
  - Permite escribir *scripts* de pruebas utilizando la aplicación web tal y como un usuario haría normalmente: a través del navegador. Actualmente puede instalarse como una extensión de los navegadores Chrome o Firefox (nosotros trabajaremos con la extensión de Firefox)
  - La herramienta puede "grabar" las "acciones" realizadas sobre la aplicación web y generar los scripts de pruebas (tests) de forma sencilla sin necesidad de conocimientos de programación. Los tests pueden editarse, y también pueden crearse de forma manual.
  - Los tests creados usan comandos "selenese", cuya sintaxis fue creada para usarse con Selenium IDE (una herramienta precursora de KAR, pero que ya no puede usarse con la versiones de Firefox actual)
    - Básicamente se trata de detectar la existencia de elementos UI (User Interface) a través de sus etiquetas HTML, y permitir interactuar con dichos elementos

Podéis consultar "Selenium Documentation" en http://seleniumhq.org/docs/index.html

Katalon Recorder

Botón de grabación

Genera comandos Selenese mientras de interacciona con el navegador ("graba" las acciones del usuario).

Ejecución de los tests

Control de la velocidad de ejecución de los tests

Editor de scripts (comandos Selenese)

Tests

Un conjunto de tests (scripts) se denomina SUITE

No todos los comandos pueden generarse de forma automática

Panel de Log

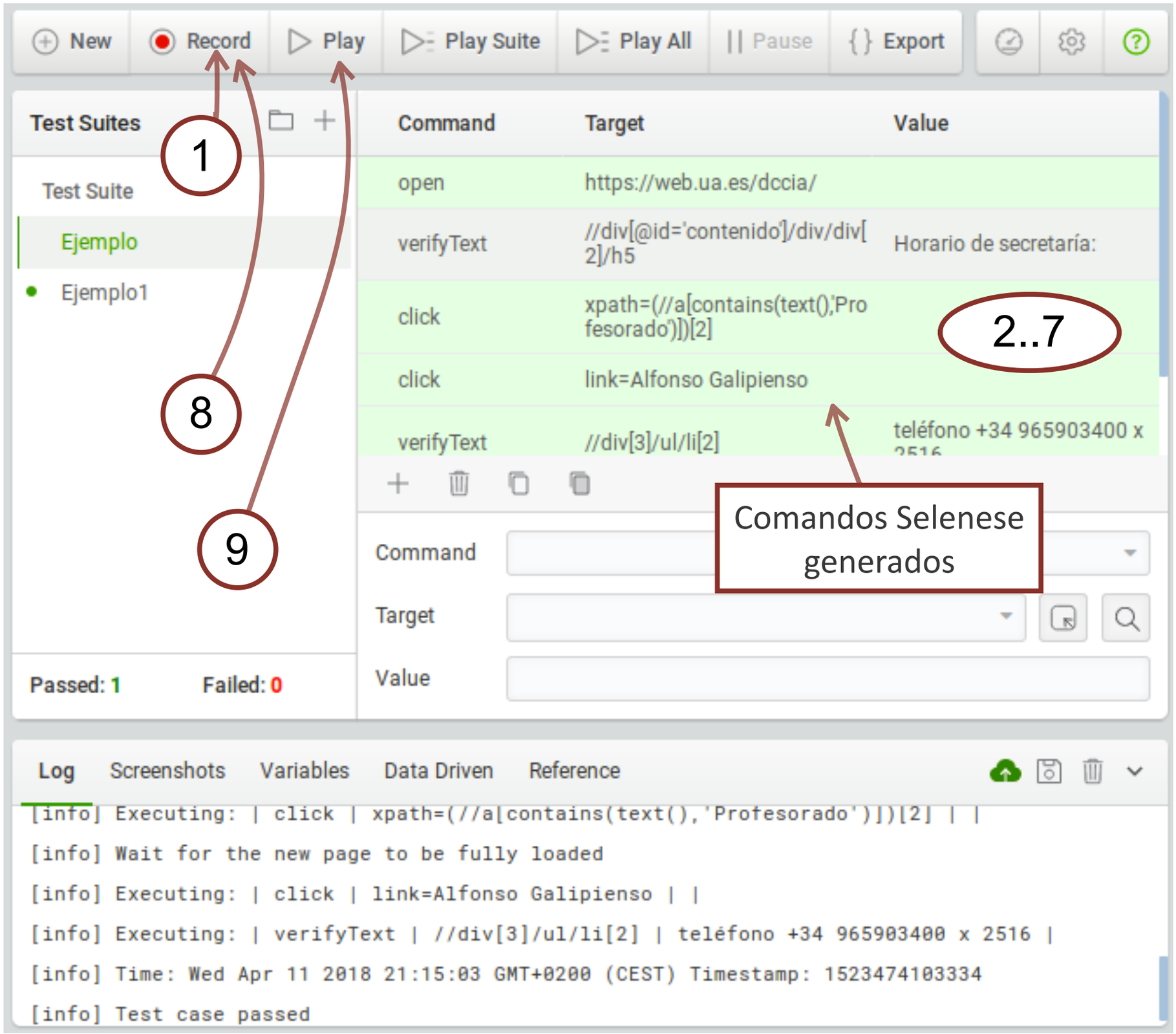
Muestra información sobre la ejecución del caso de prueba

