



Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación



Proyecto Fin de Carrera

Testing y Calidad de Software. Automatización de Pruebas con Selenium WebDriver

Autor: Rafael Cubas Montenegro

Tutor: Antonio da Silva Fariña

INDICE GENERAL

AGRADECIMIENTOS	IX
RESUMEN	XI
ABSTRACT	XII
INTRODUCCIÓN	XIII
MOTIVACIÓN DEL PFC	XIII
OBJETIVO GENERAL DEL PFC	XV
CONTENIDO DEL PFC	XVII
PARTE 1ª. MARCO TECNOLÓGICO	1
CAPÍTULO 1	
CALIDAD DE SOFTWARE	1
1.1. CONCEPTO DE CALIDAD	1
1.2. NORMATIVA EXISTENTE. ESTÁNDARES ISO/IEC RELACIONADOS CON LA CALIDAD	4
1.2.1 MODELO ISO 25000:2005 PARA LA CALIDAD DEL PRODUCTO SOFTWARE	5
1.2.2. EL ESTÁNDAR ISO/IEC 25010. CARACTERÍSTICAS DE LA CALIDAD	8
1.3 CONCLUSIÓN. RESUMEN	12
CAPÍTULO 2	
PRUEBAS SOFTWARE.....	13
2.1 INTRODUCCIÓN	13
2.2 NIVELES DE LAS PRUEBAS SOFTWARE.....	14
2.3 MÉTODOS Y TÉCNICAS DE PRUEBA.....	15
2.3.1 Pruebas de caja blanca.....	15

2.3.2 Pruebas de caja negra	16
2.4 OTROS TIPOS IMPORTANTES DE PRUEBAS.....	19
2.4.1 Pruebas de regresión	19
2.4.2 Pruebas de humo	19
CAPÍTULO 3	
AUTOMATIZACIÓN DE PRUEBAS	21
3.1 INTRODUCCIÓN A LA AUTOMATIZACIÓN DE PRUEBAS	21
3.2 NIVELES DE AUTOMATIZACIÓN	23
3.3 BENEFICIOS DE LA AUTOMATIZACIÓN DE PRUEBAS	24
3.4 CLAVES PARA REALIZAR UNA AUTOMATIZACIÓN PRODUCTIVA...	26
3.5 CONCLUSIONES	27
CAPÍTULO 4	
INTRODUCCIÓN A SELENIUM.....	29
4.1 SELENIUM 1.0 (SELENIUM RC)	29
4.2 SELENIUM 2.0 (SELENIUM WEBDRIVER)	32
PARTE 2ª. ESTUDIO DE SELENIUM	35
CAPÍTULO 5	
OBJETIVOS Y ESTRUCTURA DEL ESTUDIO	35
5.1 OBJETIVOS DEL ESTUDIO	35
CAPÍTULO 6	
ENTORNO Y HERRAMIENTAS NECESARIAS	37
6.1 DESCARGA DE LA LIBRERÍA WEBDRIVER.....	37
6.2 CREACIÓN DE UN PROYECTO EN ECLIPSE	39
6.3 FIREFOX Y ALGUNOS COMPLEMENTOS DE UTILIDAD	43

CAPÍTULO 7

INTERFAZ WEBDRIVER. LOCALIZACIÓN DE LOS ELEMENTOS DE UNA WEB 47

7.1 INTERFAZ *WebDriver*. METODOS PARA OBTENER EL CONTENIDO DE UNA WEB 48

MÉTODO `get()` 48

MÉTODO `getCurrentUrl()` 48

MÉTODO `getTitle()` 49

MÉTODO `getPageSource()` 49

MÉTODO `close()` 49

7.2 INTERFAZ *WebDriver*. METODOS PARA LOCALIZAR ELEMENTOS EN UNA WEB 51

MÉTODO `findElement()` 51

MÉTODO `findElements()` 51

7.2.1 USO DE FIREBUG PARA LOCALIZAR ELEMENTOS EN EL CÓDIGO HTML 53

7.3. MECANISMOS DE LOCALIZACIÓN DE ELEMENTOS. LA CLASE *By* 55

MÉTODO `By.className()` 55

MÉTODO `By.name()` 55

MÉTODO `By.tagName()` 55

MÉTODO `By.id()` 56

MÉTODO `By.linkText()` 56

MÉTODO `By.partialLinkText()` 56

7.4 LA CLASE *WebElement*. ACCIONES SOBRE LOS ELEMENTOS DE UNA WEB 59

MÉTODO `WebElement.getAttribute()` 59

MÉTODO `WebElement.sendKeys()` 60

MÉTODO <code>WebElement.clear()</code>	60
MÉTODO <code>WebElement.submit()</code>	61
MÉTODO <code>WebElement.getCssValue()</code>	61
MÉTODO <code>WebElement.getLocation()</code>	62
MÉTODO <code>WebElement.getSize()</code>	62
MÉTODO <code>WebElement.isSelected()</code>	63
MÉTODO <code>WebElement.isDisplayed()</code>	63
MÉTODO <code>WebElement.getText()</code>	64
MÉTODO <code>WebElement.getTagName()</code>	64
MÉTODO <code>WebElement.isEnabled()</code>	65
MÉTODO <code>WebElement.click()</code>	65

CAPÍTULO 8

LA CLASE <code>ACTIONS</code> . INTERACCIONES AVANZADAS CON EL NAVEGADOR	67
--	----

8.1 INTERACCIÓN REALIZADA CON EL RATÓN

8.1.1 MÉTODO <code>moveByOffset()</code>	68
8.1.2 MÉTODO <code>moveToElement()</code>	69
8.1.3 MÉTODO <code>click()</code>	70
8.1.4 MÉTODO <code>doubleClick()</code>	71
8.1.5 MÉTODO <code>contextClick()</code>	72
8.1.6 MÉTODO <code>clickAndHold()</code>	73
8.1.7 MÉTODO <code>release()</code>	73
8.1.8 MÉTODO <code>dragAndDropBy()</code>	74
8.1.9 MÉTODO <code>dragAndDrop()</code>	74

8.2 INTERACCIÓN REALIZADA CON EL TECLADO	75
8.2.1 MÉTODO <code>keyDown()</code>	75
8.2.2 MÉTODO <code>keyUp()</code>	76
8.2.3 MÉTODO <code>sendKeys()</code>	77
8.3. LA CLASE <code>KEYS</code>	78
 CAPÍTULO 9	
WEBDRIVER. INTERFACES ANIDADAS	79
9.1 OPERACIONES DE NAVEGACIÓN. INTERFAZ <code>WEBDRIVER.NAVIGATION</code>	80
MÉTODO <code>back()</code>	80
MÉTODO <code>forward()</code>	80
MÉTODO <code>refresh()</code>	80
MÉTODO <code>to()</code>	80
9.2 GESTIÓN DE TIEMPOS DE ESPERA. INTERFAZ <code>WEBDRIVER.TIMEOUTS</code>	82
MÉTODO <code>implicitlyWait()</code>	82
MÉTODO <code>pageLoadTimeout()</code>	82
MÉTODO <code>setScriptTimeout()</code>	83
9.3 CARACTERÍSTICAS DE LA VENTANA. INTERFAZ <code>WEBDRIVER.WINDOW</code>	84
MÉTODO <code>fullscreen()</code>	84
MÉTODO <code>getPosition()</code>	84
MÉTODO <code>getSize()</code>	84
MÉTODO <code>maximize()</code>	85
MÉTODO <code>setPosition()</code>	85

MÉTODO setSize().....	85
9.4 GESTIÓN DE FRAMES. INTERFAZ WEBDRIVER.TARGETLOCATOR	86
MÉTODO defaultContent().....	86
MÉTODO parentFrame().....	86
MÉTODO frame()	87
9.5 RESUMEN.	89
 CAPÍTULO 10	
LA CLASE FIREFOXDRIVER.	91
10.1 CAPTURAS DE PANTALLA. INTERFAZ TAKESCREENSHOT	92
MÉTODO getScreenshotAs().....	92
10.2 LA CLASE FIREFOXPROFILE.....	94
10.1.1 CREACIÓN Y UTILIZACIÓN DE DIFERENTES PERFILES.	95
10.1.2 GUARDADO Y RECUPERACIÓN DE PERFILES.....	97
MÉTODO toJson().....	97
MÉTODO fromJson().....	98
10.1.3 CONFIGURACIÓN DE LAS PREFERENCIAS DEL NAVEGADOR.....	99
MÉTODO setPreference().....	100
10.3 LA CLASE FIREFOXBINARY	102

PARTE 3ª AUTOMATIZACIÓN DE PRUEBAS FUNCIONALES CON SELENIUM WEBDRIVER..... 103

CAPÍTULO 11

DESCRIPCIÓN DEL CASO PRÁCTICO..... 103

11.1 APLICACIÓN WEB UTILIZADA PARA LOS TEST AUTOMÁTICOS.. 103

11.2 SUPUESTO PRÁCTICO..... 105

11.3 OBJETIVOS PRINCIPALES. 106

11.4 SOFTWARE Y APLICACIONES UTILIZADAS. 106

CAPÍTULO 12

MODELO DE AUTOMATIZACIÓN..... 107

12.1 LA CLASE *TESCASE* 108

12.2 LA CLASE *FIREFOXEXECUTION* 109

12.3 LA CLASE *ENVIRONMENTTEST*..... 111

12.4 LA CLASE *TESTLOG*..... 113

CAPÍTULO 13

DEFINICIÓN DE LOS CASOS DE PRUEBA 115

13.1 TESTS PARA CUBRIR LA FUNCIONALIDAD “*CREATE_USERS/ROLES*”..... 115

13.2 TESTS PARA CUBRIR LA FUNCIONALIDAD “*TESTCASE/TESTSUIT_MANAGEMENT*”..... 119

CAPÍTULO 14

DEFINICIÓN DE LOS ENTORNOS DE EJECUCIÓN 123

CAPÍTULO 15

PROYECTO EN ECLIPSE	125
---------------------------	-----

<i>15.1 ESTRUCTURA DE LOS PAQUETES JAVA</i>	<i>125</i>
---	------------

<i>15.2 DATOS DE ENTRADA NECESARIOS PARA LOS TESTCASES.</i>	<i>127</i>
--	------------

<i>15.3 INFORMACIÓN GENERADA POR LOS TESTCASES. DATOS DE SALIDA.</i>	<i>128</i>
---	------------

CAPÍTULO 16

CONCLUSIONES	131
--------------------	-----

<i>16.1 ANALISIS DEL TRABAJO REALIZADO.</i>	<i>131</i>
--	------------

<i>16.2 ANALISIS DE SELENIUM WEBDRIVER COMO HERRAMIENTA DE AUTOMATIZACIÓN.</i>	<i>132</i>
---	------------

<i>16.3 LÍNEAS DE TRABAJO FUTURAS</i>	<i>134</i>
---	------------

AGRADECIMIENTOS

Quisiera mostrar mi más sincero agradecimiento a mi tutor, Antonio da Silva, por la paciencia y dedicación mostrada durante el desarrollo del proyecto, ya que teniendo que compaginar trabajo y estudios, sacar adelante el PFC no ha resultado una tarea fácil, ni para él, ni para mí.

También me gustaría mencionar a mis padres, ya que gracias a sus esfuerzos he podido cursar estudios superiores. Todo lo que soy y lo que tengo, se lo debo a ellos.

RESUMEN

El presente proyecto fin de carrera describe los conceptos básicos relacionados con la automatización de la fase de pruebas en un proyecto software utilizando la herramienta Selenium WebDriver. Esta herramienta de automatización permite programar pruebas en un lenguaje de alto nivel (java en nuestro caso) haciendo uso del API que proporciona.

Gracias a las diferentes funcionalidades que ofrece dicho API, se puede simular la interacción que realizaría un usuario sobre una determinada aplicación web y de esta manera poder recoger así datos sobre el funcionamiento de la misma. Para estructurar toda la información recogida en el PFC, se han desarrollado tres partes, cada una de ellas con un propósito diferente.

La primera parte consiste en describir el marco tecnológico del proyecto, aquí se ha realizado una introducción a la calidad de software mencionando algunas de las normas ISO que hacen referencia a esta disciplina y presentando también las técnicas y tipos de pruebas más frecuentes. Además de lo anterior, esta primera parte también contiene una introducción a la automatización de pruebas y a la herramienta Selenium, describiendo su evolución y presentando algunos detalles sobre su funcionamiento.

Por otro lado, la segunda parte del PFC se ha centrado en el estudio de las principales clases y métodos que proporciona el API de Selenium WebDriver. Se han incluido ejemplos y descripciones con el objetivo de crear una pequeña guía para que cualquier lector, que esté interesado, pueda empezar a trabajar con ésta herramienta de una manera sencilla.

Finalmente, la tercera parte contiene la puesta en práctica de lo aprendido en las dos partes anteriores. En esta fase del proyecto se ha planteado un supuesto práctico tomando como base una aplicación web real, TestLink, y se han diseñado varios casos de prueba automáticos para poder realizar una evaluación de Selenium WebDriver. El resultado ha sido bastante satisfactorio ya que, con conocer lo más básico sobre WebDriver, hemos sido capaces de automatizar las acciones que realizaría un tester de forma manual y obtener información sobre el comportamiento mostrado por la aplicación durante la ejecución de los tests automáticos.

Para concluir el PFC se ha incluido también un capítulo sobre las posibles líneas futuras de investigación, debido a que el trabajo aquí expuesto únicamente representa una introducción al mundo de la automatización de aplicaciones utilizando Selenium.

ABSTRACT

The current project describes the basic concepts related to the automation of the testing phase in a software project using the Selenium WebDriver tool. This tool allows programming tests in a high level language, java in our case, making use of the API that it provides.

Thanks to the different functionalities offered by this API, it is possible to simulate the interaction that a user would perform on a particular web application, and in this way, to collect data about its operation. In order to structure all the information gathered in the project, three distinct parts have been developed:

The first part consists of describing the technological framework of the project. An introduction to software quality has been made mentioning some of the ISO standards that refer to this and also presenting the most frequent techniques and types of tests. In addition to the above, this first part also contains an introduction to test automation and the Selenium tool, describing its evolution and presenting some details about its operation.

The second part of the PFC has been focused on the study of the main classes and methods provided by the Selenium WebDriver API. Examples and descriptions have been included in order to create a concise guide so that any interested user can easily start working with this.

Finally, the third part consists off an implementation of the knowledge acquired in the previous sections. In this phase of the project, a practice example has been raised taking a real application (TestLink) as a base of the automation tests in order to evaluate Selenium. The obtained results have been satisfactory, that with a basic knowledge about Selenium API, we have been able to automate the actions performed by a tester and obtain information about the behavior of the application during the execution of automate tests.

Before completing the project, we have also included a chapter in the third part about future lines of research. The current document just represents an introduction to the automation tests with Selenium.

INTRODUCCIÓN

MOTIVACIÓN DEL PFC

Producir un software con un nivel de calidad insuficiente puede ser un pésimo negocio. Según publica Standish Group en su informe de 2015 (Chaos Report), el 52% de los proyectos de desarrollo software no alcanza los objetivos fijados y el 19% de ellos se consideran un fracaso. Únicamente el 29% alcanzan el éxito. Ya no es suficiente con que una determinada aplicación o programa sea funcionalmente completo, acorde a su propósito, sino que debe también satisfacer otros aspectos en términos de seguridad, rendimiento, accesibilidad, mantenibilidad, etc. Por tanto, el objetivo de cualquier organización que se dedique a la industria del software debe ser producir software de calidad, es decir, crear software correcto de forma eficiente.

Según lo expuesto en el párrafo anterior la calidad debe ser una exigencia obligada y cada vez son más las empresas que consideran, dentro del proceso de desarrollo, actividades de aseguramiento de la calidad. Estas actividades comprenden una gran variedad de tareas, el establecimiento de métodos técnicos sólidos, métricas, análisis e informes, pruebas de software bien planificadas, revisiones, etc. Los responsables de todas estas actividades tienen como objetivo identificar, documentar y comunicar las posibles desviaciones observadas durante el proceso de desarrollo con el fin de tomar las decisiones adecuadas. Aportan claridad sobre el estado y sobre la madurez del producto que se está desarrollando.

Una de las partes importantes que componen las tareas propias de un sistema de aseguramiento de la calidad son las pruebas software. Existen pruebas unitarias, de integración, funcionales, de rendimiento, etc. Algunas de ellas, por su naturaleza, son susceptibles de ser automatizadas, esto significa que la ejecución la realiza una máquina o computadora en lugar del propio tester.

Actualmente la automatización de pruebas es un paso fundamental para mejorar la calidad del desarrollo. Cada vez las aplicaciones se vuelven más complejas, los requisitos van variando, se van añadiendo o modificando funcionalidades, etc., con lo que aparecen gran cantidad de versiones y la confiabilidad que produce la calidad final del software desarrollado disminuye. Es en estos casos donde la automatización de pruebas cobra importancia y representa una ventaja clara a la hora de abordar el proceso de ejecución de los tests.

Cómo se verá a lo largo de esta primera parte del PFC, la automatización puede ser aplicada en diferentes niveles de pruebas, desde pruebas unitarias, hasta pruebas de aceptación. Éste documento tratará en profundidad la automatización de pruebas funcionales (nivel de aceptación), donde se simulará la interacción del usuario final para verificar el correcto funcionamiento de una aplicación con interfaz web.

Actualmente existen varias herramientas que permiten generar y ejecutar scripts, pequeños programas que permiten interactuar con una aplicación dada, simulando las acciones que podría realizar el usuario real. Estas herramientas ayudan a aumentar la productividad en cuanto al número de pruebas realizadas, al alcance de cada una de ellas (se pueden probar muchas más combinaciones) y a la reproducibilidad de los test (nos aseguramos de que las pruebas se realizan siempre del mismo modo). Estas pruebas automáticas no sustituyen al tester tradicional, persona física que realiza manualmente las pruebas, sino que ayudan a realizar tareas tediosas y repetitivas, siendo necesario un perfil más técnico, ya que exige poseer conocimientos profundos en diferentes lenguajes de programación, bases de datos, arquitectura de sistemas web, etc.

Dentro del testeo de aplicaciones con interfaz web, una de las herramientas de automatización de pruebas funcionales que más importancia está cobrando es Selenium. Selenium es un framework opensource que tiene una importante comunidad detrás, y que permite realizar la navegación a través de una aplicación web tal y como lo haría el usuario final. La herramienta procesa un script que describe los pasos que debe ir realizando durante la ejecución del test: abrir el navegador, ir a una URL concreta, analizar texto mostrado en pantalla, acceder a un enlace determinado, introducir datos, etc.

Selenium ha ido evolucionando hasta ofrecer una API completa para diferentes lenguajes de programación. Con las clases y métodos que proporciona se crean los scripts que representan cada uno de los test automatizados. Actualmente la herramienta soporta los principales navegadores web, Firefox, Internet Explorer, Safari, etc.

OBJETIVO GENERAL DEL PFC

El objetivo del presente proyecto es adentrarse en el área de las pruebas funcionales automatizadas, realizando una pequeña introducción al mundo de la calidad y el testeo de software. Para ello, se realizará un estudio de las herramientas ofrecidas por Selenium y se llevará a cabo una parte práctica para evaluar dichas herramientas.

En general se pretenderá:

1. Presentar los conceptos básicos relacionados con la calidad de software, sus técnicas, la importancia que tiene dentro del proceso de desarrollo y los beneficios que ofrece.
2. Realizar una introducción a la automatización de pruebas.
3. Realizar un estudio de la herramienta Selenium con el fin de adquirir los conocimientos necesarios para simular la creación de un plan de pruebas funcionales automáticas sobre una aplicación real (TestLink).
4. Crear un documento que sirva como manual de referencia para todo aquel que pretenda empezar a utilizar Selenium para la automatización de pruebas funcionales sobre aplicaciones con interfaz web.
5. Crear los scripts necesarios para poder evaluar el potencial de la herramienta de automatización, analizando los beneficios/inconvenientes que pueden aportar al proceso de ejecución de pruebas este tipo de aplicaciones.

CONTENIDO DEL PFC

El proyecto se estructura en 3 partes, cada una de ellas compuesta por una serie de capítulos, tal y como se describe a continuación.

PARTE 1ª. MARCO TECNOLÓGICO

En esta primera parte se realiza una introducción a las áreas fundamentales sobre las que se basa el proyecto, con el objetivo de poner en contexto al lector y definir el ámbito del PFC.

Esta parte, por tanto, comprende los siguientes cuatro capítulos:

- Capítulo 1. Calidad de software.

En este capítulo se presenta el concepto de calidad de software y se hace referencia a la normativa existente relacionada con los estándares ISO/IEC.

- Capítulo 2. Pruebas software.

En este apartado se exponen los diferentes niveles de pruebas, las principales técnicas y métodos para su creación y algunos tipos de tests relacionados con el ámbito del proyecto.

- Capítulo 3. Automatización de pruebas.

En el capítulo 3 se desarrollan los principales conceptos relacionados con la automatización, describiendo su necesidad y los beneficios que puede aportar su implantación. También se detallan los diferentes niveles a los que se puede aplicar y las claves para poder realizar tests automáticos de un modo productivo.

- Capítulo 4. Introducción a Selenium.

En este capítulo se realiza una introducción a Selenium, describiendo su historia y evolución. También se detalla en qué consiste y para qué puede ser útil dentro del proceso de automatización de pruebas.

PARTE 2ª. ESTUDIO DE SELENIUM WEBDRIVER

La segunda parte del proyecto se centra en el estudio de los diferentes mecanismos que ofrece el API de Selenium WebDriver. A través de este API se puede ir simulando la interacción que realiza el usuario sobre una aplicación web utilizando un navegador.

- Capítulo 5. Objetivos y estructura del estudio.

En este capítulo se define qué se quiere conseguir con el estudio del API de Selenium WebDriver y también se especifica los pasos a seguir para ir entendiendo esta herramienta.

- Capítulo 6. Entorno y herramientas necesarias.

El capítulo 6 recoge una pequeña explicación sobre las herramientas y procesos que debemos tomar para poder empezar a trabajar con Selenium WebDriver.

- Capítulo 7. Interfaz WebDriver. Localización de los elementos de una web.

Este capítulo se realiza una introducción a la interfaz WebDriver presentando los métodos que permiten obtener información sobre una determinada dirección web, localizar los diferentes elementos que componen la página y actuar sobre ellos.

- Capítulo 8. La clase Actions. Interacciones avanzadas con el navegador.

En este apartado se presentan los diferentes métodos que ofrece la clase Actions para poder simular el uso del ratón y el teclado.

- Capítulo 9. WebDriver. Interfaces anidadas.

En el capítulo 9 se estudian los principales métodos de las interfaces anidadas Navigation, Timeouts, Windows y TargetLocator ya que ofrecen funcionalidades importantes a la hora de trabajar con el navegador, maximizar o minimizar la pantalla de navegación, gestionar los timeouts de espera para la carga de una web, ir a la página anterior, etc.

- Capítulo 10. La clase FirefoxDriver.

El capítulo 10 se centra en estudiar las características que ofrece la clase FirefoxDriver relacionadas con la gestión de las preferencias del navegador, los perfiles de navegación, la toma de capturas de pantalla, etc.

PARTE 3ª. AUTOMATIZACIÓN DE PRUEBAS FUNCIONALES CON SELENIUM WEBDRIVER.

En la tercera parte del proyecto se pondrá en práctica lo aprendido durante la fase de estudio. Para ello, se creará una estructura que soporte la ejecución de test automáticos y, tomando como objeto de test la aplicación web TestLink, se desarrollaran scripts en java para la realización de pruebas funcionales utilizando Selenium WebDriver.

- Capítulo 11. Descripción del caso práctico.

En el capítulo 11 se define el caso práctico sobre el cual se basará la parte práctica del PFC.

- Capítulo 12. Modelo de Automatización.

En este capítulo se presenta el modelo diseñado para poder dar soporte a la ejecución de los test automáticos. Se describe cómo se representará cada test automático y el mecanismo para poder ejecutarlo.

- Capítulo 13. Definición de los casos de prueba.

El capítulo 13 define los casos de prueba que se implementarán para ejecutarlos de forma automática. Se proporciona la descripción de cada uno de ellos, los principales pasos de ejecución y los datos de entrada y salida.

- Capítulo 14. Definición de los entornos de ejecución.

En el capítulo 14 se definen las condiciones sobre las que se ejecutarán los casos de prueba, es decir, su entorno de ejecución.

- Capítulo 15. Proyecto en eclipse.

Todo el desarrollo necesario para la automatización de los test se lleva a cabo utilizando el entorno que nos ofrece Eclipse. En este capítulo se describe cómo está organizado el proyecto en esta herramienta.

- Capítulo 16. Conclusiones.

En este último capítulo se exponen las conclusiones describiendo la experiencia obtenida con Selenium WebDriver durante el proceso de automatización de pruebas funcionales, y también, se hace referencia a las posibles líneas de investigación futuras.

PARTE 1ª. MARCO TECNOLÓGICO

CAPÍTULO 1

CALIDAD DE SOFTWARE

1.1. CONCEPTO DE CALIDAD

¿Qué es la calidad de software?

Empecemos por definir el concepto general de “Calidad”. El estándar ISO 9000 la define de la siguiente manera: “Calidad: grado en el que un conjunto de características inherentes cumple con los requisitos”, entendiéndose por requisito “necesidad o expectativa establecida, generalmente implícita u obligatoria”.

En el ámbito de la calidad de software, podemos hacer referencia a las siguientes definiciones formales:

- “Calidad de Software: Grado con el cual, el cliente o usuario percibe que el software satisface sus expectativas” [IEEE Standard 729-1983. IEEE Standard Glossary of Software Engineering Terminology].
- “La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario” [IEEE STD 610-1990. IEEE Standard Glossary of Software Engineering Terminology].
- “La calidad de software es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados, y de las características implícitas que se espera de todo software desarrollado profesionalmente” [Pressman, R. 2005. Ingeniería del Software. 6ª Ed. McGraw-Hill. Parte III, cap. 16-19].

Analizando las definiciones anteriores podemos deducir que:

1. La desviación del cumplimiento con los requisitos o expectativas fijadas por el cliente se traduce en una falta de calidad del software desarrollado.
2. La ausencia de estándares durante el proceso de desarrollo puede desembocar en un producto con un nivel de calidad insuficiente.
3. Existen ciertos requisitos implícitos exigibles a todo producto software, que también determinan el nivel de calidad alcanzado.

¿Qué objetivos tiene?

La calidad de software persigue conseguir:

1. Desde el punto de vista del usuario final: cumplir con los requisitos acordados, satisfaciendo sus necesidades y expectativas.
2. Desde el punto de vista del propio software: Desarrollar un producto que alcance un nivel de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad, suficiente.
3. Desde el punto de vista de la organización: Conseguir un producto rentable.

¿Qué consecuencias puede tener desarrollar un software con un nivel de calidad insuficiente?

A lo largo de la historia de la ingeniería de software se han registrado grandes fracasos, donde problemas o fallos del software han provocado catástrofes que han tenido consecuencias económicas graves e incluso pérdida de vidas humanas. Algunos ejemplos:

- **Ariane 5, V501:** El desbordamiento de una variable interna del sistema inercial provocó que la computadora principal, responsable del control de vuelo, desviase la lanzadera en un rápido cambio de orientación lo cual causó que se desintegrara a los 39 segundos después de su lanzamiento. REF WEB [http://www.upv.es/satelite/trabajos/pract_9/kike/paginas/accid.htm].
- **Acelerador médico Therac 25:** A causa de un error de programación, el acelerador podía exponer al paciente a una dosis letal de radiación. Este instrumento médico estuvo envuelto en al menos 6 accidentes entre 1985 y 1987 provocando la muerte de 3 de pacientes. REF WEB [<http://es.wikipedia.org/wiki/Therac-25>].

Generalmente los problemas relacionados con una falta de calidad del producto software no son tan dramáticos, pero sí que pueden ocasionar:

- Insatisfacción del cliente.
- Retrasos en el desarrollo debido a las continuas correcciones de los errores detectados. Falta de confiabilidad.
- Desviaciones del presupuesto.
- Deterioro de la imagen de la compañía.
- Pérdidas económicas

¿La Calidad se puede medir?

Queda patente que la calidad de software es algo que debería perseguir la industria, pero ¿cómo se está seguro de que un producto software tiene un nivel de calidad suficiente?

Existe un hándicap a la hora de medir la calidad del software, ya que un programa de ordenador no se fabrica, si no que se desarrolla, y por tanto no posee características físicas. Por ejemplo, la calidad de una tuerca se puede medir por su resistencia a la erosión, a la torsión o por la desviación en sus medidas, pero la calidad de una aplicación software, ¿cómo la medimos?

Existen un conjunto de normas y estándares que establecen una serie de criterios con el fin de poder determinar el grado o nivel de calidad de un determinado programa o aplicación software. Por ejemplo, el estándar ISO/IEC 25010, que se detallará más adelante, establece un marco de trabajo para facilitar este proceso.

1.2. NORMATIVA EXISTENTE. ESTÁNDARES ISO/IEC RELACIONADOS CON LA CALIDAD

A continuación se mencionan algunos estándares ISO/IEC de evaluación del producto y mejora de procesos software relacionados con la calidad:

- ISO/IEC 9001:2000. Promueve la adopción de un enfoque basado en procesos cuando se desarrolla, implementa y mejora la eficacia de un sistema de gestión de la calidad, para aumentar la satisfacción del cliente mediante el cumplimiento de sus requisitos.
- ISO/IEC 9001:2008. Modificación de la ISO/IEC 9001:2000
- ISO/IEC 9000-3:2004. Guía la aplicación de ISO 9001 para el desarrollo, la aplicación y mantenimiento de software.
- ISO/IEC 12207:1995. Define los procesos del ciclo de vida del software.
- ISO/IEC 12207:2008. Establece un marco común para los procesos de ciclo de vida de software, con terminologías bien definidas. Contiene los procesos, actividades y tareas que se aplican durante la adquisición de un producto de software o servicios y el desarrollo, operación, mantenimiento.
- ISO/IEC 9126:2001. Permite evaluar la calidad del producto software y establece las características de la calidad.
- ISO/IEC 15939:2007. Define un proceso de medición través de un modelo que define las actividades y es adaptable, flexible a las necesidades de diferentes usuarios.
- ISO/IEC 15504:2004. Proporciona un marco para la evaluación y mejora de la capacidad y madurez de los procesos. Se aplica junto ISO/OEC 12207, para evaluar y mejorar la calidad del proceso de desarrollo y mantenimiento de software.
- ISO/IEC 14598:1999. Presenta pautas que ayudan al proceso de evaluación del producto software.
- ISO/IEC 25000:2005. Proporciona una guía para el uso de las nuevas series de estándares internacionales.

1.2.1 MODELO ISO 25000:2005 PARA LA CALIDAD DEL PRODUCTO SOFTWARE

La ISO/IEC 25000, conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del software. Es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto software, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos software.

Básicamente, este nuevo estándar nace debido a la necesidad de corregir incoherencias, ampliar el alcance y crear un único referente en la evaluación de la calidad del producto software.

Esta familia se encuentra compuesta por cinco divisiones:

ISO/IEC 2500n – Gestión de Calidad

Las normas que forman este apartado definen todos los modelos, términos y definiciones comunes referenciados por todas las otras normas de la familia 25000. Actualmente esta división se encuentra formada por:

- ISO/IEC 25000 - *Guide to SQuaRE*: contiene el modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.
- ISO/IEC 25001 - *Planning and Management*: establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

ISO/IEC 2501n – Modelo de Calidad

Las normas de este apartado presentan modelos de calidad detallados, incluyendo características para calidad interna, externa y en uso del producto software. Actualmente esta división se encuentra formada por:

- ISO/IEC 25010 - *System and software quality models*: describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.
- ISO/IEC 25012 - *Data Quality model*: define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran almacenados de manera estructurada y forman parte de un Sistema de Información.

ISO/IEC 2502n – Medición de Calidad

Estas normas incluyen un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad (interna, externa y en uso) y guías prácticas para su aplicación. Actualmente esta división se encuentra formada por:

- ISO/IEC 25020 - *Measurement reference model and guide*: presenta una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad. También proporciona una guía para que los usuarios seleccionen o desarrollen y apliquen medidas propuestas por normas ISO.
- ISO/IEC 25021 - *Quality measure elements*: define y especifica un conjunto recomendado de métricas base y derivadas que puedan ser usadas a lo largo de todo el ciclo de vida del desarrollo software.
- ISO/IEC 25022 - *Measurement of quality in use*: define específicamente las métricas para realizar la medición de la calidad en uso del producto.
- ISO/IEC 25023 - *Measurement of system and software product quality*: define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.
- ISO/IEC 25024 - *Measurement of data quality*: define específicamente las métricas para realizar la medición de la calidad de datos.

ISO/IEC 2503n – División de Requisitos de Calidad

Las normas que forman este apartado ayudan a especificar requisitos de calidad que pueden ser utilizados en el proceso de elicitación de requisitos de calidad del producto software a desarrollar o como entrada del proceso de evaluación. Para ello, este apartado se compone de:

- ISO/IEC 25030 - *Quality requirements*: provee de un conjunto de recomendaciones para realizar la especificación de los requisitos de calidad del producto software.

ISO/IEC 2504n – División de Evaluación de Calidad

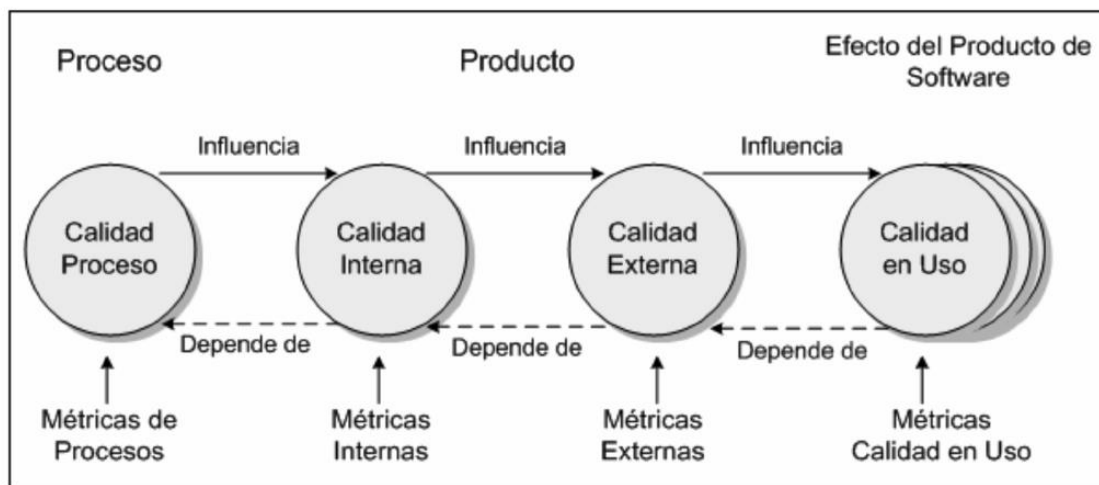
Este apartado incluye normas que proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación del producto software. Esta división se encuentra formada por:

- ISO/IEC 25040 - *Evaluation reference model and guide*: propone un modelo de referencia general para la evaluación, que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas.
- ISO/IEC 25041 - *Evaluation guide for developers, acquirers and independent evaluators*: describe los requisitos y recomendaciones para la implementación práctica de la evaluación del producto software desde el punto de vista de los desarrolladores, de los adquirentes y de los evaluadores independientes.
- ISO/IEC 25042 - *Evaluation modules*: define lo que la Norma considera un módulo de evaluación y la documentación, estructura y contenido que se debe utilizar a la hora de definir uno de estos módulos.
- ISO/IEC 25045 - *Evaluation module for recoverability*: define un módulo para la evaluación de la subcaracterística Recuperabilidad (Recoverability).

1.2.2. EL ESTÁNDAR ISO/IEC 25010. CARACTERÍSTICAS DE LA CALIDAD

Para ayudar a entender mejor el concepto de calidad del producto software, describiremos brevemente el estándar ISO/IEC 25010, el cual recoge una serie de características que debe cumplir todo software para ser considerado de calidad.

La norma ISO/IEC 25010 es una actualización del estándar ISO 9126 y establece varios puntos de vista o enfoques, desde los cuales puede describirse la calidad:



Es importante resaltar la idea de que estos “tipos de calidad” o “enfoques”, tienen una relación de influencia/dependencia unos de otros. Esto quiere decir que, si se observa una falta de calidad en el proceso de desarrollo, seguramente pueda acarrear que la calidad interna se vea afectada, deteriorando a su vez la calidad externa y en última instancia la calidad en uso del producto software.

El estándar 25010 se centra en describir la calidad del producto software, diferenciando entre calidad interna, externa y calidad en uso. A continuación se describen cada una de ellas.

Calidad de producto. Calidad Interna y externa.

- **Calidad interna:** Aquella que puede ser medible a partir de las características intrínsecas del propio producto software, tales como: especificación de requerimientos, arquitectura o diseño, el código fuente, etc. Se puede realizar en etapas tempranas del ciclo de vida del producto. Es necesario tener en cuenta que, asegurar un buen nivel de calidad interna no es sinónimo de asegurar también un alto nivel de calidad externa, a pesar de que estén relacionadas.

- **Calidad externa:** Aquella que puede ser medida y evaluada por medio de propiedades dinámicas del código ejecutable en un sistema de computación, es decir, cuando el software se ejecuta en una computadora o en una red simulando lo más cercanamente posible un escenario real.

Características relacionadas con la calidad Interna y externa.

- **Funcionalidad:** Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Completitud funcional.** Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.
 - **Corrección funcional.** Capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido.
 - **Pertinencia funcional.** Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.
- **Eficiencia de desempeño:** Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Comportamiento temporal.** Los tiempos de respuesta y procesamiento y los ratios de throughput de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas en relación con un banco de pruebas (benchmark) establecido.
 - **Utilización de recursos.** Las cantidades y tipos de recursos utilizados cuando el software lleva a cabo su función bajo condiciones determinadas.
 - **Capacidad.** Grado en que los límites máximos de un parámetro de un producto o sistema software cumplen con los requisitos.
- **Compatibilidad:** Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Coexistencia.** Capacidad del producto para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes sin detrimento.

- **Interoperabilidad.** Capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.
- **Usabilidad:** Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Capacidad para reconocer su adecuación.** Capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.
 - **Capacidad de aprendizaje.** Capacidad del producto que permite al usuario aprender su aplicación.
 - **Capacidad para ser usado.** Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
 - **Protección contra errores de usuario.** Capacidad del sistema para proteger a los usuarios de hacer errores.
 - **Estética de la interfaz de usuario.** Capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
 - **Accesibilidad.** Capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.
- **Fiabilidad:** Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Madurez.** Capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
 - **Disponibilidad.** Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
 - **Tolerancia a fallos.** Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
 - **Capacidad de recuperación.** Capacidad del producto software para recuperar los datos directamente afectados y restablecer el estado deseado del sistema en caso de interrupción o fallo.
- **Seguridad:** Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Confidencialidad.** Capacidad de protección contra el acceso de datos e información no autorizados, ya sea accidental o deliberadamente.

- **Integridad.** Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizados a datos o programas de ordenador.
- **No repudio.** Capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que dichas acciones o eventos no puedan ser repudiados posteriormente.
- **Responsabilidad.** Capacidad de rastrear de forma inequívoca las acciones de una entidad.
- **Autenticidad.** Capacidad de demostrar la identidad de un sujeto o un recurso.
- **Mantenibilidad:** Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Modularidad.** Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.
 - **Reusabilidad.** Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
 - **Analizabilidad.** Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
 - **Capacidad para ser modificado.** Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
 - **Capacidad para ser probado.** Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.
- **Portabilidad:** Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:
 - **Adaptabilidad.** Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
 - **Capacidad para ser instalado.** Facilidad con la que el producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.

- **Capacidad para ser reemplazado.** Capacidad del producto para ser utilizado en lugar de otro producto software determinado con el mismo propósito y en el mismo entorno.

Calidad de producto. Calidad en Uso.

La calidad en uso es aquella que se puede medir durante la utilización, por parte del usuario, del producto software en un entorno de producción o pre-producción. La norma ISO 25100 la define como “*grado en que un producto, utilizado por usuarios específicos, satisface sus necesidades para conseguir los objetivos establecidos con eficacia, eficiencia, flexibilidad, seguridad en uso y satisfacción en contextos de uso determinados*” [Calidad del producto y proceso software. Coral Calero Muñoz]

Características relacionadas con la calidad en uso.

Sin entrar en más detalle podríamos resumir que la calidad en uso tiene en cuenta las siguientes características durante la ejecución/utilización del producto software:

- Eficacia.
- **Productividad**: Eficiencia + Consecución de objetivos.
- **Satisfacción**: Placer de uso + Confort de uso + Confianza.
- **Seguridad en uso**: Riesgo de daño económico + Riesgo de salud + Riesgo de daño medioambiental.
- **Contexto de uso**: Flexibilidad + Accesibilidad.

1.3 CONCLUSIÓN. RESUMEN

En este capítulo nos hemos centrado en presentar el concepto de calidad a través de la descripción realizada por las normas ISO/IEC, resaltando la importancia que tiene producir software con un nivel suficiente de calidad para que éste sea rentable y cumpla con las expectativas y necesidades del cliente.

Por otro lado, es evidente que para alcanzar cierto nivel de calidad, es necesario poner en práctica modelos y metodologías que se centren no solo en cómo evaluar el software, sino también en cómo producirlo (un ejemplo es el modelo CMMI relacionado con la ingeniería de procesos) o incluso en cómo mejorar la calidad de la fase de pruebas para así asegurar la calidad del producto final (TMAP, por ejemplo es una metodología para mejorar el proceso de pruebas). Todos esos modelos o metodologías quedan fuera del alcance de este capítulo en particular y del PFC en general. El principal objetivo de este apartado era el de introducir al lector en los principales conceptos e ideas relacionados con la calidad de software.