Panel de referencia

Muestra la documentación sobre los comandos

Ejemplo de Test Script

1. Abrimos Katalon y pulsamos el botón grabar
2. Acceder desde Firefox a http://www.dccia.ua.es
3. Seleccionamos el texto "Horario" en la página, y con botón derecho seleccionamos el comando "verifyText ... "
4. Desplegamos el elemento "COMPONENTES"
5. Pinchamos sobre "PROFESORADO"
6. Pinchamos sobre "Alfonso Galipienso, María Isabel"
7. Seleccionamos el texto "2516" y con botón derecho "verifyText ..."
8. Detenemos la grabación
9. Ejecutamos el caso de prueba



Código Generado

- Comandos Selenese

Se generan automáticamente durante la grabación, pero también podemos generarlos de forma manual

Command	Target	Value
open	https://web.ua.es/dccia/	
verifyText	//div[@id='contenido']/div/div[2]/h5	Horario de secretaría:
click	//div[@id='menu']/ul/li[3]/div	
click	xpath=(//a[contains(text(),'Profesorado')])[2]	
click	link=Alfonso Galipienso	
verifyText	//div[3]/ul/li[2]	teléfono +34 965903400 x 2516

- Podemos "guardar" el código generado en formato html (opción "Export")

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html ...>
<title>Test Suite</title>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<tr>
    <td>open</td>
    <td>https://web.ua.es/dccia/</td>
    <td></td>
</tr>
<tr>
    <td>verifyText</td>
    <td>//div[@id='contenido']/div/div[2]/h5</td>
    <td>Horario de secretaría:</td>
</tr>
<tr>
    <td>click</td>
    <td>//div[@id='menu']/ul/li[3]/div</td>
    <td></td>
</tr>
```

```
<tr>
    <td>click</td>
    <td>xpath=(//a[contains(text(),'Profesorado')])[2]</td>
    <td></td>
</tr>
<tr>
    <td>click</td>
    <td>link=Alfonso Galipienso</td>
    <td></td>
</tr>
<tr>
    <td>verifyText</td>
    <td>//div[3]/ul/li[2]</td>
    <td>teléfono +34 965903400 x 2516</td>
</tr>