CAPÍTULO 2

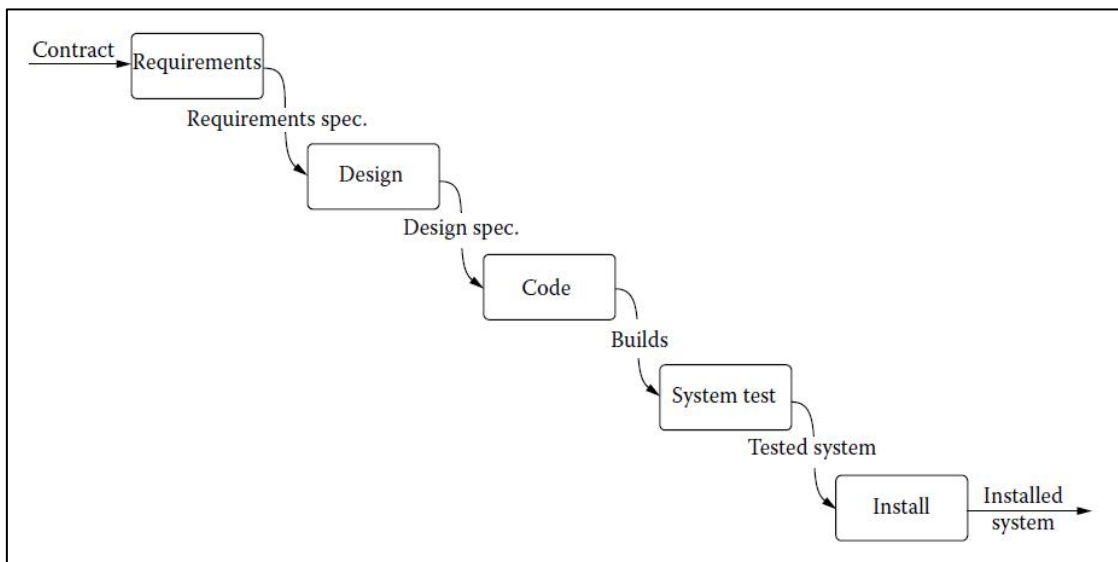
PRUEBAS SOFTWARE

2.1 INTRODUCCIÓN

Los pruebas software son de vital importancia dentro del ciclo de vida del desarrollo de un producto. Proporcionan información valiosa sobre la calidad alcanzada y son en sí mismas un mecanismo efectivo para descubrir fallos en el funcionamiento del software.

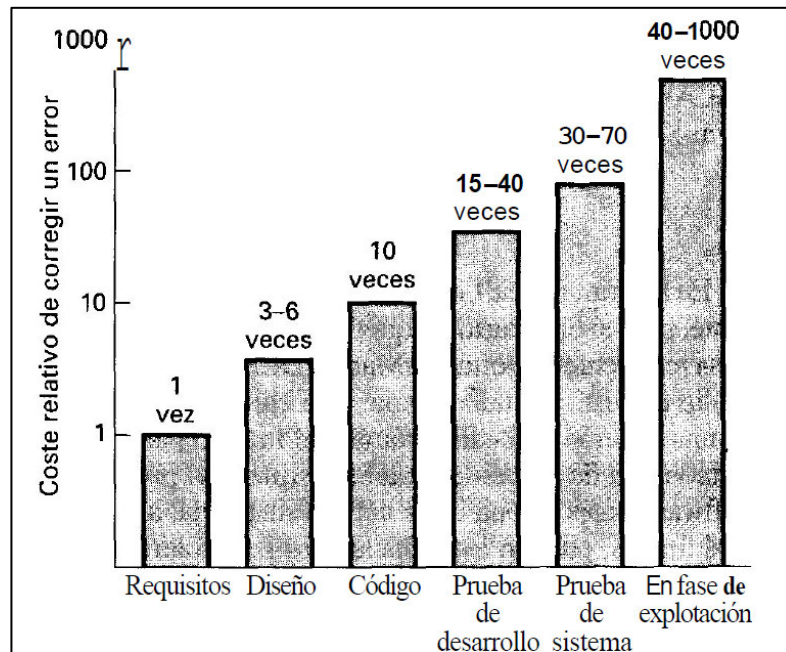
Por otro lado, es necesario entender que el proceso de pruebas por sí sólo, no aporta calidad al producto software, simplemente la confirma o no <<La calidad, si no está ahí antes de comenzar la prueba, no estará cuando termine>>, por lo tanto, un buen proceso de pruebas tiene por objetivo ayudar a tomar las decisiones necesarias para alcanzar el nivel de calidad exigido y aportar confiabilidad al software desarrollado.

Un enfoque clásico era iniciar el proceso de pruebas tras el desarrollo del producto software, con el fin de depurar la aplicación antes de entregarla al cliente.



(FIGURE 4.1 The waterfall model Manage.Software.Testing - Farrel-Vinay)

Este planteamiento presenta importantes inconvenientes. Se ha demostrado que cuanto antes se detecte un error mucho menor será el coste de corregirlo, reduciendo así su impacto en el proyecto.



(FIGURA 8.1. Coste relativo de corregir un error. Ingeniería del Software – Pressman, Roger; 6ed, capítulo 8)

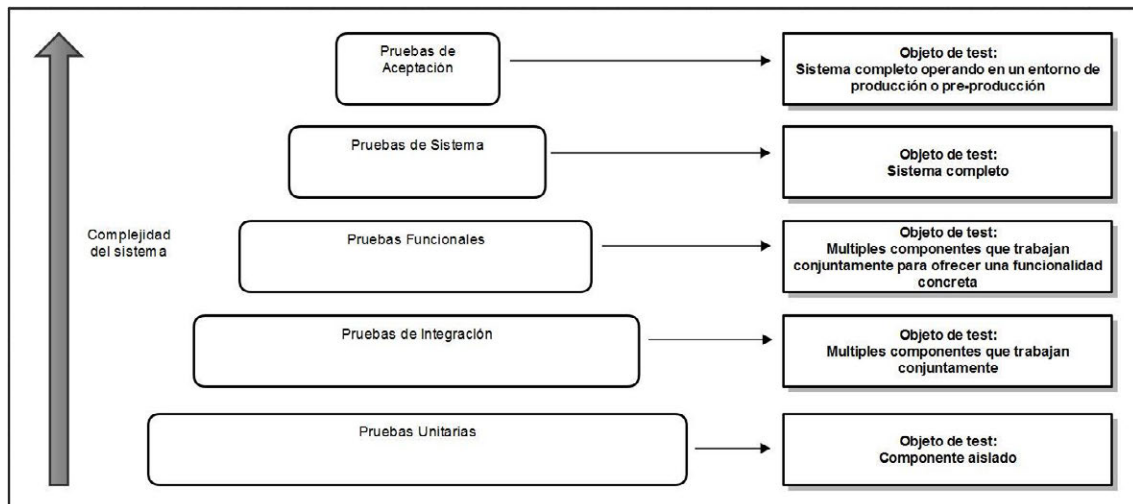
En la actualidad, sobre todo en entornos de desarrollo ágiles, el enfoque es distinto; el proceso de pruebas comienza con lo pequeño y progresa hacia lo grande. Los tests se ejecutan conforme el software se va construyendo, ejecutando un tipo u otro de pruebas, en función de cada fase del desarrollo.

2.2 NIVELES DE LAS PRUEBAS SOFTWARE

Las pruebas software se pueden realizar en diferentes momentos dentro del proceso de desarrollo, centrándose en diferentes partes del software. Por ejemplo, se pueden realizar pruebas sobre un componente aislado (test unitario), sobre dos o más componentes que operan juntos (test de integración), sobre el conjunto de componentes que brindan una determinada funcionalidad (test funcional) o sobre el sistema completo (pruebas de sistema).

Según cual sea objeto sobre el cual se realice la prueba, podemos realizar una primera clasificación de las pruebas software:

- Pruebas Unitarias.
- Pruebas de Integración.
- Pruebas Funcionales.
- Pruebas de Sistema.
- Pruebas de Aceptación.

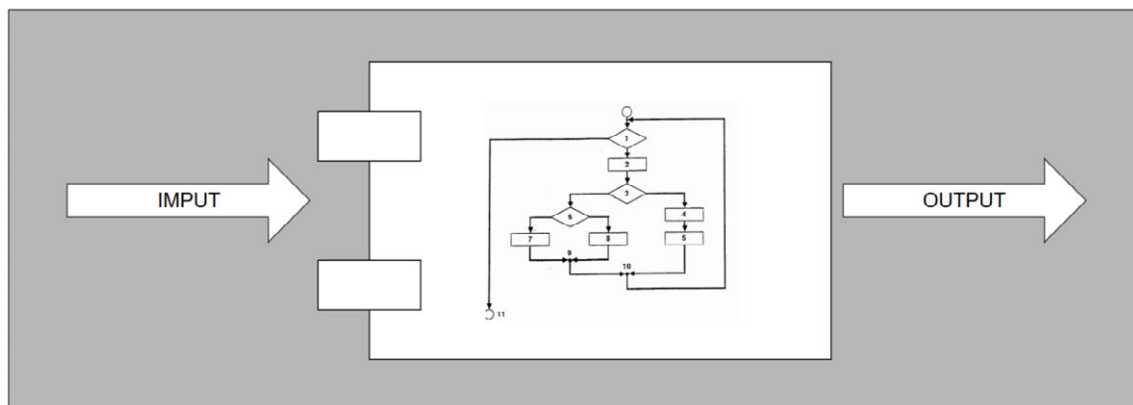


2.3 MÉTODOS Y TÉCNICAS DE PRUEBA

Para el diseño de pruebas se pueden emplear diferentes técnicas, obteniendo así varios tipos de tests. Cada técnica de pruebas está orientada a alcanzar un objetivo concreto. A continuación describiremos algunos de ellos indicando en qué nivel, de los anteriormente mencionados, puede ser interesante aplicarlos.

2.3.1 Pruebas de caja blanca

En estas pruebas se analiza la estructura de control del código fuente. También son conocidas como “pruebas de caja de cristal” ya que para su diseño es necesario hacer transparente el componente software, para que su código pueda ser explorado.



El objetivo/objetivos de este tipo de pruebas pueden ser:

1. Garantizar que se recorre por lo menos una vez todos los caminos independientes de cada componente software.
2. Recorrer todas las decisiones lógicas en sus condiciones verdadera y falsa.
3. Recorrer todos los bucles con sus límites operacionales.
4. Recorrer las estructuras internas de datos para asegurar su validez

La prueba estructural ideal sería poder analizar todos los posibles flujos de ejecución del código, pero la gran cantidad de tiempo y esfuerzo que requeriría este tipo de pruebas exhaustivas las hacen inviables, por tanto, es necesario seleccionar aquellos flujos de ejecución que tienen más probabilidad de poder encontrar algún fallo.

Criterios de cobertura lógica:

- Cobertura de sentencias.
- Cobertura de decisiones.
- Cobertura de condiciones.
- Cobertura de ciclos.

Un ejemplo de una prueba de caja blanca podría ser la “Prueba del camino básico”. Esta prueba garantiza que se ejecute, al menos, una vez cada sentencia del programa (Cobertura de sentencias). Aporta información sobre la complejidad del diseño procedimental del código que se está analizando.

Este tipo de pruebas se aplican fundamentalmente en los test unitarios, donde se prueban componentes de forma aislada.

2.3.2 Pruebas de caja negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa, por tanto no existe visibilidad sobre el código fuente del componente software analizado.



Estas pruebas pueden estar presentes en casi todos los niveles de test, siendo particularmente importantes para los 3 primeros, test unitario, de integración y funcional.

Para las pruebas unitarias y de integración, la técnica de caja negra pretende demostrar que:

- La entrada se acepta de forma correcta, no existen errores en las interfaces.
- Los componentes software analizados son operativos, no existen funciones ausentes o incorrectas.
- Se produce una salida correcta, no aparecen problemas en estructuras de datos o en accesos a bases de datos externas.
- Los componentes software no presentan errores de inicialización y terminación.
- No se producen errores de rendimiento.

Un ejemplo de técnicas de caja negra son “la partición de equivalencia” y “el análisis de valores límite”. Estas dos técnicas suelen ser muy utilizadas durante las pruebas unitarias y de integración.

En estos test los valores de entrada del componente o sistema se dividen en grupos que se espera que produzcan un comportamiento o resultado similar. Estos grupos, o clases de equivalencia, pueden ser valores de entrada válidos o valores que deban rechazarse.

El comportamiento en el límite de cada clase de equivalencia suele tener más posibilidades de ser incorrecto, por lo tanto, las pruebas prestan atención a los valores máximos y mínimos de cada grupo.

Por ejemplo, supongamos que queremos realizar una prueba de caja negra sobre un componente que realiza la división entre dos números que recibe como parámetros. Los parámetros de entrada son 2 cadenas de caracteres que representan al dividendo y al divisor y el parámetro de salida es el resultado:

- **dividendo**::[cadena de caracteres de hasta 10 dígitos. Puede utilizarse el carácter punto para representar números con hasta 3 decimales. No se permiten números negativos]
- **divisor**::[cadena de caracteres de hasta 10 dígitos. Puede utilizarse el carácter punto para representar números con hasta 3 decimales. No se permiten números negativos]

- **resultado::**[cadena de caracteres de hasta 10 dígitos. Se utiliza el carácter punto para representar números con hasta 3 decimales. Para resultados fuera de rango o indeterminados -> carácter "?". Para situaciones de error en la entrada de parámetros -> "inputerror"]

Clase de equivalencia [1] -> cadenas de caracteres sin decimales que pueden provocar resultados fuera de rango.

Valores límite para dividendo: [max] 999999999 / [min] 0

Valores límite para divisor: [max] 999999999 / [min] 0

- Prueba 1.1 -> 999999999 / 0 :: Resultado esperado: "?"
- Prueba 1.2 -> 0 / 0 :: Resultado esperado: "?"
- Prueba 1.3 -> 1 / 999999999 :: Resultado esperado: "?"
- Etc.

Clase de equivalencia [2] -> cadenas de caracteres no numéricas o con caracteres diferentes al "." que pueden provocar un "inputerror".

- Prueba 2.1 -> 10,60 / 3,5 :: Resultado: "inputerror"
- Prueba 2.2 -> -100.67 / 2 :: Resultado: "inputerror"
- Etc.

Para las pruebas funcionales también se suelen aplicar técnicas de caja negra. El principal objetivo de este tipo de tests es el de verificar que el software en construcción cumple con las especificaciones desde el punto de vista de la funcionalidad percibida por el usuario.

2.4 OTROS TIPOS IMPORTANTES DE PRUEBAS

Más adelante, durante la parte práctica del PFC, se realizarán pruebas funcionales automáticas que estarán muy relacionadas con algunos de los tipos de test que describiremos a continuación.

2.4.1 Pruebas de regresión

Cuando se efectúan cambios en el software, por ejemplo, se añaden nuevas funcionalidades, es necesario cerciorarse de que dichos cambios no afectan a otras funcionalidades o partes del código que hasta ahora estaban funcionando correctamente. Este tipo de pruebas son las conocidas como pruebas de regresión.

Conforme el software va creciendo, los ciclos de regresión son cada vez más pesados, por este motivo, la automatización de este tipo de test suele presentar grandes ventajas.

Por último, mencionar que estas pruebas pueden estar presentes prácticamente en todas las fases de construcción del software, siendo más frecuente encontrarlas en los tres primeros niveles, pruebas unitarias, de integración y funcionales.

2.4.2 Pruebas de humo

Este tipo de pruebas se realizan en fases tardías del desarrollo, y sirven para asegurarse de que la funcionalidad básica del software, o de una parte de él, se encuentra estable y responde al comportamiento esperado.

No son pruebas exhaustivas, sino tests sencillos que comprueban que ciertos caminos de la aplicación (happy path) funcionan correctamente.

Es útil pasar este tipo de pruebas antes de invertir tiempo en pruebas más detalladas, ya que con ellas se puede detectar ciertos bloqueos.

CAPÍTULO 3

AUTOMATIZACIÓN DE PRUEBAS

3.1 INTRODUCCIÓN A LA AUTOMATIZACIÓN DE PRUEBAS

¿Por qué puede ser necesaria la automatización de pruebas?

Supongamos que cierto equipo de desarrollo software tiene listo el primer prototipo completo de una aplicación web. El equipo de calidad, que es el encargado de realizar la verificación y la validación del software, realiza un pequeño análisis de los diferentes escenarios posibles, sobre los que realizar sus pruebas:

COMPONENTES	Nº	TIPOS
Plataforma HW.	2	i-386, x86_64
SSOO	4	Windows 7, Windows 10, Ubuntu 15.10, OS X 10.10
Clientes Front-End (Navegadores)	11	IED 10, IE9, IE11, Firefox 45, Firefox 44, Firefox 43, Firefox 42, Chrome 43.0, Chrome 44.0, Chrome 46.0, Safari 9.0.3
Procesos de Negocio	18	Login, Creación de proyectos, Creación de Test Suit, Creación de Test Cases, Creación de Test Plan, Administración de usuarios y Roles, Ejecución de pruebas, Generación de Informes, etc...
Data Sets	20	Usuarios con distintos Roles, Contraseñas, Cadenas de búsqueda, etc.
Nº Total de Escenarios de Test	2x4x11x18x20	¡31.680 escenarios!

Como es lógico, abordar la ejecución manual de 31.680 escenarios de test, con sus correspondientes pruebas es algo muy costoso. El tiempo dedicado a las pruebas podría ser varias veces superior al necesario para el desarrollo de la aplicación.

¿Cuál puede ser la solución? ¿Recortar el número de pruebas para ahorrar tiempo? ¿Aumentar el tiempo de puesta en producción de la aplicación?

Si se disminuye la cantidad de pruebas durante el proceso de verificación y validación del software puede suceder que no se detecten ciertos fallos o anomalías en el funcionamiento de la aplicación, lo que conduciría a entregar

un producto con un nivel de calidad insuficiente. Por otro lado, si el tiempo necesario para alcanzar el nivel de calidad deseado es muy elevado, puede provocar que los costes aumenten y que la viabilidad del proyecto peligre.

Para este tipo de situaciones una solución podría ser automatizar un porcentaje de los tests, de forma que el tiempo de ejecución de la fase de pruebas se redujese.

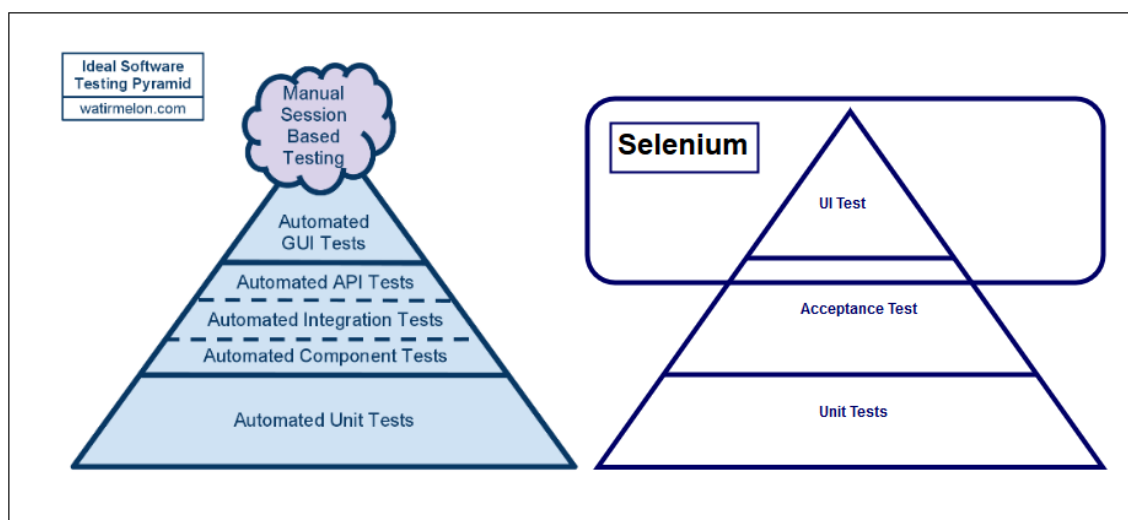
Otro ejemplo donde la automatización puede ser necesaria, es cuando diferentes versiones de un software van siendo liberadas y las nuevas funcionalidades deben ser probadas en conjunto con las ya existentes, generándose gran cantidad de pruebas de regresión y creando ciclos de pruebas cada vez más grandes. Si las pruebas que se deben ejecutar cada vez que se libera una versión están automatizadas la carga de trabajo para el equipo de pruebas disminuye considerablemente, pudiendo centrar su esfuerzo sobre todo en las nuevas funcionalidades incorporadas.

¿Qué es exactamente la automatización de pruebas?

En el ámbito de las pruebas software, cuando utilizamos el termino automatizar, nos referimos a conseguir que las pruebas que se realizan de forma manual puedan ser ejecutadas de forma desatendida (sin intervención del tester), por medio de alguna herramienta que realice el proceso automáticamente.