</tbody></table>
</body>
</html>
```

Comandos Selenese (I)

COMMAND + [TARGET] + [VALUE]

- Un **comando** Selenese puede tener uno o dos parámetros:
  - **target**: hace referencia a un elemento HTML de la página a la que se accede. (p.ej "link" indica un hipernlace)
  - **value**: contiene un texto, patrón, o variable a introducir en un campo de texto o para seleccionar una opción de una lista de opciones
- Una secuencia de comandos Selenese forman un "test script" (caso de prueba). Una secuencia de tests scripts forman una test suite

Hay 3 tipos de comandos:

actions

interactúan directamente con los elementos de la página.
P.ej. "click", "type"

Muchas actions pueden llevar el sufijo AndWait

accessors

permiten almacenar valores en variables
P.ej. "storeTitle"

assertions

verifican que se cumple una determinada condición

Hay 3 tipos:
- Assert: detienen la ejecución si no se cumple
- Verify: se registra el fallo y continúa la ejecución
- WaitFor: espera a que se cumpla una condición antes de fallar (por defecto tiene un timeout de 30 segundos)

Comandos Selenese (II)

open: abre una página usando una URL

click/clickAndWait: realiza una operación click y opcionalmente espera a que se cargue la nueva página

verifyTitle/assertTitle: verifica el valor del título de una página

verifyText: verifica que un texto esté presente en algún lugar de la página, requiere el uso de un locator (indicado en el campo target del comando)

verifyElementPresent: verifica que un elemento de la página, definido por su etiqueta html esté presente en la página

waitForPageToLoad: pausa la ejecución hasta que la nueva página esperada termine de cargarse

store, storeText, storeEval: almacena un valor constante, un texto, o el resultado de aplicar un script java, respectivamente en una variable

SINTAXIS:

open(url)

click(locator)

assertTitle(pattern)

verifyText(locator, pattern)

verifyElementPresent(locator)

waitForPageToLoad(timeout)

store(expression, variableName)

storeText(locator, variableName)

storeEval(script, variableName)

Ver http://www.seleniumhq.org/docs/02_selenium_ide.jsp#selenium-commands-selenese

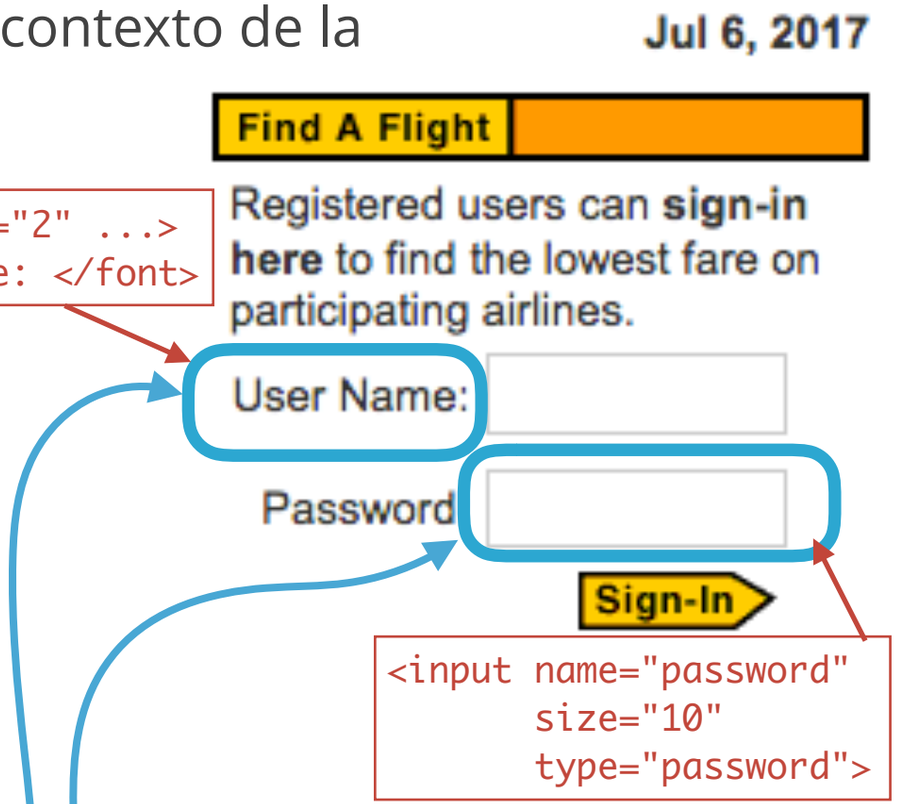
Ver https://docs.katalon.com/display/KD/Selenese+%28Selenium+IDE%29+Commands+Reference

Ver https://www.guru99.com/first-selenium-test-script.html

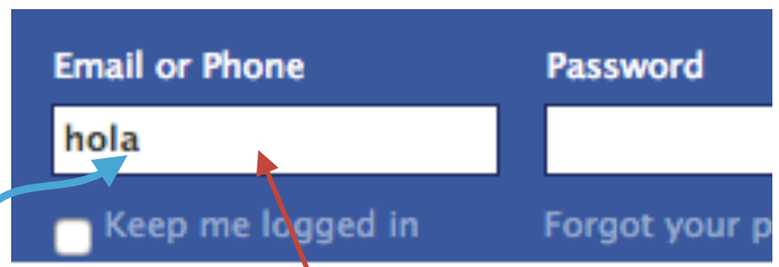
Locators

Ver http://www.seleniumhq.org/docs/02_selenium_ide.jsp#locating-elements

- Se utilizan en el campo **target** e identifican un elemento en el contexto de la aplicación web. La sintaxis es `locatorType=location`
- Podemos utilizar los siguientes tipos de "locators":
  - id: hace referencia al atributo **id** del elemento html
  - name: atributo **name** del elemento html. Adicionalmente podemos añadir un "filtro" consistente en añadir otro atributo adicional junto con su valor
  - xpath: es el lenguaje utilizado para localizar nodos en un documento HTML
    - Podemos utilizar rutas absolutas: xpath=/html/body/form[1]
    - o rutas relativas: xpath=//form[1]



Command	Target	Value
open	http://demo.guru99.com/test/newtours/	
verifyText	//tr[4]/td/table/tbody/tr[2]/td/font	User*
verifyElementPresent	name=password	
open	http://demo.guru99.com/test/facebook.html	
type	id=email	hola