3.2 NIVELES DE AUTOMATIZACIÓN

Según la pirámide de Cohn descrita en su libro *Succeeding with Agile*, aparecen 3 grandes niveles de automatización, la automatización de los test unitarios, la de los test de aceptación y la de los tests de interfaz de usuario. Esta pirámide también representa el esfuerzo o la cantidad de test automáticos que debería tener cada nivel, empezando por el más bajo, donde se debería concentrar el mayor número de test automáticos, hasta llegar al tercer nivel, el de la interfaz de usuario, donde el número de pruebas, por su complejidad, debería ser menor.



El conjunto de herramientas que nos ofrece Selenium sirve para cubrir toda la automatización del nivel 3 (Interfaz de usuario) y parte del nivel 2 (test de aceptación), de una aplicación con interfaz web. Por tanto, en capítulos posteriores se profundizará en la automatización de pruebas funcionales sobre aplicaciones con interfaz web.

3.3 BENEFICIOS DE LA AUTOMATIZACIÓN DE PRUEBAS

La automatización de pruebas puede traer consigo una serie de beneficios que podrían ayudar a elevar el nivel de calidad del producto software y también a disminuir su coste. Los más destacados podrían ser:

- Se pueden ejecutar un número mayor de pruebas.
Una vez que se ha automatizado un determinado caso de prueba, es fácil ejecutarlo sucesivas veces variando solo sus datos de entrada.
Consecuencia: La cobertura aumenta, se prueban muchas más combinaciones, aumentando el nivel de confiabilidad del producto software.
- Se pueden ejecutar pruebas de forma desatendida.
La ejecución se puede lanzar a cualquier hora del día, durante la noche, en periodos no laborables, etc.
Consecuencia: Se reduce considerablemente el tiempo de la fase de ejecución de pruebas. Menor coste.
- Reducción de errores durante la ejecución de las pruebas.
Una prueba automatizada se ejecuta siempre de la misma manera, mientras que en una ejecución manual se pueden cometer errores, sobre todo, dependiendo del nivel del tester que realice dicha ejecución y de la complejidad de la prueba.
Consecuencia: Aumenta la calidad de las pruebas. Posible aumento también de la calidad general del producto software.
- Ayuda a estandarizar procesos.
Implantar un sistema de automatización de pruebas pasa por analizar bien los procesos que se realizan durante la ejecución de pruebas, lo que ayuda a estandarizarlos y a crear una fase de pruebas más consistente.
Consecuencia: Aumenta la calidad de las pruebas. Posible aumento también de la calidad general del producto software.

- Facilita las pruebas de regresión.

La automatización de pruebas favorece que las pruebas de regresión se realicen sin apenas coste, ya que la repetición de pruebas no supone una inversión grande de tiempo.

Consecuencia: Se reduce el coste durante las fases de mejora o evolución del producto software.

También hay que tener en cuenta diversos factores antes de decidir si merece la pena invertir tiempo y dinero en automatizar pruebas. Si el proceso de automatización no se realiza adecuadamente muchos de los posibles beneficios que en teoría debería aportar se pueden volver en nuestra contra.

Por ejemplo, seguramente se necesite personal más cualificado para realizar la automatización. La falta de experiencia puede provocar que las pruebas no estén bien diseñadas, que surjan problemas técnicos que alarguen el proceso de creación de los test automáticos, etc., provocando que el nivel de calidad de las propias pruebas sea insuficiente, aportando mucha incertidumbre sobre el nivel de calidad del producto software final.

Otro problema puede venir de una mala decisión sobre lo que se automatizará y sobre lo que no. Existen algunos procesos que por su complejidad es posible que sea más costoso automatizarlos que realizarlos manualmente. Es necesario saber bien dónde invertir los esfuerzos de automatización.

3.4 CLAVES PARA REALIZAR UNA AUTOMATIZACIÓN PRODUCTIVA

Lo principal para que la automatización pueda contribuir al éxito del proyecto, es elegir una buena estrategia de pruebas, identificando las partes donde puede suponer un beneficio el realizar procesos automáticos. Por ello, es necesario tener en cuenta que:

- No todo se puede automatizar. Un objetivo realista podría ser entorno al 50 % de las pruebas.
- Comenzar automatizando las tareas repetitivas más básicas que consumen una cantidad significativa de tiempo.
- La automatización se basa en la reutilización, si un test se diseña para ser ejecutado una sola vez, no tiene sentido automatizarlo.
- Los test que necesitan ejecutarse para cada compilación son los mejores candidatos para ser automatizados.
- Los test que utilizan múltiples valores para las mismas tareas, también son buenos candidatos.
- En cada caso, es necesario realizar una valoración de si realmente la automatización puede acarrear algún beneficio frente al testing manual.
- Los test con resultados NO predecibles deben ser descartados para el proceso de automatización.

3.5 CONCLUSIONES

A modo de resumen, a la hora de abordar el proceso de automatización de pruebas es necesario asegurarse de:

- Definir una buena estrategia.
- Tratar el proceso de automatización de pruebas cómo un proceso de desarrollo software.
- Contar con un equipo bien cualificado con sólidos conocimientos en programación.

También es necesario no perder de vista los objetivos que persigue la automatización:

- Ahorro de tiempo. Mejorar el tiempo de puesta en producción del software y mejorar el tiempo dedicado a las pruebas asociadas a la evolución y mejora del producto (tests de regresión y sanity test).
- Reducción de costes.
- Aumento de la calidad de las pruebas. Pruebas más consistentes, más sólidas. Asegurar que la ejecución se realiza siempre del mismo modo.
- Aumento de la calidad del producto software. Gracias al aumento de la cobertura proporcionada por los test automáticos.
- Liberar a los testers de tareas repetitivas y tediosas, permitiéndoles concentrarse en lo que verdaderamente importa, la funcionalidad de negocio que debe cumplir el software desarrollado.

Por tanto, la automatización puede ser beneficiosa, siempre y cuando se lleve a cabo de una manera inteligente y bien planificada, entendiendo que multiplicar el número de pruebas ejecutadas no siempre es sinónimo de mejorar la confiabilidad del software testado, y debe ser tomada siempre como un complemento de la ejecución manual.

“Un tonto con una herramienta sigue siendo un tonto”

[Grady Booch](#) (n. 1955), ingeniero de software estadounidense, co-autor del Lenguaje Unificado de Modelado ([UML](#))

CAPÍTULO 4

INTRODUCCIÓN A SELENIUM

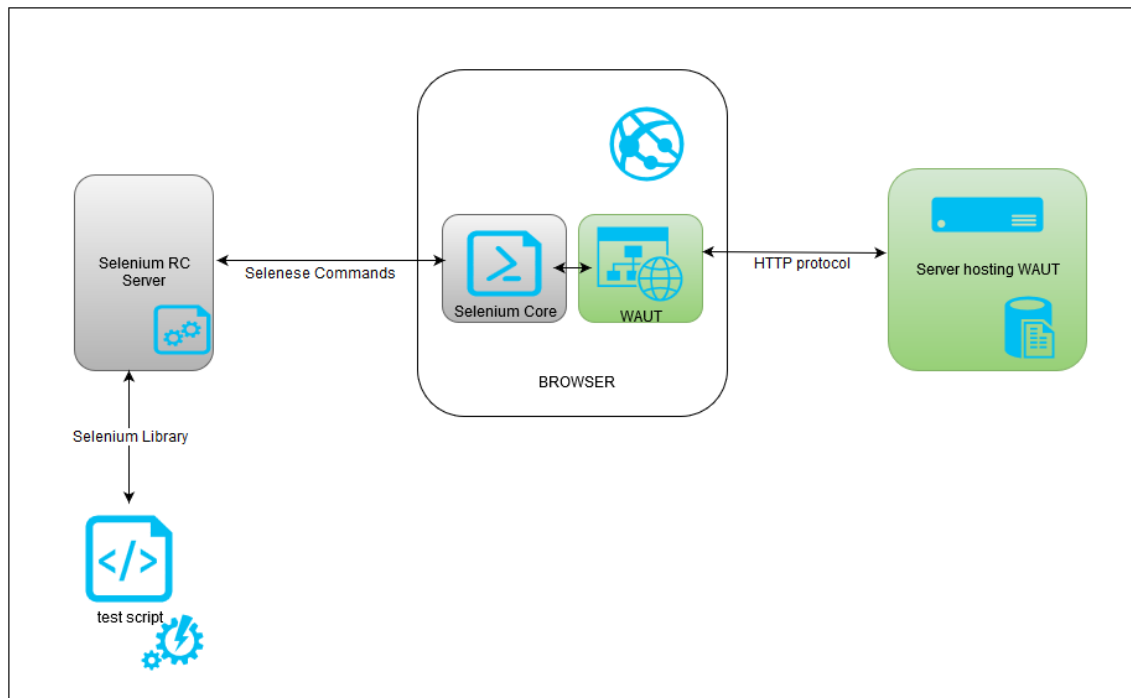
4.1 SELENIUM 1.0 (SELENIUM RC)

Selenium nace con el objetivo de ofrecer una librería de funciones, que permita automatizar pruebas de interfaz de usuario sobre aplicaciones web. El concepto es sencillo, un test escrito en un lenguaje de alto nivel, haciendo uso del API ofrecida por Selenium, es capaz de interactuar con una aplicación web (WAUT -Web App under Test -) a través de un navegador, de forma similar a como lo haría un usuario real.

Selenium 1.0 o Selenium RC (Remote Control) es la primera versión de Selenium. Desde el punto de vista técnico, Selenium RC utiliza dos componentes, Selenium Remote Control Server y Selenium Core. Estos dos componentes se comunican entre sí mediante una serie de comandos llamados “selenese commands”.

Selenium Core, por su parte, es un componente escrito en JavaScript que se ejecuta dentro del navegador y se encarga de ir dirigiendo la navegación sobre la aplicación web bajo test - a partir de ahora llamada WAUT (Web Application Under Test)-.

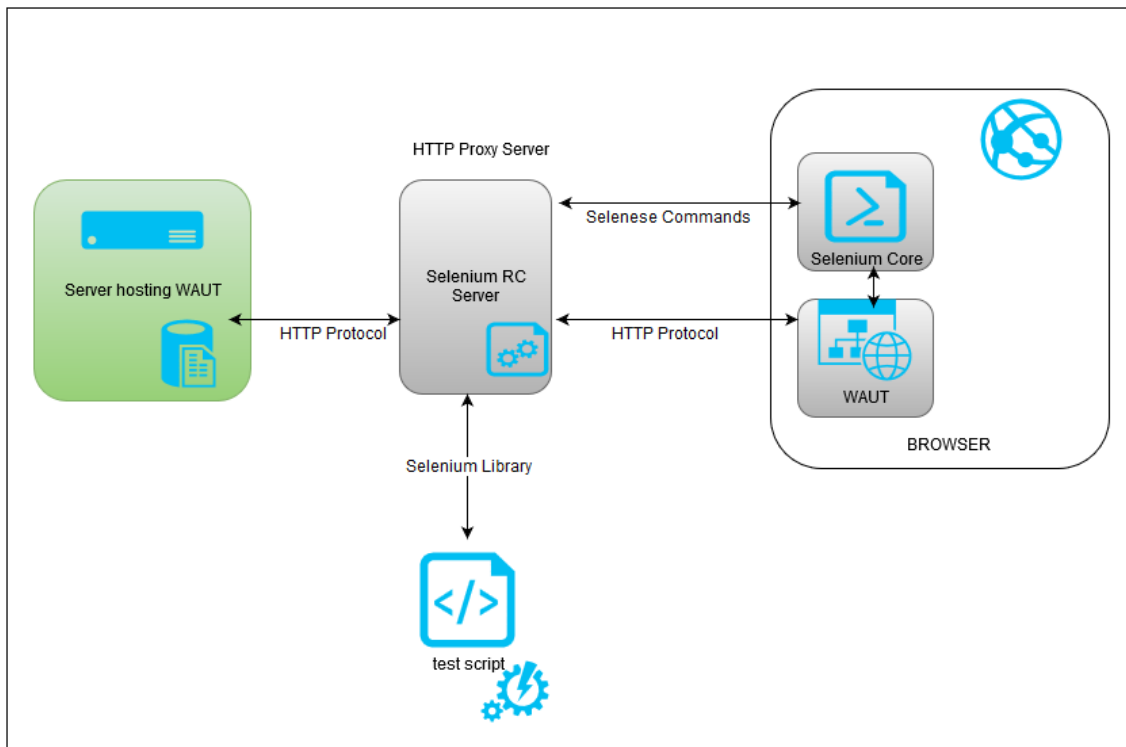
Por otro lado, Selenium RC Server es el encargado de inyectar el código de Selenium Core en el navegador y traducir las funciones de librería a comandos que intercambia con Selenium Core (Selenese Commands) para dirigir la navegación sobre una determinada WAUT.



Esta filosofía de funcionamiento plantea un problema de seguridad en la mayoría de los navegadores, ya que vulnera una importante medida de seguridad llamada “Política del mismo origen” en la cual, NO se debe permitir que código perteneciente a un determinado dominio web pueda ejecutar o interactuar con código perteneciente a otro dominio web diferente.

En el diagrama anterior, el navegador interpreta que el código Javascript del componente Selenium Core pertenece a un dominio diferente al código que compone la WAUT, por lo que realiza su ejecución de manera aislada, no permitiendo la interacción entre ambos.

Para poder evitar esta restricción de seguridad, el componente Selenium RC Server actúa como servidor Proxy situándose entre el navegador y el servidor real de la aplicación web.

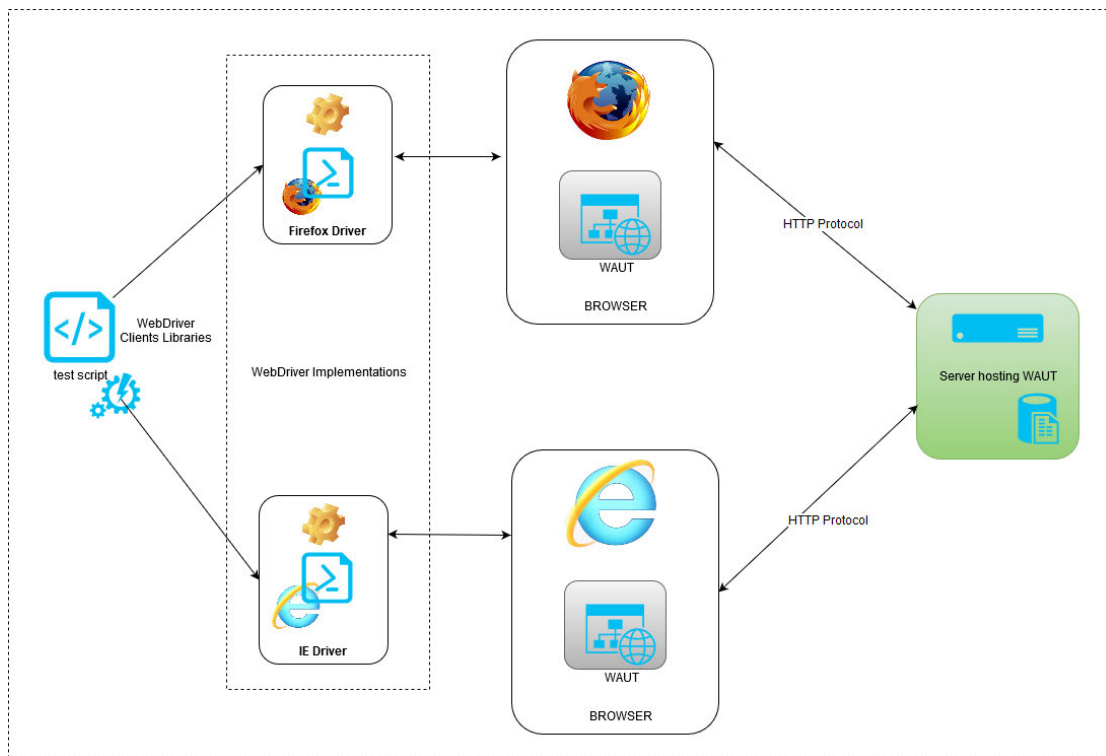


En esta última configuración, el navegador entiende que tanto Selenium Core, como la WAUT tienen el mismo origen y por lo tanto pertenecen al mismo dominio, permitiendo que el código JavaScript inyectado por Selenium RC Server pueda interactuar con el código de la página web.

4.2 SELENIUM 2.0 (SELENIUM WEBDRIVER)

La primera versión de Selenium fue un buen punto de partida en el camino de la automatización de pruebas funcionales sobre aplicaciones web, pero pronto se hicieron patentes ciertas limitaciones que se vieron corregidas ya en su segunda versión, Selenium 2.0 o Selenium WebDriver, la cual, es el objeto de estudio del PFC.

Selenium WebDriver modifica el paradigma planteado en su versión anterior para conseguir una herramienta más flexible y mucho más potente. Un poco más adelante analizaremos las mejoras que aporta la versión 2.0 frente a su predecesora, Selenium RC, ahora vamos a centrarnos en ver qué modificaciones hay en su planteamiento.



En esta nueva versión, WebDriver emplea implementaciones nativas específicas para cada navegador, evitando así las limitaciones que suponía el uso de código JavaScript para dirigir las acciones sobre la WAT.

Ahora cada implementación particular de WebDriver maneja instrucciones nativas de un navegador específico, manteniendo una interfaz de acceso común que permite a un script escrito en un lenguaje de alto nivel simular diferentes acciones sobre una página web, siendo transparente para él en qué navegador se ejecutan.

Si comparamos las dos versiones de Selenium, Selenium RC y Selenium WebDriver, podemos ver que:

- Selenium WebDriver supera las limitaciones de su predecesor, a la hora de poder simular las diferentes acciones que realiza un usuario en el navegador, durante la ejecución de una determinada aplicación web. Ahora Selenium es mucho más potente al utilizar comandos nativos de cada navegador y eliminar el uso del componente Selenium Core que utilizaba instrucciones JavaScript, con las restricciones que eso implicaba.
- WebDriver aísla al programador de las particularidades de cada navegador. Selenium RC podía presentar diferentes comportamientos en función del navegador utilizado para la ejecución de los tests. Algunas pruebas que funcionaban correctamente sobre Firefox no lo hacían en otros navegadores, y viceversa. Ahora existe una implementación nativa de los principales browsers, aislando a los usuarios del API de las particularidades del uso de cada navegador.
- La librería que ofrece la versión 2.0 de Selenium es más completa y más potente. El API de WebDriver está disponible para varios lenguajes de alto nivel (Java, Ruby, Python...) y es orientada a objetos, lo que facilita su versatilidad e integración.
- Selenium WebDriver soporta el testing de aplicaciones móviles. Mediante el uso de iPhoneDriver/AndroidDriver se pueden ejecutar test automáticos sobre aplicaciones móviles en los diferentes simuladores/emuladores.

Para terminar de hablar sobre la última versión de Selenium, es necesario añadir también ofrece mecanismos para desacoplar la ejecución del test de la ejecución del navegador. Esto quiere decir lo siguiente, supongamos que nuestros tests se encuentran en la máquina A, la cual funciona bajo un entorno Windows donde se encuentran diferentes herramientas que se utilizan para las pruebas. Por otro lado, la aplicación web que queremos probar debe funcionar en un navegador que se ejecuta en una distribución Linux concreta instalada en la máquina B. En este caso concreto ¿Sería necesario que los test fuesen lanzados desde la máquina con la distribución Linux (máquina B)? La respuesta a la pregunta es, NO. Los test podrían ser lanzados desde la máquina A. Como veremos en capítulos posteriores, Selenium WebDriver ofrece la posibilidad de dirigir las ejecuciones de navegadores que se encuentren funcionando en máquinas remotas (ejecución distribuida). Esto se consigue gracias a las herramientas Selenium RemoteWebDriver y Selenium Grid.

PARTE 2ª. ESTUDIO DE SELENIUM

CAPÍTULO 5

OBJETIVOS Y ESTRUCTURA DEL ESTUDIO

5.1 OBJETIVOS DEL ESTUDIO

El principal objetivo del estudio del API que ofrece Selenium 2.0 es presentar y describir sus principales clases y métodos, creando un pequeño manual de referencia que sirva para iniciarse en el empleo de WebDriver para la automatización de pruebas funcionales.

Los siguientes capítulos sientan las bases para desarrollar la tercera parte del proyecto, en la cual, se creará un entorno de pruebas para la automatización de tests funcionales sobre una aplicación con interfaz web.

A lo largo de esta segunda parte, se proporcionaran pequeños ejemplos de código, para facilitar al lector la comprensión y el manejo de los principales métodos de las clases estudiadas.