```
<input class="inputtext"
       name="email"
       id="email"
       value=""
       tabindex="1"
       type="text">
```

Ver https://www.guru99.com/locators-in-selenium-ide.html

Ejemplos de locators (I)

- Suponemos que el código HTML de nuestra página web es:

```
1  <html>
2   <body>
3    <form id="loginForm">
4     <input name="username" type="text" />
5     <input name="password" type="password" />
6     <input name="continue" type="submit" value="Login" />
7     <input name="continue" type="button" value="Clear" />
8    </form>
9   </body>
10  <html>
```

- id=loginForm (línea 3)
- name=username (4)
- name=continue type=button (6)

Usamos un filtro para refinar la búsqueda

- xpath=/html/body/form[1] (3) es una ruta absoluta
- //form[1] (3) es una ruta relativa (encuentra el primer "form" en el HTML)

- link=Continue (4)
- link=Cancel (5)

```
1  <html>
2   <body>
3    <p>Are you sure you want to do this?</p>
4    <a href="continue.html">Continue</a>
5    <a href="cancel.html">Cancel</a>
6  </body>
7  <html>
```

Localización de elementos desde Firefox

- Podemos obtener información de los elementos de una página web mediante la utilidad "inspeccionar elemento" desde el menú contextual (desde dicho elemento)

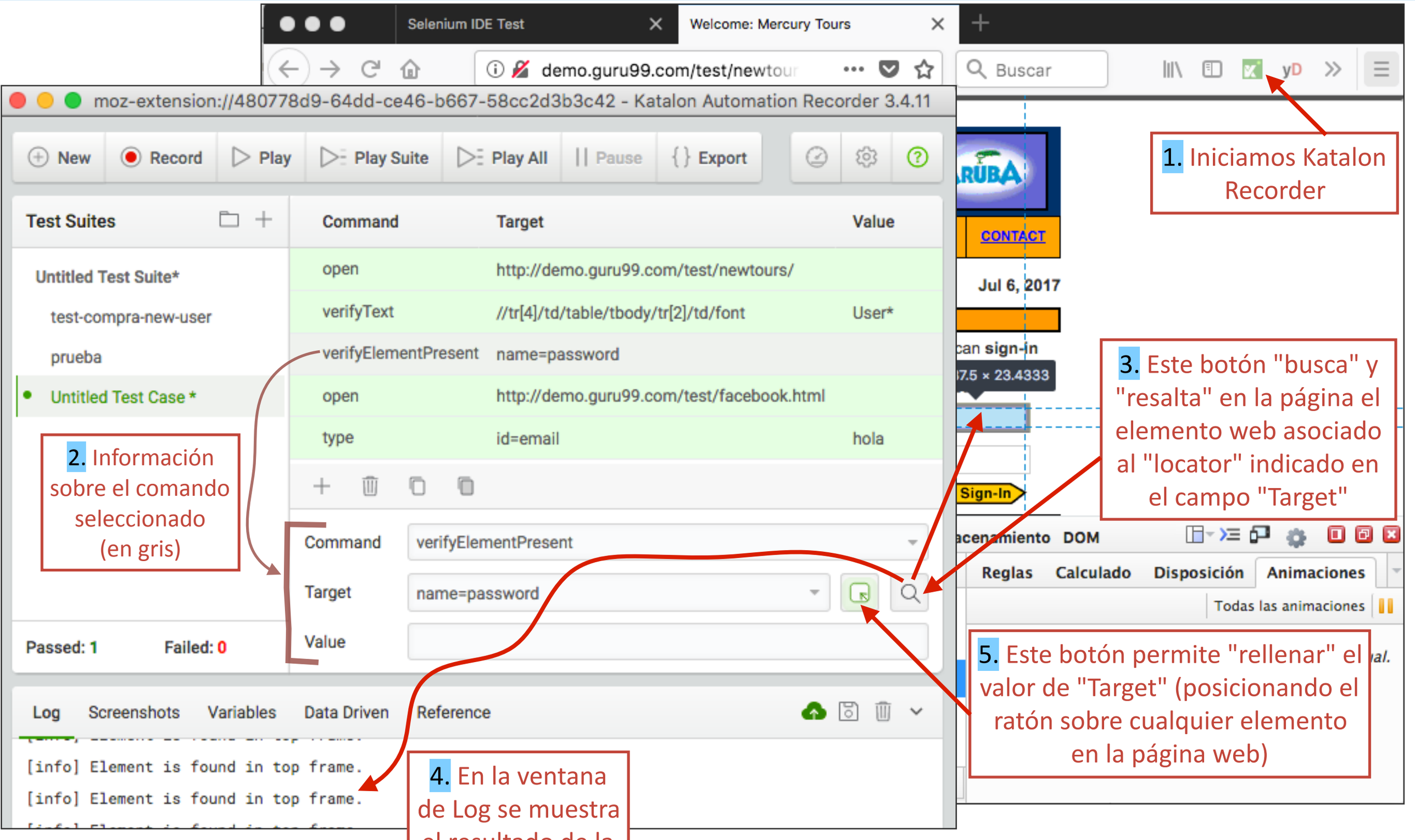
1. Situamos el ratón dentro del cuadro de texto y seleccionamos "Inspecionar elemento desde el menú contextual (botón derecho)

3. Si nos posicionamos sobre el resaltado en azul de la ventana de inspección (2), también se resaltará en elemento en la página

4. En cualquier momento podemos "buscar" cualquier otro elemento activando el botón de inspección

2. En la ventana del inspector se resalta el código del elemento

Localización de elementos desde Katalon



COMANDOS SELENESE: VARIABLES

Ver https://www.guru99.com/store-variables-handling-selenium-ide.html

- Algunos comandos selenese (comandos del grupo "accessors") pueden usar "variables"

store

value of the variable

Command	Target	Value
store	tutorial	myVariable

name of the variable

almacenamos en la variable con nombre "myVariable" el valor "tutorial"

la variable creada puede usarse en otros comandos

Command	Target	Value
open		
store	tutorial	myVariable
type	name=userName	${myVariable}

storeText