5.2 ESTRUCTURA DEL ESTUDIO

Esta segunda parte del PFC estará dividida en varios capítulos con el objetivo de poder estructurar toda la información contenida en el API de Selenium WebDriver, ayudando así a poder manejarla de una forma más sencilla. A continuación se exponen las diferentes líneas de trabajo:

- Establecer las herramientas y programas necesarios para poder trabajar con Selenium de una manera eficaz.
- Conocer la interfaz WebDriver y sus principales métodos. ¿Cómo obtener el contenido de una determinada página web? ¿Cómo trabajar con los elementos que muestra?
- Conocer qué acciones pueden ser simuladas mediante Selenium. Acciones realizadas con el ratón (clic, doble clic, arrastrar objetos, etc.) y acciones realizadas con el teclado (introducir texto, simular determinadas teclas, etc.)
- Conocer que funcionalidades nos puede ofrecer Selenium desde el punto de vista de la ejecución de pruebas.
- Conocer las características de la implementación de la interfaz WebDriver para un navegador específico, en este caso Firefox.

CAPÍTULO 6

ENTORNO Y HERRAMIENTAS NECESARIAS

Para poder empezar a trabajar con Selenium, necesitamos primero descargar la librería y contar con un entorno donde poder desarrollar y ejecutar los tests. En este capítulo veremos cómo y de donde descargar la librería de Selenium WebDriver, cómo crear el proyecto en Eclipse y algunas otras herramientas del navegador Firefox que nos facilitarán ciertas tareas cuando necesitemos conocer con detalle el código HTML de la página web sobre la que estemos trabajando.

6.1 DESCARGA DE LA LIBRERÍA WEBDRIVER

El primer paso que debemos dar para empezar a usar WebDriver es obtener la librería que contiene el API. Desde la página de Selenium nos podemos descargar su última versión < <http://www.seleniumhq.org/download/> >. Este API está disponible para varios lenguajes de programación, en nuestro caso, como el PFC se desarrollará en Java, elegiremos esta opción.

Haciendo clic en el enlace “[Download](#)” procedemos a descargar el archivo selenium-java-2.53.0.zip que contiene la librería de Selenium WebDriver. Descomprimos el archivo en el directorio donde queramos dejarla accesible.

SeleniumHQ
Browser Automation

edit this page search selenium: Go

Projects **Download** Documentation Support About

Selenium Downloads

- [Previous Releases](#)
- [Source Code](#)
- [Maven Information](#)

Donate to Selenium

with PayPal

[Donate](#)

through sponsorship

You can [sponsor the Selenium project](#) if you'd like some public recognition of your generous contribution.

Selenium Sponsors

See who [supports the Selenium project](#).

BrowserStack

SAUCE LABS

experitest
Selenium for Mobile

Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of [previous releases](#), [source code](#), and additional information for [Maven users](#) (Maven is a popular Java build tool).

Selenium Standalone Server

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium WebDriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

Download version [2.53.0](#)

To use the Selenium Server in a Grid configuration [see the wiki page](#).

The Internet Explorer Driver Server

This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your \$PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.53.0 for (recommended) [32 bit Windows IE](#) or [64 bit Windows IE](#)

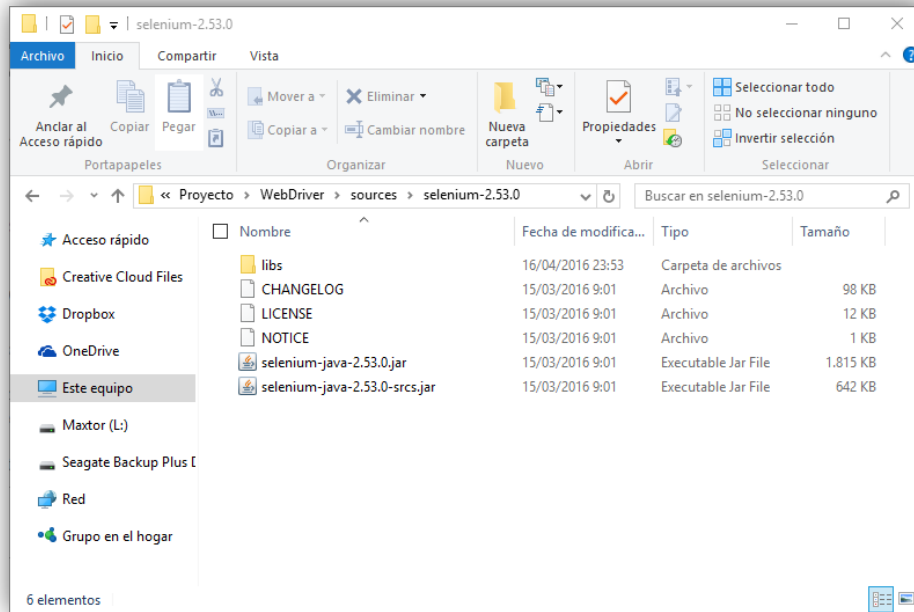
[CHANGELOG](#)

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for [other languages exist](#), these are the core ones that are supported by the main project hosted on google code.

Language	Client Version	Release Date	Download	Change log	API docs
Java	2.53.0	2016-03-15	Download	Change log	Javadoc
C#	2.53.0	2016-03-16	Download	Change log	API docs
Ruby	2.53.0	2016-03-15	Download	Change log	API docs
Python	2.53.0	2016-03-15	Download	Change log	API docs
Javascript (Node)	2.53.1	2016-03-15	Download	Change log	API docs

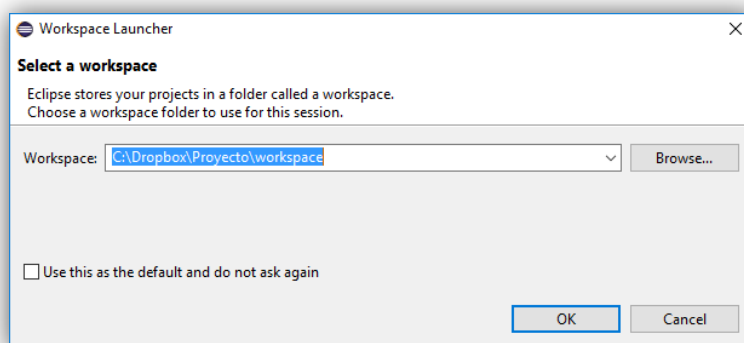


6.2 CREACIÓN DE UN PROYECTO EN ECLIPSE

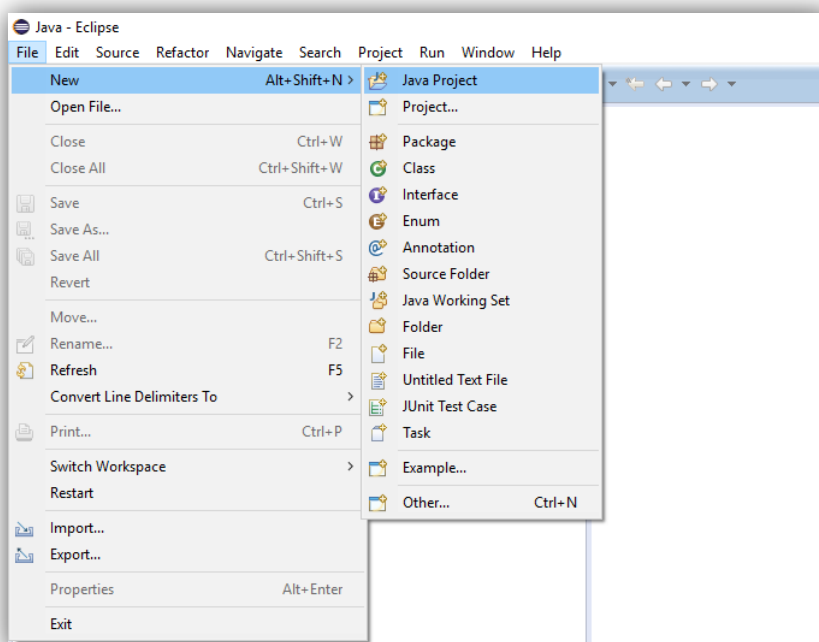
A la hora de crear los scripts y ejemplos en Java, se utilizará Eclipse como entorno de desarrollo. Cabe destacar que, el uso de la librería Java que nos proporciona Selenium en ningún caso está vinculado al uso de un entorno o framework de desarrollo concreto. Se ha decidido emplear Eclipse por su popularidad, por ser un software bajo licencia open source y por el amplio soporte de que dispone.

Desde la URL < <https://eclipse.org/downloads/> > podemos descargar la versión que necesitamos. En nuestro caso se ha optado por utilizar la versión Mars 1, del paquete Eclipse IDE for Java Developers.

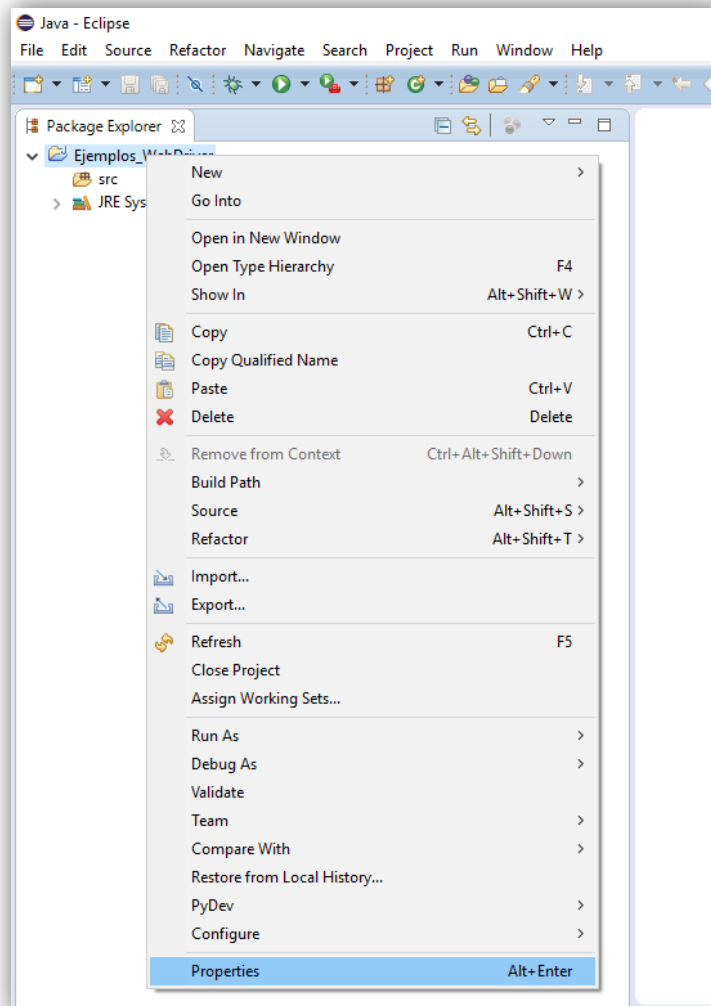
Una vez que ya dispongamos de Eclipse, al iniciar la aplicación se nos preguntará por el directorio de trabajo que deseamos utilizar. Eclipse lo llama workspace.



Con la perspectiva Java activa, crearemos un nuevo proyecto desde la barra de menú superior.

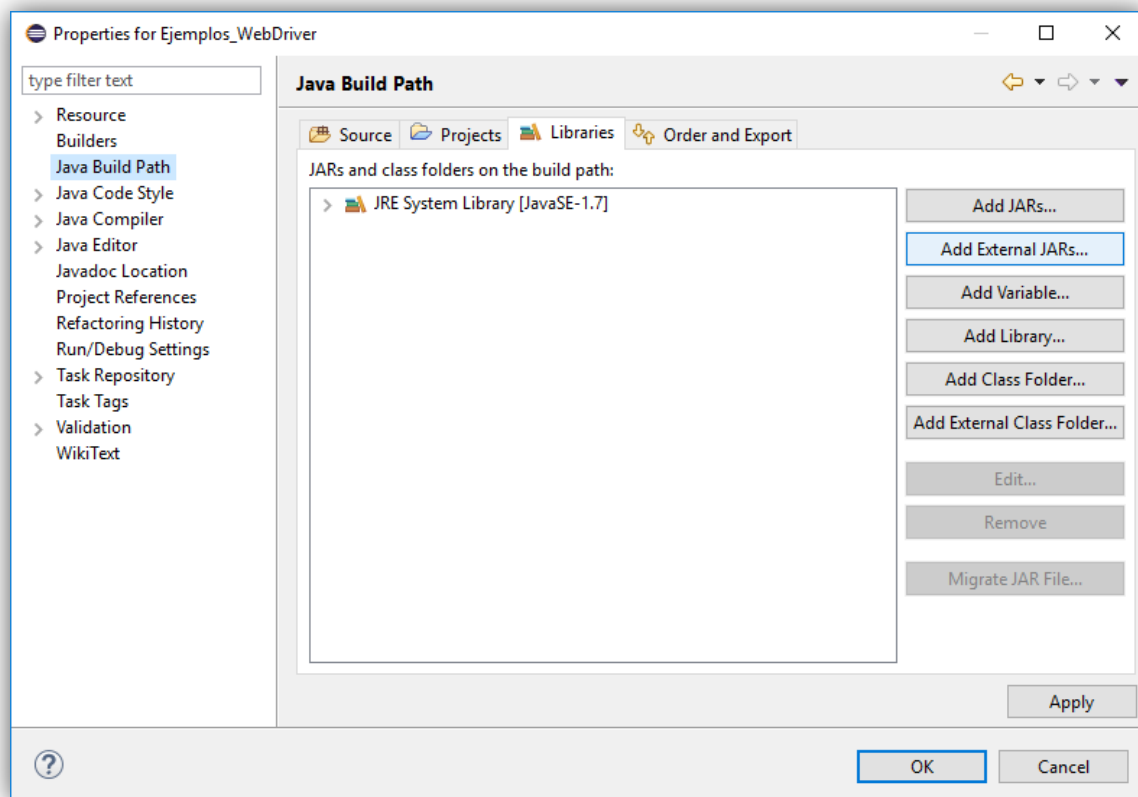


Por último, para hacer uso de la librería de Selenium descargada en el apartado anterior, es necesario indicarle a Eclipse cuál es su ubicación. Para ello, hacemos clic con el botón derecho sobre el proyecto que acabamos de crear y seleccionamos-> Properties.



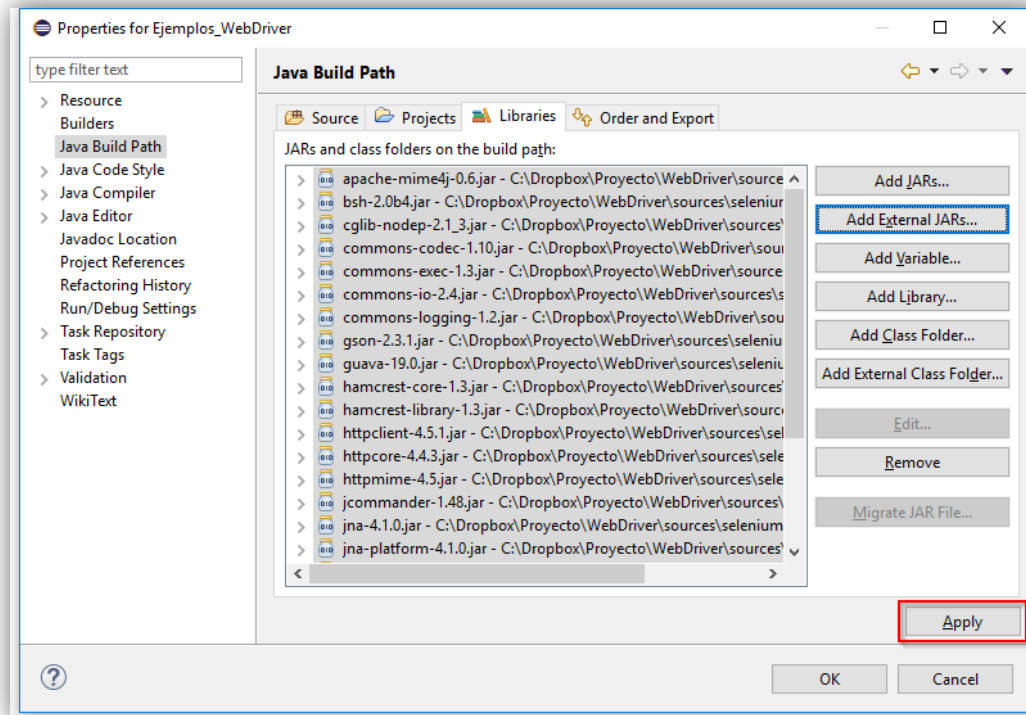
Dentro de la pantalla de propiedades, en el menú derecho, seleccionamos la opción “Java Build Path”.

Ahora, en la pestaña “Libraries”, pulsamos el botón derecho “Add External JARs...”



Por último, seleccionamos los archivos *selenium-java-2.53.0.jar*, *selenium-java-2.53.0-srcs.jar* y todos los contenidos en la carpeta *libs*.

Pulsando el botón “Apply” terminamos este proceso.



6.3 FIREFOX Y ALGUNOS COMPLEMENTOS DE UTILIDAD

El navegador web utilizado en los ejemplos será Firefox, ya que, nos ofrece varios complementos que nos ayudarán a la hora de trabajar con el código HTML. En la siguiente URL podemos descargarlo < <https://www.mozilla.org/en-US/firefox/all/#es-ES> >.

Como ya hemos comentado, para facilitarnos la interacción con el código HTML de las páginas web, además del propio navegador, necesitaremos algunos complementos. Estos son Firebug y FirePath.

Firebug lo podemos obtener de:

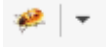
- < <https://addons.mozilla.org/es/firefox/addon/firebug/> >

y Firepath de:

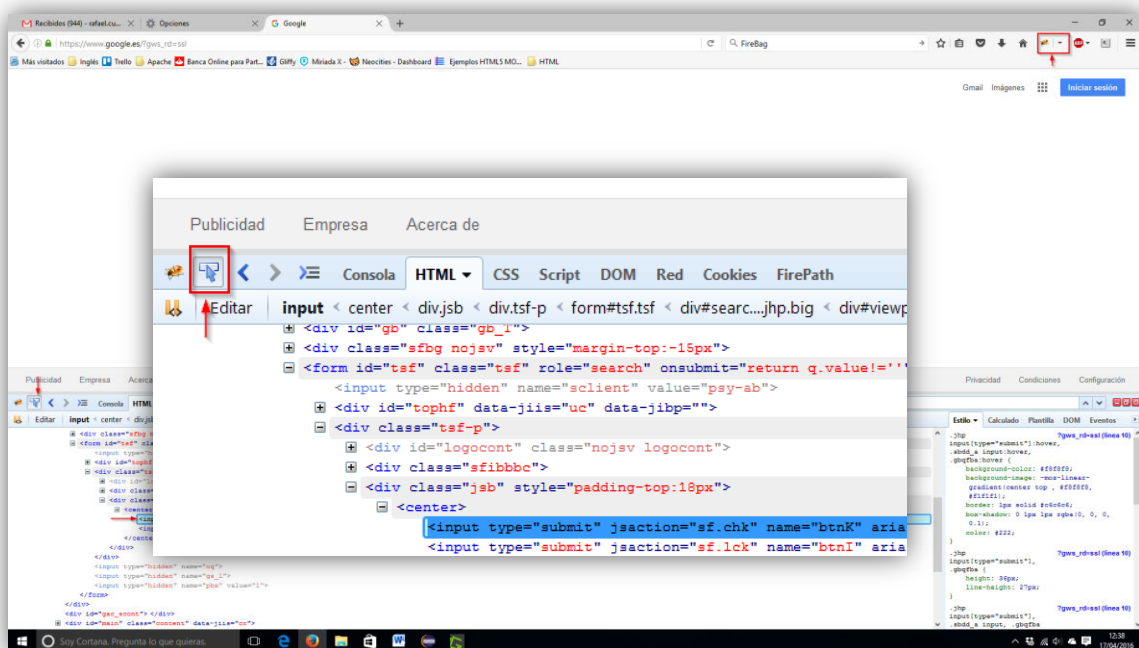
- < <https://addons.mozilla.org/es/firefox/addon/firepath/> >.

Firebug

Una vez que tengamos instalado este complemento, en la barra superior de menú, nos aparecerá a la derecha un botón como el siguiente:

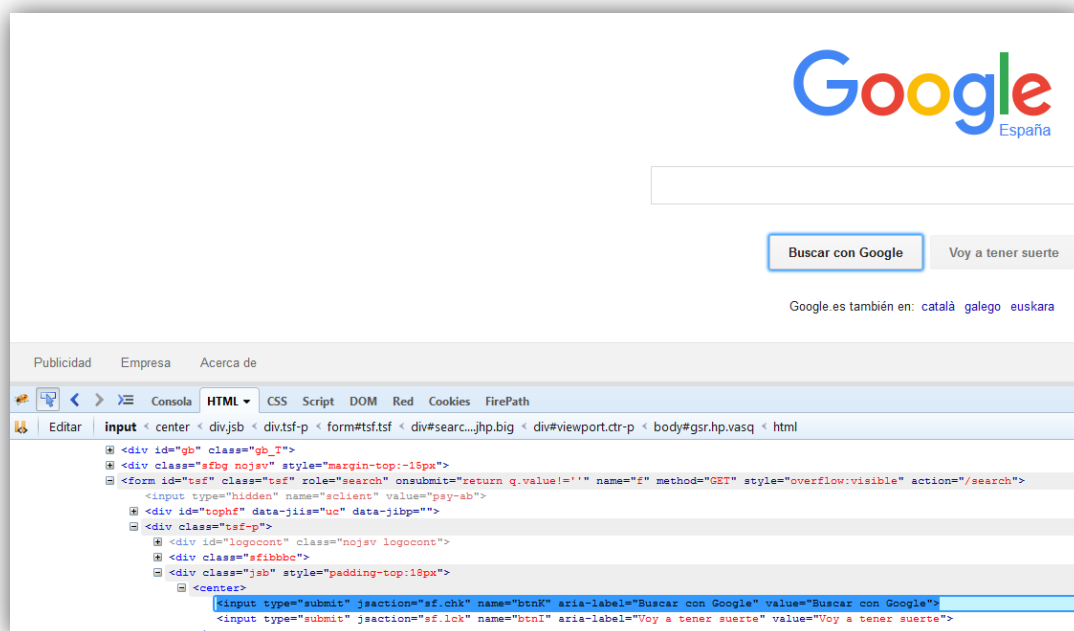


Firebug nos permitirá analizar el código HTML de la página que este mostrando el navegador en ese momento, pudiendo localizar cada uno de los elementos de la página web. En la siguiente imagen, estamos localizando el botón “Buscar con Google”.



Para ello, después de activar el complemento, pulsando sobre el icono del insecto en el menú de la ventana inferior, debemos seleccionar la función de localización.

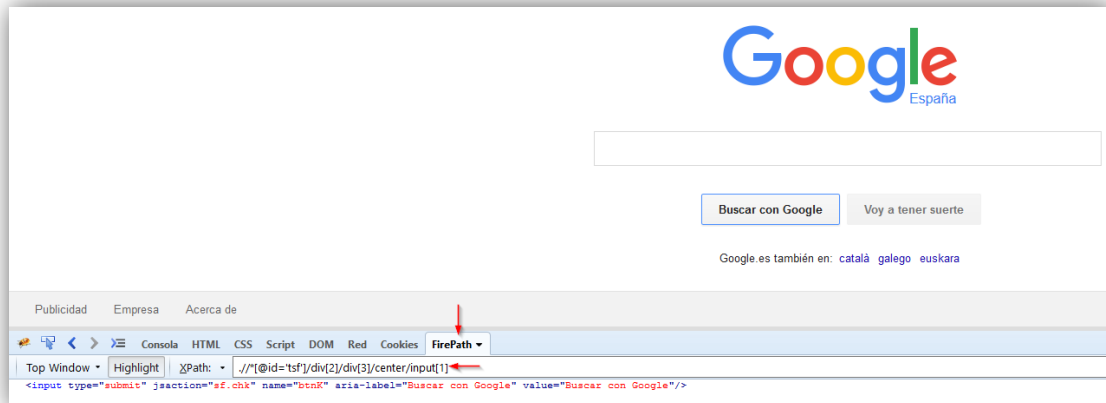
Esta función nos permitirá localizar en el código HTML cada uno de los elementos que visualmente percibimos en el navegador, seleccionándolos con el cursor del ratón. En nuestro ejemplo, después de activar esta función, si situamos el cursor del ratón sobre el botón de “Buscar con Google”, el complemento de Firebug nos indica el código HTML que hace referencia a ese elemento.



FirePath

FirePath es un complemento que añade una importante función a Firebug, permitiendo localizar elementos de una página web mediante su ruta XPath.

En el ejemplo anterior, si pulsamos sobre la opción FirePath del menú de la ventana de Firebug, podemos ver la dirección XPath que hace referencia al botón “Buscar con Google” `//*[@id='tsf']/div[2]/div[3]/center/input[1]`



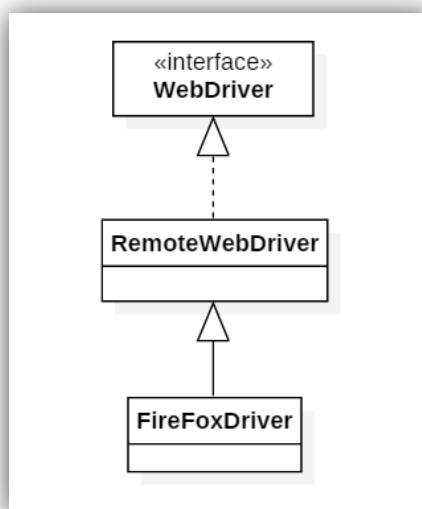
Todas estas funciones ofrecidas por Firebug + FirePath, nos serán de gran ayuda como veremos en capítulos posteriores.

CAPÍTULO 7

INTERFAZ WEBDRIVER. LOCALIZACIÓN DE LOS ELEMENTOS DE UNA WEB

En este capítulo nos iniciaremos en el estudio del API Java que ofrece Selenium WebDriver para poder interactuar con una aplicación web tal y cómo lo haría un usuario con su navegador. Empezaremos describiendo algunos métodos de la interfaz WebDriver que nos permitan cargar el contenido de una página web y seleccionar sus elementos para poder realizar diferentes acciones sobre ellos.

La interfaz WebDriver define una serie de funcionalidades comunes para poder trabajar con los diferentes navegadores web.



La clase RemoteWebDriver implementa, entre otras, la interfaz WebDriver, y la clase FireFoxDriver realiza una especialización mediante herencia para poder adaptar el comportamiento de la interfaz original (WebDriver) a un navegador específico, en este caso Firefox.

7.1 INTERFAZ WEBDRIVER. METODOS PARA OBTENER EL CONTENIDO DE UNA WEB

El primer paso para poder realizar acciones sobre una aplicación web es poder acceder a ella, es decir, necesitamos cargar la página en un navegador.

Para poder trabajar con el contenido de una web, la interfaz WebDriver define una serie de métodos:

MÉTODO get()

Return	Método(Parámetros)
void	get(java.lang.String url)
Descripción	
<ul style="list-style-type: none">- Método que se encarga de acceder a la url que se le pasa como parámetro y cargar su contenido en una ventana del navegador web.- Si el navegador no se encuentra abierto, primero lo abre, y posteriormente carga la página web en una ventana nueva. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); driver.get("www.google.es");</pre>	

MÉTODO getCurrentUrl()

Return	Método(Parámetros)
java.lang.String	getCurrentUrl()
Descripción	
<ul style="list-style-type: none">- Método que devuelve la url que está mostrando el navegador en el momento de su llamada. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); driver.get("www.google.es"); System.out.println(driver.getCurrentUrl());</pre>	

MÉTODO getTitle()

Return	Método(Parámetros)
java.lang.String	getTitle()
Descripción	
<p>- Método que devuelve el título de la ventana que está mostrando el navegador en el momento de su llamada.</p> <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); driver.get("www.google.es"); System.out.println(driver.getTitle());</pre>	

MÉTODO getPageSource()

Return	Método(Parámetros)
java.lang.String	getPageSource()
Descripción	
<p>- Método que devuelve el código fuente HTML de la página web que está mostrando el navegador en el momento de su llamada.</p> <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); driver.get("www.google.es"); System.out.println(driver.getPageSource());</pre>	

MÉTODO close()

Return	Método(Parámetros)
void	Close()
Descripción	
<p>- Método que cierra la ventana activa que esté mostrando el navegador en el momento de su llamada. Si es la última ventana, también cierra el navegador.</p> <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); driver.get("www.google.es"); driver.close();</pre>	

En el siguiente ejemplo se utilizan los 5 métodos descritos anteriormente. El script abre una ventana del navegador Firefox y carga la página login_page.htm, que en este caso, se encuentra ubicada en un directorio local de nuestra máquina.

Ejemplo cap7.Ejemplo1.java

```
/*
 * EJEMPLO 1: Métodos básicos interfaz WebDriver
 * ACCIONES:
 * Cargar la pagina web
 * Mostrar por consola la url de la pagina cargada
 * Mostrar por consola el titulo de la ventana abierta
 * Mostrar por consola el codigo HTML de la pagina cargada
 */

WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");

System.out.println(driver.getCurrentUrl());
System.out.println(driver.getTitle());
System.out.println(driver.getPageSource());

Thread.sleep(5000);
driver.close();
```

Analizando más en detalle el código anterior, si nos fijamos en la declaración de la variable `driver`, veremos que gracias al polimorfismo, dicha variable puede contener cualquier objeto de una clase que implemente la interfaz `WebDriver`, de esta forma, si en lugar de querer utilizar el navegador Firefox quisiéramos ejecutar este mismo ejemplo sobre el navegador Safari, bastaría con sustituir la línea `WebDriver driver = new FirefoxDriver()` por `WebDriver driver = new SafariDriver()`;

En el ejemplo 1, el objeto `driver` carga una página que se encuentra en disco, pero si quisiéramos cargar cualquier otra bastaría con indicarle al método `get()` la URL deseada.

Por ejemplo, podríamos cargar la página <http://www.etsist.upm.es/>

```
WebDriver driver = new FirefoxDriver();
driver.get("https://www.etsist.upm.es/");

System.out.println(driver.getCurrentUrl());
System.out.println(driver.getTitle());
System.out.println(driver.getPageSource());

Thread.sleep(5000);
driver.close();
```


7.2 INTERFAZ WEBDRIVER. METODOS PARA LOCALIZAR ELEMENTOS EN UNA WEB

Después de cargar una determinada página web, el siguiente paso es poder localizar los diferentes elementos que la componen con el objetivo de analizarlos o actuar sobre ellos. Por ejemplo, en una aplicación web donde para acceder a un determinado servicio se nos ofrece un portal de inicio de sesión, seguramente nos interese localizar los campos donde poder introducir el usuario y la contraseña.

Cada elemento que compone una página web Selenium WebDriver lo representa mediante un objeto de tipo `WebElement`. Los cajetines de texto anteriormente mencionados, donde poder introducir el usuario y la contraseña, serán manejados a partir de su representación como objetos de este tipo.

La interfaz `WebDriver` define 2 métodos para la localización de elementos:

MÉTODO `findElement()`

Return	Método(Parámetros)
<code>WebElement</code>	<code>findElement(By by)</code>
Descripción	
<p>- Método que devuelve el primer elemento encontrado que cumpla con el tipo de búsqueda (mecanismo) definido por el parámetro <code>by</code>.</p> <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username"));</pre>	

MÉTODO `findElements()`

Return	Método(Parámetros)
<code>Java.util.List<WebElement></code>	<code>findElements(By by)</code>
Descripción	
<p>- Método que devuelve una lista con los elementos encontrados que cumplan con el tipo de búsqueda definida por el parámetro <code>by</code>.</p>	

Para ilustrar cómo se manejan los dos métodos anteriormente mencionados podemos analizar los siguientes ejemplos:

Ejemplo cap7.Ejemplo2.java

```
/*
 * EJEMPLO 2: Localización de WebElements
 * ACCIONES:
 * Cargar la página web
 * Localizar cajetín para introducir usuario
 * Teclear texto
 * Borrar texto
 * Teclear texto
 */

WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");

//Localizar cajetín para introducir el usuario
WebElement user = driver.findElement(By.name("username"));
Thread.sleep(1000);
//Escribir sobre el cajetín
user.sendKeys("Usuario");

Thread.sleep(2000);
//Borrar el contenido del cajetín
user.clear();
Thread.sleep(500);
//Volver a escribir en el cajetín
user.sendKeys("OtroUsuario");

Thread.sleep(2000);
driver.close();
```

La sentencia `driver.findElement(By.name("username"))` nos devuelve un objeto de tipo `WebElement` que representa, en este caso, al elemento HTML que recoge el “username” que debe introducir el usuario. Se trata del cajetín de texto cuyo atributo `name` coincide con “*username*”.

En este otro ejemplo, mediante el método `findElements()`, lo que obtenemos es una lista con todos los elementos de tipo checkbox:

Ejemplo cap7.Ejemplo3.java

```
/*
 * EJEMPLO 3: Localización de WebElements
 * ACCIONES:
 * Cargar la página web
 * Localizar todos los checkbox (crear lista)
 * Hacer click sobre ellos
 */

WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");
Thread.sleep(2000);

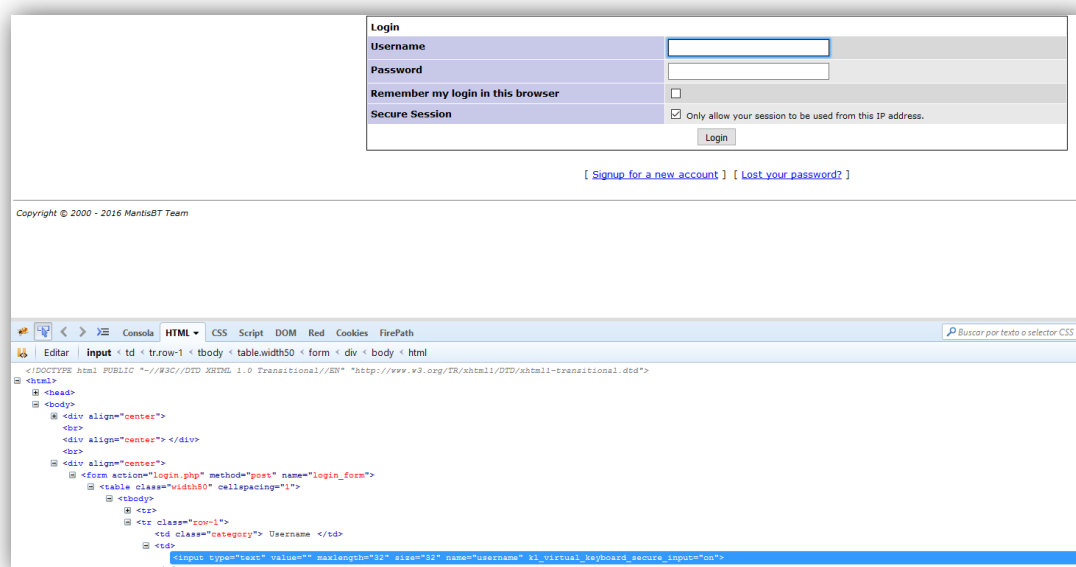
//Localizar todos los elementos de tipo checkbox
List<WebElement> checkboxList =
driver.findElements(By.xpath("html/body/div[3]/form/table/tbody/tr/td/input[@
type='checkbox']"));

Iterator<WebElement> it = checkboxList.iterator();
while(it.hasNext()){
    it.next().click();
    Thread.sleep(2000);
}

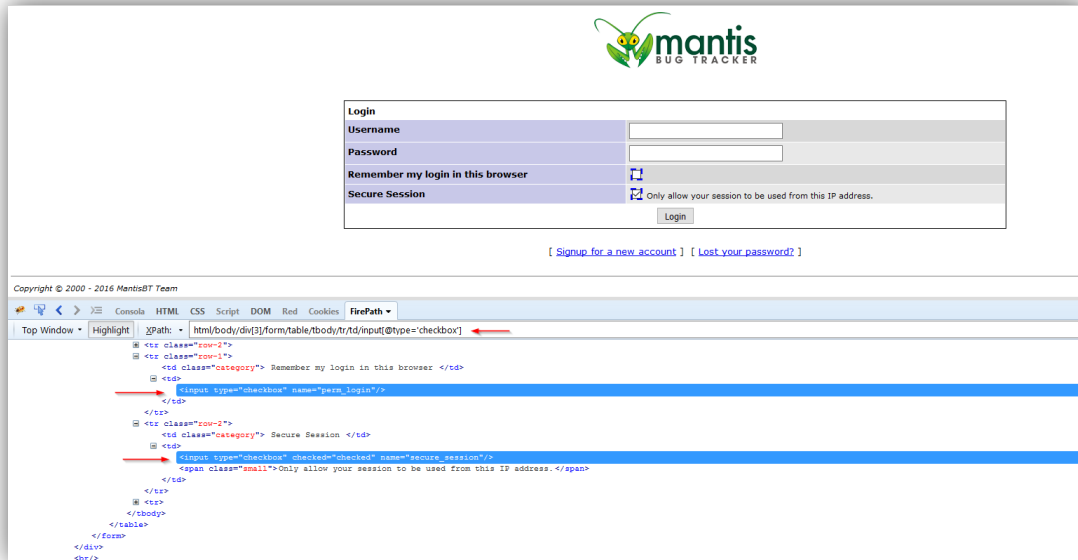
Thread.sleep(2000);
driver.close();
```

7.2.1 USO DE FIREBUG PARA LOCALIZAR ELEMENTOS EN EL CÓDIGO HTML

En los ejemplos anteriores para conocer la correspondencia entre los elementos visualizados por el navegador y el código HTML, nos hemos ayudado de los complementos de Firefox, FireBug y FirePath. Por ejemplo, el cajetín donde introducimos el usuario podemos localizarlo seleccionando la función de localización (ver apartado 6.3) y posicionando el puntero del ratón sobre dicho cajetín. En la parte inferior de la consola de FireBug se marca el código HTML asociado al elemento.



En el caso del Ejemplo 3 hemos utilizado la dirección XPath para poder seleccionar los elementos del tipo checkbox. Para ello, hemos pinchado sobre la opción FirePath de la barra de herramientas del complemento FireBug.



7.3. MECANISMOS DE LOCALIZACIÓN DE ELEMENTOS. LA CLASE By

Cómo hemos visto anteriormente, los métodos `findElement` y `findElements` tienen como parámetro de entrada un objeto de tipo `By` que determina el tipo de búsqueda a realizar.

La clase `By` posee 8 métodos estáticos que determinan el mecanismo a utilizar para localizar un elemento HTML dentro de la página web. Cada mecanismo es implementado por una clase anidada.

Clases anidadas de la clase `By` son:

- `By.ByClassName`
- `By.ByCssSelector`
- `By.ById`
- `By.ByLinkText`
- `By.ByName`
- `By.ByPartilaLinkText`
- `By.ByTagName`
- `By.ByXpath`

MÉTODO `By.className()`

Return	Método(Parámetros)
<code>By.ByClassName</code>	<code>className(java.lang.String className)</code>
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el atributo HTML "class".	

MÉTODO `By.name()`

Return	Método(Parámetros)
<code>By.ByName</code>	<code>name(java.lang.String name)</code>
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el atributo HTML "name".	

MÉTODO `By.tagName()`

Return	Método(Parámetros)
<code>By.ByTagName</code>	<code>tagName(java.lang.String tagName)</code>
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el nombre de la etiqueta HTML.	

MÉTODO By.id()

Return	Método(Parámetros)
By.ById	id(java.lang.String id)
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el atributo HTML "id".	

MÉTODO By.linkText()

Return	Método(Parámetros)
By.ByLinkText	linkText(java.lang.String linkText)
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el texto contenido por los elementos con atributo "href". Debe coincidir el texto "linkText" en su totalidad con el texto del atributo.	

MÉTODO By.partialLinkText()

Return	Método(Parámetros)
By.ByPartialLinkText	partialLinkText(java.lang.String partialLink)
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el texto contenido por los elementos con atributo "href". Se tiene que encontrar la subcadena "partialLink" dentro del texto del atributo.	

MÉTODO By.xpath()

Return	Método(Parámetros)
By.ByXPath	xpath(java.lang.String xpathExpression)
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en la expresión Xpath "xpathExpression".	

MÉTODO By.cssSelector()

Return	Método(Parámetros)
By.CssSelector	cssSelector(java.lang.String selector)
Descripción	
- Método que devuelve el mecanismo de búsqueda basado en el selector CSS "selector".	

El siguiente ejemplo ilustra el funcionamiento de los 8 métodos estáticos de la clase **By**:

Ejemplo cap7.Ejemplo4.java

```

/*
 * EJEMPLO 4: Localización de WebElements. Ejemplo 8 Mecanismos By
 * ACCIONES:
 * Localizar elemento ByClassName y añadir a la lista
 * Localizar elemento ByName y añadir a la lista
 * Localizar elemento ByTagName y añadir a la lista
 * Localizar elemento ById y añadir a la lista
 * Localizar elemento ByLinkText y añadir a la lista
 * Localizar elemento ByPartialLinkText y añadir a la lista
 * Localizar elemento ByXPath y añadir a la lista
 * Localizar elemento ByCssSelector y añadir a la lista
 * Recorrer la lista de elementos imprimiendo informacion de cada uno de
 * ellos
 */

WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");

List<WebElement> listaElementos = new ArrayList<WebElement>();
//Localizar elemento ByClassName
listaElementos.add(driver.findElement(By.className("button")));

//Localizar elemento ByName
listaElementos.add(driver.findElement(By.name("username")));

//Localizar elemento ByTagName
listaElementos.add(driver.findElement(By.tagName("td")));

//Localizar elemento ById
listaElementos.add(driver.findElement(By.id("session")));

//Localizar elemento ByLinkText
listaElementos.add(driver.findElement(By.linkText("Signup for a new
account")));

//Localizar elemento ByPartialLinkText
listaElementos.add(driver.findElement(By.partialLinkText("Lost")));

//Localizar elemento ByXPath
listaElementos.add(driver.findElement(By.xpath("html/body/div[3]/form/table/t
body/tr[4]/td[1]")));

//Localizar elemento ByCssSelector
listaElementos.add(driver.findElement(By.cssSelector("body > table")));

Iterator<WebElement> it = listaElementos.iterator();
WebElement e = null;
while(it.hasNext()){
    e = it.next();
    System.out.println(e.toString());
    System.out.println("Tag: " + e.getTagName() + "   Position: " +
e.getLocation().toString());
    System.out.println("Text: " + e.getText() + "\n");
}
Thread.sleep(2000);
driver.close();

```

El Ejemplo 4 abre una ventana del navegador Firefox, carga la página login_page.htm y localiza 8 elementos de dicha página, cada uno de ellos mediante un mecanismo diferente. Los WebElements seleccionados se van metiendo en una lista, para posteriormente poder operar sobre ellos. Concretamente, se obtiene su representación como cadena de caracteres, el tag HTML del elemento, la posición que tiene en pantalla y el texto que contiene, si es que contiene alguno.