```
<div class="_6_ _74">
    <h1 class="mbs _3m8 _6n _6s _6v " style=
    36px">Sign Up</h1>
    <h2 class="mbl _3m7 _6o _6s _mf ">It's f
    will be.</h2>
```

almacenamos en la variable "tester" el valor "Sign Up" cuyo locator es css=h1

Command	Target	Value
open		
storeText	css=h1	textVar
echo	${textVar}	

Log	Reference	UI-Element	Rollup

[info] Executing: |open | | |
[info] Executing: |storeText | css=h1 | textVar |
[info] Executing: |echo | ${textVar} | |
[info] echo: Sign Up

storeElementPresent

name=userName

Note: The element with name=firstname does not exist in this page

almacenamos "true" o "false" en las variables var1 y var2 en función de que los elementos estén presentes en la página

Command	Target	Value
open		
storeElementPresent	name=userName	var1
storeElementPresent	name=firstname	var2
echo	${var1}	
echo	${var2}	

[info] Executing: |echo | ${var1} | |
[info] echo: true
[info] Executing: |echo | ${var2} | |
[info] echo: false

Sobre el proceso de "grabación" de pruebas

- Durante el proceso de "grabación" de las pruebas, Selenium nos "propone" comandos Selenese asociados a las acciones que estamos realizando sobre el navegador. Pero este proceso NO es INFALIBLE, es decir, en ocasiones tendremos que "retocar" el script generado para asegurar un buen funcionamiento
  - El ejemplo más claro es el comando **click**. Normalmente este comando provocará la carga de una nueva página, y tendremos que indicar a Selenium que espere hasta que se produzca la carga antes de continuar con la nueva instrucción (por lo que tendremos que utilizar el comando clickAndWait en vez de click). Algunas veces Selenium nos propondrá el comando clickAndWait, pero otras no. En cuyo caso tendremos que modificar manualmente el script generado. Ocasionalmente, el comando clickAndWait no esperará lo suficiente a que se cargue la página, en cuyo caso, tendremos que utilizar el comando waitForPageToLoad, (o waitForElementPresent)
- En ocasiones Selenium no podrá crear automáticamente el comando Selenese asociado. Un ejemplo claro es cuando queramos por ejemplo verificar que un texto NO aparece en la página

Y AHORA VAMOS AL LABORATORIO...

Vamos a implementar tests de aceptación (para validar propiedades emergentes funcionales) para una aplicación web con selenium IDE

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

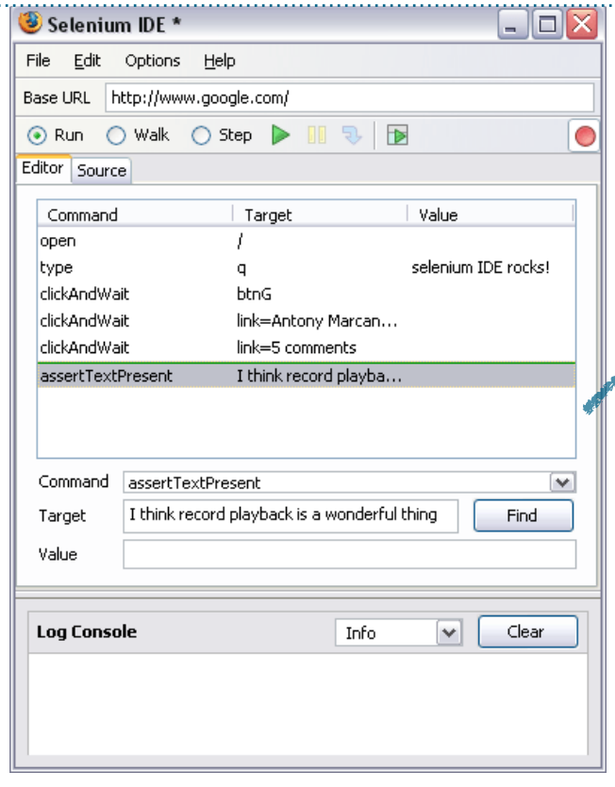
Tests aceptación

driver

Informe

Navegador FireFox

usaremos scripts Selenese utlizando Selenium IDE



Sevidor Web

Referencias bibliográficas

- Software testing and quality assurance. Kshirasagar Naik & Priyadarshi Tripathy. Wiley. 2008
  - Capítulo 14: Acceptance testing
- Software Engineering. 9th edition. Ian Sommerville. 2011
  - Capítulo 8.3: Release testing
- Selenium 2 testing tools. David Burns. 2012
  - Capítulos 1 y 2
- Tutorial Selenium (http://www.guru99.com/selenium-tutorial.html)
  - Apartados 1..6