Por último, indicar varias referencias web donde encontrar información sobre el funcionamiento y uso de XPath y CSS Selector:

- <https://www.w3.org/TR/xpath/>
- http://www.w3schools.com/xsl/xpath_intro.asp
- <http://www.w3schools.com/css/default.asp>
- http://www.w3schools.com/cssref/css_selectors.asp

7.4 LA CLASE WEBeLEMENT. ACCIONES SOBRE LOS ELEMENTOS DE UNA WEB

En el apartado anterior hemos visto cómo localizar elementos dentro de una página web (WebElemets), ahora veremos qué acciones podemos realizar sobre ellos.

MÉTODO WebElement.getAttribute()

Una de las acciones que podemos realizar sobre un WebElement determinado, es obtener el contenido de uno de sus atributos.

Return	Método(Parámetros)
java.lang.String	getAttribute(java.lang.String nameAttribute)
Descripción	
<ul style="list-style-type: none"> - Mediante este método se puede obtener el valor del atributo "nameAttribute" de un elemento web determinado. El valor del atributo es devuelto como un objeto de tipo String. - Si no encuentra el atributo mencionado el método devuelve null. <p>NOTA: Este método puede ser aplicado a cualquier tipo de elemeto web.</p> <p>Ejemplo suponiendo un código HTML:</p> <pre>----- <input type="text" maxlength="32" size="32" name="username"> -----</pre> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); String type = user.getAttribute("type"); System.out.print(type);</pre>	

MÉTODO WebElement.sendKeys ()

La acción de poder introducir texto en un elemento de tipo “caja de texto” o de tipo “área de texto” se puede realizar mediante el método sendKeys().

Return	Método(Parámetros)
void	sendKeys(java.lang.String keysToSend)
Descripción	
<ul style="list-style-type: none">- Método que simula la introducción de texto por parte del usuario en elementos de tipo caja de texto o área de texto.- Si el elemento en cuestión no soporta la introducción de texto, el método no produce acción alguna. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); user.sendKeys("Miguel");</pre> <p>NOTA: Para simular la pulsación de ciertas teclas especiales es necesario ayudarse de la clase Keys que proporciona las funcionalidades necesarias para emular el teclado de un PC. Para más información, consultar el recurso web http://seleniumhq.github.io/selenium/docs/api/java/</p> <p>Ejemplo:</p> <pre>user.sendKeys(Keys.chord(Keys.SHIFT, "Miguel"));</pre>	

MÉTODO WebElement.clear()

Si con el anterior método, sendKeys(), se podía simular la introducción de texto sobre un elemento web, con el método clear() se puede simular la acción de borrarlo.

Return	Método(Parámetros)
void	clear()
Descripción	
<ul style="list-style-type: none">- Método que borra el texto introducido en elementos de tipo caja de texto o área de texto.- Si el elemento en cuestión no soporta la introducción/borrado de texto, el método no produce acción alguna. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); user.sendKeys("Miguel"); user.clear();</pre>	

MÉTODO WebElement.submit()

Cuando el elemento web en cuestión es un formulario, o está dentro de un formulario, el método submit() valida el formulario y lo envía al servidor.

Return	Método(Parámetros)
void	submit()
Descripción	
<ul style="list-style-type: none"> - Método para validar y enviar un elemento de tipo formulario o perteneciente a un formulario. - Si el elemento seleccionado no es o forma parte de un formulario, el método submit() lanza una excepción NoSuchElementException. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); user.sendKeys("Miguel"); user.submit();</pre>	

MÉTODO WebElement.getCssValue()

Para obtener las diferentes propiedades CSS relacionadas con un elemento web utilizamos el método getCssValue().

Return	Método(Parámetros)
java.lang.String	getCssValue(java.lang.String propertyName)
Descripción	
<ul style="list-style-type: none"> - Método para obtener las propiedades CSS de un determinado elemento web. - La propiedad buscada se le pasa como parámetro y el valor de la misma es devuelto como valor de retorno del método. - Metodo aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); System.out.println(user.getCssValue("font-family"));</pre>	

MÉTODO WebElement.getLocation()

Para poder conocer en qué posición de la pantalla está mostrando el navegador un determinado elemento web utilizaremos el método getLocation().

Return	Método(Parámetros)
org.openqa.selenium.Point	getLocation()
Descripción	
<ul style="list-style-type: none">- Método para obtener la posición en pantalla de un elemento web.- El método devuelve un objeto de tipo Point, que representa una tupla de coordenadas (x,y). Indica la posición en píxeles tomando como punto de referencia la esquina superior izquierda de la ventana del navegador.- Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); System.out.println(user.getLocation());</pre>	

MÉTODO WebElement.getSize()

Para poder conocer el tamaño que ocupa un determinado elemento web, la clase WebElement cuenta con el método getSize()

Return	Método(Parámetros)
org.openqa.selenium.Dimension	getSize()
Descripción	
<ul style="list-style-type: none">- Método para obtener la dimensión en pantalla ocupada por un elemento web.- El método devuelve un objeto de tipo Dimension, que representa el tamaño en píxeles que ocupa el elemento web.- El tamaño se representa por medio de una tupla (x, y) que indica el número de píxeles horizontales(x) y el número de píxeles verticales (y) ocupados.- Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement user = driver.findElement(By.name("username")); System.out.println(user.getSize());</pre>	

MÉTODO WebElement.isSelected()

Cuando el elemento web sobre el que estamos trabajando es de tipo checkbox o similar, seguramente nos interese saber si encuentra seleccionado o no. Para este tipo de acciones utilizaremos el método isSelected() que devuelve True o False dependiendo de si lo está o no.

Return	Método(Parámetros)
boolean	isSelected()
Descripción	
<ul style="list-style-type: none"> - Método para conocer si un elemento de tipo checkbox o similar se encuentra seleccionado. - Devuelve True o False, en función de si se encuentra seleccionado o no el elemento en el momento de realizar la llamada al método. - Método aplicable a cualquier elemento web, lo que ocurre es que en algunos casos siempre se volverá False debido a que el elemento web en cuestión NO es seleccionable. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement checkboxList = driver.findElement(By.xpath("input[@type='checkbox']")); System.out.println(user.isSelected())</pre>	

MÉTODO WebElement.isDisplayed()

Para conocer si un determinado elemento web está siendo mostrado, podemos hacer uso del método isDisplayed()

Return	Método(Parámetros)
boolean	isDisplayed()
Descripción	
<ul style="list-style-type: none"> - Método para conocer si un elemento está siendo mostrado por el navegador. - Devuelve True o False, en función de si se muestra o no el elemento en cuestión en el momento de realizar la llamada al método. - Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement button = driver.findElement(By.className("button")); System.out.println(button.isDisplayed())</pre>	

MÉTODO WebElement.getText()

En ocasiones, nos interesará comprobar el texto que está mostrando un determinado elemento web. Para ello WebElement ofrece el método getText()

Return	Método(Parámetros)
java.lang.String	getText()
Descripción	
<ul style="list-style-type: none">- Método para obtener el texto mostrado o contenido por un elemento.- Devuelve una cadena de caracteres. Si el elemento no contiene texto, la cadena devuelta será vacía.- Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement e = driver.findElement(By.tagName("address")); System.out.println(e.getText());</pre>	

MÉTODO WebElement.getTagName()

Cuando necesitamos conocer la etiqueta HTML correspondiente al elemento web seleccionado utilizaremos el método getTagName().

Return	Método(Parámetros)
java.lang.String	getTagName()
Descripción	
<ul style="list-style-type: none">- Método para obtener el nombre de la etiqueta HTML del elemento web seleccionado.- Devuelve una cadena de caracteres.- Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement button = driver.findElement(By.className("button")); System.out.println(button.getTagName());</pre>	

MÉTODO WebElement.isEnabled()

Para conocer si un determinado elemento está habilitado o no, utilizaremos el método isEnabled().

Return	Método(Parámetros)
boolean	isEnabled()
Descripción	
<ul style="list-style-type: none"> - Método para conocer si un elemento esta habilitado. Devuelve True o False en función de si lo está o no. - Método aplicable a cualquier elemento web. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement button = driver.findElement(By.className("button")); System.out.println(button.isEnabled())</pre>	

MÉTODO WebElement.click()

Cuando necesitemos hacer “clic” sobre un determinado elemento utilizaremos el método click().

Return	Método(Parámetros)
void	click()
Descripción	
<ul style="list-style-type: none"> - Método para realizar clic sobre el elemento web seleccionado. - Hay que tener en cuenta que si ésta acción tienen como consecuencia un cambio de página, todas las referencias a elementos web pueden cambiar. <p>Ejemplo:</p> <pre>WebDriver driver = new FirefoxDriver(); ... WebElement button = driver.findElement(By.className("button")); button.click()</pre>	

CAPÍTULO 8

LA CLASE ACTIONS. INTERACCIONES AVANZADAS CON EL NAVEGADOR

En este capítulo se estudiará la parte del API WebDriver que permite simular la interacción que realiza el usuario con la aplicación web mediante el uso del teclado y el ratón.

Por norma general, las acciones que solemos realizar durante la navegación son:

- Clic/Doble clic con el botón izquierdo sobre un área o elemento web.
- Clic con el botón derecho del ratón sobre un área o elemento web.
- Pinchar y arrastrar un elemento web de una posición a otra de la pantalla.
- Introducir texto en un elemento web.
- Mantener presionada una tecla del teclado mientras completamos alguna otra acción y luego liberar dicha tecla al finalizar.
- Pulsar teclas especiales de control cómo pueden ser el Shift, Ctrl, F5, etc.

En los siguientes apartados estudiaremos los mecanismos y métodos que nos ofrece la clase Actions para poder simular todas estas interacciones. Para ello se diferenciará entre las operaciones realizadas desde el ratón y las realizadas desde el teclado.

8.1 INTERACCIÓN REALIZADA CON EL RATÓN

La clase Actions nos va a permitir describir una serie de acciones que posteriormente vamos a poder realizar de manera secuencial (una detrás de la otra, siguiendo el orden de aparición). En este apartado concretamente, estudiaremos los métodos para poder interactuar con nuestra aplicación web mediante el uso del ratón.

8.1.1 MÉTODO moveByOffset()

Return	Método(Parámetros)
Actions	moveByOffset(int xOffset, int yOffset)
Descripción	
<ul style="list-style-type: none">- xOffset : num de pixeles de desplazamiento horizontal. Valor positivo desplazamiento a la derecha, valor negativo desplazamiento a la izquierda.- yOffset : num de pixeles de desplazamiento vertical. Valor positivo desplazamiento hacia abajo, valor negativo desplazamiento hacia arriba.- Este método desplaza el cursor del ratón desde su posición actual "xOffset" pixeles a la derecha e "yOffset" pixeles hacia abajo.- Punto de partida del cursor: (0, 0) -> origen de coordenadas. El origen de coordenadas es la esquina superior izquierda.	
Ejemplo cap8.MoveByOffset1.java	
<pre>WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement checkbox = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[4]/td[2]/input")); Thread.sleep(2000); Actions clickEncursor = new Actions(driver); clickEncursor .moveByOffset(checkbox.getLocation().getX() + 4, checkbox.getLocation().getY() + 4) .click(); clickEncursor.perform(); Thread.sleep(2000); driver.close();</pre>	

8.1.2 MÉTODO moveToElement()

Return	Método(Parámetros)
Actions	moveByOffset(WebElement element)
Descripción	
<p>- element: WebElemeto sobre el cual el método coloca el cursor del ratón.</p> <p>- Este método desplaza el cursor del ratón desde su posición actual hasta situarlo sobre el elemento "element".</p>	
Ejemplo cap8.MoveToElement.java	
<pre> WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement checkbox1 = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[4]/td[2]/input")); WebElement checkbox2 = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[5]/td[2]/input")); Thread.sleep(2000); Actions clickEncursor1 = new Actions(driver); clickEncursor1 .moveToElement(checkboxbox1) .click(); clickEncursor1.perform(); Thread.sleep(2000); Actions clickEncursor2 = new Actions(driver); clickEncursor2 .moveToElement(checkboxbox2) .click(); clickEncursor2.perform(); Thread.sleep(2000); driver.close(); </pre>	

8.1.3 MÉTODO click()

Return	Método(Parámetros)
Actions	click() click(WebElement onElement)
Descripción	
<ul style="list-style-type: none">- onElement: WebElement sobre el cual se realiza el click del ratón.- Este método simula la acción que realiza el usuario con el ratón al hacer click con el botón izquierdo.- Si no se le pasa un WebElement como parámetro, el click se realiza donde se encuentre situado el cursor en el momento de realizar la llamada a este método.- La segunda opción consiste en pasarle como parámetro el elemento sobre cual queremos realizar click.	
Ejemplo cap8.Click.java <pre>WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement checkbox1 = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[4]/td[2]/input")); WebElement checkbox2 = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[5]/td[2]/input")); Thread.sleep(2000); Actions clickEncursor1 = new Actions(driver); clickEncursor1 .moveToElement(checkboxbox1) .click(); clickEncursor1.perform(); Thread.sleep(2000); Actions clickEnElemento = new Actions(driver); clickEnElemento .click(checkboxbox2); clickEnElemento.perform(); Thread.sleep(2000); driver.close();</pre>	

8.1.4 MÉTODO doubleClick()

Return	Método(Parámetros)
Actions	doubleClick() doubleclick(WebElement onElement)
Descripción	
<ul style="list-style-type: none"> - onElement: WebElement sobre el cual se realiza el doble click del ratón. - Este método simula la acción que realiza el usuario con el ratón al hacer doble click con el botón izquierdo del mismo. - Si no se le pasa un WebElement como parámetro, el doble click se realiza donde se encuentre situado el cursor en el momento de realizar la llamada a este método. - La segunda opción consiste en pasarle como parámetro el elemento sobre cual realizar la acción. 	
Ejemplo cap8.DoubleClick.java	
<pre> WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement username = driver.findElement(By.xpath("html/body/div[3]/form/table/tbody/tr[2]/td[1]")); ; WebElement password = driver.findElement(By.xpath("html/body/div[3]/form/table/tbody/tr[3]/td[1]")); ; Actions ejemploDobleClick1 = new Actions(driver); ejemploDobleClick1 .doubleClick(username); ejemploDobleClick1.perform(); Actions ejemploDobleClick2 = new Actions(driver); ejemploDobleClick2 .moveToElement(password) .doubleClick(); ejemploDobleClick2.perform(); </pre>	

8.1.5 MÉTODO contextClick()

Return	Método(Parámetros)
Actions	contextClick() contextClick(WebElement onElement)
Descripción	
<ul style="list-style-type: none">- onElement: WebElement sobre el cual se realiza el click con el botón derecho del ratón.- Este método simula la acción que realiza el usuario con el ratón al hacer click con el botón derecho del mismo.- Si no se le pasa un WebElement como parámetro, el click se realiza donde se encuentre situado el cursor en el momento de realizar la llamada a este método.- La segunda opción consiste en pasarle como parámetro el elemento sobre cual queremos realizar click.	
Ejemplo cap8.ContextClick.java <pre>WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement element1 = driver.findElement(By.xpath("html/body/div[1]/a/img")); WebElement element2 = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[3]/td[2]/input")); Thread.sleep(2000); Actions clickEncursor1 = new Actions(driver); clickEncursor1 .moveToElement(element1) .contextClick(); clickEncursor1.perform(); Thread.sleep(2000); Actions clickEnElemento = new Actions(driver); clickEnElemento .sendKeys(Keys.ESCAPE) .contextClick(element2); clickEnElemento.perform(); Thread.sleep(2000); driver.close();</pre>	

8.1.6 MÉTODO clickAndHold()

Return	Método(Parámetros)
Actions	clickAndHold() clickAndHold(WebElement onElement)
Descripción	
<ul style="list-style-type: none"> - onElement: WebElement sobre el cual se realiza el click. - Este método simula la acción de realizar click sobre un elemento web manteniendo pulsado el botón izquierdo del ratón. - Al igual que en los ejemplos anteriores podemos usar este método de dos maneras, realizando el click en la posición donde se encuentre el cursor del ratón en ese momento o especificar sobre qué elemento hacerlo. 	
Ejemplo cap8.ClickAndHold.java	
<pre> WebElement testCase = driver.findElement(By.xpath("./*[@id='extdd-11']")); WebElement testSuit = driver.findElement(By.xpath("./*[@id='extdd-8']")); Actions moverTestCase = new Actions(driver); moverTestCase .clickAndHold(testCase) .moveToElement(testSuit) .release(); moverTestCase.perform(); </pre>	

8.1.7 MÉTODO release()

Return	Método(Parámetros)
Actions	release() release(WebElement onElement)
Descripción	
<ul style="list-style-type: none"> - onElement: WebElement que se libera. - Este método simula la acción de soltar el botón izquierdo del ratón. - Al igual que en los ejemplos anteriores podemos usar este método de dos maneras, realizando la acción de liberar el boton izquierdo del raton en la posición donde se encuentre en ese momento el cursor o liberarlo en la posición de un determinado elemento. 	
Ejemplo cap8.Release.java	
<pre> WebElement testCase = driver.findElement(By.xpath("./*[@id='extdd-11']")); WebElement testSuit = driver.findElement(By.xpath("./*[@id='extdd-8']")); Actions moverTestCase = new Actions(driver); moverTestCase .clickAndHold(testCase) .release(testSuit); moverTestCase.perform(); </pre>	

8.1.8 MÉTODO dragAndDropBy()

Return	Método(Parámetros)
Actions	dragAndDropBy(WebElement source, int xOffset, int yOffset)
Descripción	
<ul style="list-style-type: none">- source: WebElement que se desea arrastrar.- xOffset: Desplazamiento lateral.- yOffset: Desplazamiento vertical. <p>- Este método simula la acción de arrastrar un elemento web de una posición a otra de la pantalla.</p>	
Ejemplo cap8.DragAndDropBy.java	
<pre>WebElement testCase = driver.findElement(By.xpath(".*[@id='extdd-11']")); WebElement testSuit = driver.findElement(By.xpath(".*[@id='extdd-8']")); int xOffset = testSuit.getLocation().x - testCase.getLocation().x; int yOffset = testSuit.getLocation().y - testCase.getLocation().y; Actions moverTestCase = new Actions(driver); moverTestCase .dragAndDropBy(testCase, xOffset, yOffset); moverTestCase.perform();</pre>	

8.1.9 MÉTODO dragAndDrop()

Return	Método(Parámetros)
Actions	dragAndDrop(WebElement source, WebElement target)
Descripción	
<ul style="list-style-type: none">- source: WebElement que se desea arrastrar.- target: WebElement hasta el cual se arrastra el elemento web "source". <p>- Este método simula la acción de arrastrar el elemento web "source" desde su posición a la posición del elemento "target"</p>	
Ejemplo cap8.DragAndDrop.java	
<pre>WebElement testCase = driver.findElement(By.xpath(".*[@id='extdd-11']")); WebElement testSuit = driver.findElement(By.xpath(".*[@id='extdd-8']")); Actions moverTestCase = new Actions(driver); moverTestCase .dragAndDrop(testCase, testSuit); moverTestCase.perform();</pre>	

8.2 INTERACCIÓN REALIZADA CON EL TECLADO

En el apartado anterior realizamos un estudio de los 9 métodos que ofrece Selenium para simular la interacción que realiza el usuario sobre el navegador web haciendo uso del ratón. A continuación haremos lo propio con las acciones realizadas desde el teclado.

8.2.1 MÉTODO keyDown()

Return	Método(Parámetros)
Actions	keyDown(Keys theKey) keyDown(WebElement onElement, Keys theKey)
Descripción	
<ul style="list-style-type: none"> - onElement: Lugar donde se realizará la acción de dejar pulsada la tecla especificada. - theKey: Tecla que se deja presionada. - Este método simula la acción mantener pulsada una tecla. Para realizar la acción contraria, soltar la tecla, se hace uso del método keyUp que se estudiará en el apartado siguiente. - Al igual que métodos anteriores, puede aplicarse sobre un elemento en concreto o donde se encuentre en ese momento el cursor. 	
Ejemplo cap8.KeyDown.java	
<pre> WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement username = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[2]/td[2]/input")); Thread.sleep(2000); Actions escribirMayus = new Actions(driver); escribirMayus .keyDown(Keys.SHIFT) .sendKeys(username, "usuario"); escribirMayus.perform(); Thread.sleep(2000); driver.close(); </pre>	

8.2.2 MÉTODO keyUp()

Return	Método(Parámetros)
Actions	keyUp(Keys theKey) keyUp(WebElement onElement, Keys theKey)
Descripción	
<ul style="list-style-type: none">- onElement: Lugar donde se realizará la acción de levantar la tecla especificada.- theKey: Tecla que se deja de presionar. <p>- Este método simula la acción de levantar o dejar de pulsar una determinada tecla. Es la acción contraria vista en el punto 8.2.1</p> <p>- Al igual que métodos anteriores, puede aplicarse sobre un elemento en concreto o donde se encuentre en ese momento el cursor.</p>	
Ejemplo cap8.KeyUp.java	
<pre>Actions escribirMayus = new Actions(driver); escribirMayus .keyDown(Keys.SHIFT) .sendKeys(username, "usuario"); escribirMayus.perform(); Thread.sleep(2000); Actions escribirMinus = new Actions(driver); escribirMinus .keyUp(Keys.SHIFT) .sendKeys(username, "usuario"); escribirMinus.perform(); driver.close();</pre>	

8.2.3 MÉTODO sendKeys()

Return	Método(Parámetros)
Actions	sendKeys(String keysToSend) sendKeys (WebElement onElement, String keysToSend)
Descripción	
<ul style="list-style-type: none"> - onElement: Elemento al que se le envía la cadena de texto tecleada. - keysToSend: Cadena de texto tecleada. <p>- Este método simula la acción de teclear una cadena de texto en un elemento web. Para que tenga efecto, el elemento web debe permitir la entrada de texto.</p> <p>- Al igual que métodos anteriores, puede aplicarse sobre un elemento en concreto o donde se encuentre en ese momento el cursor.</p>	
Ejemplo cap8.SendKeys.java	
<pre> WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); WebElement username = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[2]/td[2]/input")); WebElement password = driver.findElement (By.xpath("html/body/div[3]/form/table/tbody/tr[3]/td[2]/input")); Thread.sleep(2000); Actions escribir = new Actions(driver); escribir .moveToElement(username) .sendKeys("usuario"); escribir.perform(); Thread.sleep(2000); escribir .moveToElement(password) .sendKeys(password, "password"); escribir.perform(); Thread.sleep(2000); driver.close(); </pre>	

8.3. LA CLASE KEYS

Cuando utilizamos el teclado de nuestro PC existen un gran número de teclas con funciones determinadas que no se corresponden con caracteres alfanuméricos. Por ejemplo, cualquiera de las teclas de función F1, F2, F3, la tecla Esc, Ctrl, Shift, Alt Gr, etc.

Estas teclas pueden tener ciertas funcionalidades durante la navegación web. Para poder hacer referencia a ellas desde los métodos anteriormente descritos (keyDown, keyUp y sendKeys) Selenium WebDriver define la clase Keys.

Esta clase representa una enumeración de las teclas de función que pueden encontrarse en un teclado de PC estándar:

- Teclas de función: F1, F2,...F12.
- Esc, Enter, Return, Tab, barra de espacio, Shift, Ctrl, Alt, Insert, Delete, etc.
- Flechas: Arrow left, Arrow up, Arrow right y Arrow down.

Clase Keys

A continuación se presentan unos ejemplos sobre el uso de la clase Keys para simular la utilización, por parte del usuario, de las teclas especiales del teclado estándar.

Ejemplo cap8.EjemploKeys.java

```
WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");
WebElement username = driver.findElement
(By.xpath("html/body/div[3]/form/table/tbody/tr[2]/td[2]/input"));
Actions ejemploKeys = new Actions(driver);
ejemploKeys
    .moveToElement(username)
    .sendKeys(Keys.TAB)
    .sendKeys("password");
ejemploKeys.perform();

Thread.sleep(2000);
ejemploKeys.sendKeys(Keys.F5);
ejemploKeys.perform();

Thread.sleep(2000);
ejemploKeys.sendKeys("Usuario");
ejemploKeys.perform();

Thread.sleep(2000);
ejemploKeys
    .sendKeys(Keys.ARROW_LEFT)
    .sendKeys(Keys.DELETE);
ejemploKeys.perform();

Thread.sleep(2000);
ejemploKeys
    .sendKeys(Keys.HOME)
    .sendKeys(Keys.DELETE);
ejemploKeys.perform();
Thread.sleep(2000);
driver.close();
```

CAPÍTULO 9

WEBDRIVER. INTERFACES ANIDADAS

En el capítulo 8 hemos estudiado en profundidad los diferentes métodos que nos ofrecía la clase Actions para poder simular la interacción del usuario con la aplicación web. A continuación, iremos viendo operaciones que nos pueden ser de ayuda a la hora de crear nuestros tests, ya que recogen procesos frecuentes que se realizan durante la navegación por una página web.

Concretamente en este capítulo estudiaremos:

- Operaciones de navegación. Actualizar la página web, ir a la URL anterior, a la URL siguiente, etc.
- Gestión del tiempo de espera para la carga de los elementos web.
- Gestión del tamaño de la ventana del navegador.
- Gestión de Frames. Como poder pasar de un marco a otro.

9.1 OPERACIONES DE NAVEGACIÓN. INTERFAZ WEBDRIVER.NAVIGATION

La interfaz WebDriver encierra a su vez una serie de interfaces (interfaces anidadas) para poder manejar ciertas características relacionadas con el navegador y con el proceso de navegación realizado por el usuario.

Una de ellas es la interfaz Navigation, que define una serie de métodos para poder realizar las operación de ir adelante, atrás, actualizar la página o navegar a una URL determinada.

Métodos Interfaz WebDriver.Navigation

```
public static interface WebDriver.Navigation
    • back()
    • forward()
    • refresh()
    • to()
```

MÉTODO back()

Return	Método(Parámetros)
void	back()
Descripción	
- Este método carga la página anterior. Equivale a la acción que realiza el usuario cuando hace click en la flecha del navegador que hace referencia a la página anterior.	

MÉTODO forward()

Return	Método(Parámetros)
void	forward()
Descripción	
- Este método carga la página siguiente. Equivale a la acción que realiza el usuario cuando hace click en la flecha del navegador que hace referencia a la página siguiente.	

MÉTODO refresh()

Return	Método(Parámetros)
void	refresh()
Descripción	
- Este método vuelve a cargar la página actual. Equivale a la acción que realiza el usuario cuando hace click en la flecha o botón del navegador que hace referencia a la actualización de la página.	

MÉTODO to()

Return	Método(Parámetros)
void	to(java.lang.String url)
Descripción	
- Método que carga la página web especificada por su parámetro de entrada	

El método `navigate()` devuelve un objeto que implementa la interfaz `Navigation`. Veamos a continuación un ejemplo:

Ejemplo cap9.Navigation.java

```

/*
 * EJEMPLO: interfaz WebDriver.Navigation
 * ACCIONES
 * Ir a la URL http://www.google.es
 * Ir a la URL http://localhost:8080/testlink-1.9.11/login.php
 * Logear al usuario "admin" en la aplicaon de TestLink
 * Hacer click "Test Specification" de la barra de herraminetas superior
 * Ir a la pagina anterior
 * Ir a la pagina siguiente
 * Cerrar el navegador
 */

ProfilesIni profile = new ProfilesIni();
FirefoxProfile p = profile.getProfile("default");

FirefoxBinary binary = new FirefoxBinary(new File("C:\\Program
Files\\Mozilla Firefox\\firefox.exe"));
FirefoxDriver driver = new FirefoxDriver(binary,p);

driver.navigate().to("http://www.google.es");
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
//Login
Thread.sleep(3000);
driver.navigate().to("http://localhost:8080/testlink-1.9.11/login.php");

WebElement user = driver.findElement(By.xpath(".*[@id='login']"));
WebElement password =
driver.findElement(By.xpath(".*[@id='login_div']/form/p[2]/input"));
WebElement iniciarSesion =
driver.findElement(By.xpath(".*[@id='login_div']/form/input[5]"));

user.sendKeys("admin");
Thread.sleep(500);
password.sendKeys("1234");
iniciarSesion.click();

Thread.sleep(3000);
driver.switchTo().frame(driver.findElement(By.name("titlebar")));
WebElement e = driver.findElement(By.xpath("html/body/div[3]/a[2]"));
e.click();

driver.switchTo().defaultContent();
Thread.sleep(3000);
driver.switchTo().frame(driver.findElement(By.name("mainframe")));
driver.switchTo().frame(driver.findElement(By.name("treeframe")));
Thread.sleep(3000);

driver.navigate().back();
Thread.sleep(3000);
driver.navigate().forward();
Thread.sleep(3000);

driver.close();

```

9.2 GESTIÓN DE TIEMPOS DE ESPERA. INTERFAZ WEBDRIVER.TIMEOUTS

Un aspecto importante a tener en cuenta durante la creación de los tests automáticos con Selenium es el tiempo de carga de la página o de los elementos web. Si no se tienen en cuenta estos tiempos, y no se ponen las protecciones adecuadas frente a ellos, es posible que nuestros tests fallen debido a un error en la programación de la prueba.

Como hemos estudiado en capítulos anteriores, los métodos para la localización de elementos web lanzan una excepción cuando no encuentran ningún WebElement que coincida con el patrón o mecanismo de búsqueda. Para evitar que esta situación se produzca por el hecho de que no se haya cargado el elemento web en el momento en el que el script intenta localizarlo, Selenium ofrece unos mecanismos de espera a través de la interfaz WebDriver.Timeouts.

Métodos Interfaz WebDriver.Timeouts

```
public static interface WebDriver.Timeouts
{
    • implicitlyWait()
    • pageLoadTimeout()
    • setScriptTimeout()
```

MÉTODO implicitlyWait()

Return	Método(Parámetros)
WebDriver.Timeouts	implicitlyWait(long time, java.util.concurrent.TimeUnit unit)
Descripción	
<ul style="list-style-type: none">- Define el tiempo maximo de espera para la carga de cualquier WebElement.- Si vence el timeout se lanza la excepción: org.openqa.selenium.TimeoutException- El metodo devuelve una referencia al propio objeto.	

MÉTODO pageLoadTimeout()

Return	Método(Parámetros)
WebDriver.Timeouts	pageLoadTimeout(long time, java.util.concurrent.TimeUnit unit)
Descripción	
<ul style="list-style-type: none">- Define el tiempo maximo de espera para la carga completa de la página web.- Si vence el timeout se lanza la excepción: org.openqa.selenium.TimeoutException- El metodo devuelve una referencia al propio objeto.	

MÉTODO setScriptTimeout()

Return	Método(Parámetros)
WebDriver.Timeouts	setScriptTimeout(long time, java.util.concurrent.TimeUnit unit)
Descripción	
<ul style="list-style-type: none"> - Define el tiempo maximo de espera para que la ejecucion asincrona de un programa javascript finalice. - Si vence el timeout se lanza la excepción: org.openqa.selenium.TimeoutException - El metodo devuelve una referencia al propio objeto. 	

A través del método `manage().timeouts()` accedemos a los métodos anteriormente descritos.

Ejemplo cap9.Timeouts.java

```

/*
 * EJEMPLO: Interfaz WebDriver.Timeouts
 * ACCIONES
 * Al no poder cargar la pagina en 1 milisegundo, el metodo pageLoadTimeout
 * lanza la excepcion TimeoutException
 */

WebDriver driver = new FirefoxDriver();
try{
    driver.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);
    driver.manage().timeouts().pageLoadTimeout(1, TimeUnit.MILLISECONDS);
    ...

    Actions clickEncursor1 = new Actions(driver);
    clickEncursor1
        .moveToElement(checkbox1)
        .click();
    clickEncursor1.perform();

    Thread.sleep(2000);
    Actions clickEnElemento = new Actions(driver);
    clickEnElemento
        .click(checkbox2);
    clickEnElemento.perform();

    Thread.sleep(2000);
}catch(org.openqa.selenium.TimeoutException e){
    System.out.println(e);
}finally{
    driver.close();
}

```

9.3 CARACTERISTICAS DE LA VENTANA. INTERFAZ WEBDRIVER.WINDOW

Otra característica importante que se puede manejar durante la ejecución de los test automáticos es el tamaño de la ventana de la aplicación web. El tamaño de la ventana influye en la forma en la cual se representa la información y por tanto, en la forma en la que aparecen los elementos web.

Selenium ofrece a través de los métodos de la interfaz WebDriver.Window mecanismos para conocer y modificar la posición y el tamaño de la ventana de navegación.

Métodos Interfaz WebDriver.Window

```
public static interface WebDriver.Window
    • fullscreen()
    • getPosition()
    • getSize()
    • maximize()
    • setPosition()
    • setSize()
```

MÉTODO fullscreen()

Return	Método(Parámetros)
void	Fullscreen()
Descripción	
- Amplía la ventana del navegador de forma que ocupe toda la pantalla.	

MÉTODO getPosition()

Return	Método(Parámetros)
Point	getPosition()
Descripción	
- Este método devuelve la posición de la ventana actual del navegador. Utiliza como referencia la esquina superior izquierda.	

MÉTODO getSize()

Return	Método(Parámetros)
Dimension	getSize()
Descripción	
- Este método devuelve un objeto del tipo Dimension indicando el tamaño de la ventana actual del navegador. - EL objeto Dimension esta compuesto por atributos, anchura y altura que representan la anchura y la altura de la ventana medida en pixeles.	

MÉTODO maximize()

Return	Método(Parámetros)
void	maximize()
Descripción	
- Este método maximiza la ventan actual del navegador.	

MÉTODO setPosition()

Return	Método(Parámetros)
void	setPosition(Point targetPosition)
Descripción	
- Este método coloca la ventana actual del navegador en la posición indicada por su parámetro "targetPosition".	

MÉTODO setSize()

Return	Método(Parámetros)
void	setSize(Dimension targetSize)
Descripción	
- Este método fija el tamaño de la ventana actual del navegador al tamaño indicado por su parámetro "targetSize".	

A través del método `manage().window()` accedemos a los métodos anteriormente descritos.

Ejemplo cap9.Window.java
<pre> /* * EJEMPLO: Interfaz WebDriver.Window * ACCIONES * Mostrar por consola la posicion y el tamaño de la ventana abierta por * selenium * Fijar la posicion de la ventana en el punto (150, 50) * Fijar las dimensiones de la ventana a (1000 x 500) * Maximizar la ventana * Cerrar la ventana del navegador */ WebDriver driver = new FirefoxDriver(); driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm"); Thread.sleep(2000); System.out.println(driver.manage().window().getPosition()); System.out.println(driver.manage().window().getSize()); Thread.sleep(2000); driver.manage().window().setPosition(new Point(150,50)); Thread.sleep(2000); driver.manage().window().setSize(new Dimension(1000,500)); Thread.sleep(2000); driver.manage().window().maximize(); Thread.sleep(2000); driver.close(); </pre>

9.4 GESTIÓN DE FRAMES. INTERFAZ

WEBDRIVER.TARGETLOCATOR

Cuando una determinada página web alcanza cierta complejidad, la información que contiene se organiza en marcos (frames). Estos marcos a su vez, contienen pequeñas paginas HTML con la estructura clásica:

```
<HTML><HEAD>...</HEAD><BODY><H1>...</H1>...</BODY></HTML>
```

Para poder gestionar los diferentes marcos la interfaz WebDriver define la interfaz anidada TargetLocator.

Es importante comentar que la interfaz TargetLocator no diferencia entre Frames e iFrames. Los iFrames se declaran en línea sin necesidad de utilizar los <FRAMESET> pero tienen un significado y una utilidad similar a los Frames.

Principales métodos Interfaz WebDriver.TargetLocator

```
public static interface WebDriver.Window
{
    • defaultContent()
    • parentFrame()
    • frame()
```

MÉTODO defaultContent()

Return	Método(Parámetros)
WebDriver	defaultContent()
Descripción	
Este método selecciona la página principal en la cual se realiza el primer frameset. Si se esta trabajando con iframes, selecciona la página principal donde se define la línea en la que aparece el primer iframe. NOTA: Pueden encontrarse frames anidadas ó iframes que hagan referencia a otros iframes. Este método seleccionaría la página origen de todas ellas.	

MÉTODO parentFrame()

Return	Método(Parámetros)
WebDriver	parentFrame()
Descripción	
- Cuando existen frames anidadas o iframes con referencias a otros iframes, este método devuelve la página que contiene el frame o iframe de orden inmediatamente superior. - Por ejemplo: Supongamos que la página principal define el iframe(1) y este a su vez el iframe(1.1) y éste a su vez el iframe(1.1.1). Cuando nos encontremos situados en el iframe(1.1.1) si ejecutamos el método "driver.switchTo().parentFrame()" la página que analizaría nuestro test sería la definida por el iframe(1.1). Si por el contrario optamos por utilizar el método definido anteriormente, "driver.switchTo().defaultContent()", entonces nuestro test analizaría la página donde se definió el iframe(1).	

MÉTODO frame()

Return	Método(Parámetros)
WebDriver	<code>frame(int index)</code> <code>frame(java.lang.String nameOrId)</code> <code>frame(WebElement frameElement)</code>
Descripción	
<p>- Este método permite situarse en un determinado frame o iframe, en función de su parámetro de entrada.</p> <p>- Existen varios modos de poder utilizarlo</p> <p>1)especificando el índice el frame o iframe. Esto es, cuando en una pagina se definen varios frameso iframes, se numeran del 0, a n. Si se quiere acceder al primero de ellos utilizaríamos <code>"driver.switchTo().frame(0)"</code></p> <p>2)especificando el atributo name o id del frame o iframe. Para seleccionar el frame <code><frame name = frameA, id = f1..></frame></code> podríamos utilizar <code>"driver.switchTo().frame("frameA")"</code> o <code>"driver.switchTo().frame("f1")"</code></p> <p>3)especificando el WebElement que hace referencia al frame o iframe. <code>WebElement frame = driver.findElement(By.id("f1"));</code> <code>driver.switchTo().frame(frame);</code></p>	

Ejemplo cap9.TargetLocator.java

```
/*
 * EJEMPLO: Interfaz TargetLocator()
 * ACCIONES
 * Logear al usuario "admin" en la aplicaon de TestLink
 * Hacer click en la opción "Test Specification" de la barra de herraminetas
 * superior
 * Movernos de un frame a otro utilizando los diferentes metodos
 */

ProfilesIni profile = new ProfilesIni();
FirefoxProfile p = profile.getProfile("default");

FirefoxBinary binary = new FirefoxBinary
(new File("C:\\Program Files\\Mozilla Firefox\\firefox.exe"));
FirefoxDriver driver = new FirefoxDriver(binary,p);

driver.navigate().to("http://localhost:8080/testlink-1.9.11/login.php");
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
...

System.out.println("                PRINT MAIN PAGE");
driver.switchTo().frame(driver.findElement(By.name("titlebar")));
System.out.println("                PRINT frame [main.titlebar]");
WebElement e = driver.findElement(By.xpath("html/body/div[3]/a[2]"));
e.click();

driver.switchTo().defaultContent();
System.out.println("                PRINT MAIN PAGE");
driver.switchTo().frame(driver.findElement(By.name("mainframe")));
System.out.println("                PRINT frame [main.mainframe]");
driver.switchTo().frame(driver.findElement(By.name("treeframe")));
System.out.println("                PRINT frame [main.mainframe.treeframe]");

e = driver.findElement(By.xpath(".*/*[@id='extdd-4']/img[1]"));
e.click();

driver.switchTo().parentFrame();
System.out.println("                PRINT frame [main.mainframe]");
driver.switchTo().defaultContent();
System.out.println("                PRINT MAIN PAGE");
driver.switchTo().frame(0);
System.out.println("                PRINT frame [main.titlebar]");
driver.switchTo().parentFrame();
System.out.println("                PRINT MAIN PAGE");
driver.switchTo().frame(1);
System.out.println("                PRINT frame [main.mainframe]");
driver.switchTo().frame(1);
System.out.println("                PRINT frame [main.mainframe.workframe]");
driver.switchTo().parentFrame();
System.out.println("                PRINT frame [main.mainframe]");
WebElement workframe = driver.findElement(By.name("workframe"));
driver.switchTo().frame(workframe);
System.out.println("                PRINT frame [main.mainframe.workframe]");

driver.close();
```

9.5 RESUMEN.

Durante el desarrollo del capítulo se han ido describiendo varias interfaces, cada una de ellas orientada a una funcionalidad específica y compuesta por una serie de métodos. A continuación, resumiremos lo visto en los 5 puntos anteriores:

- Interfaz anidada **WebDriver.Navigation**: Para funciones relacionadas con el proceso de navegación a través de diferentes webs. Ir a la página anterior, a la página siguiente, actualizar página, etc.
- Interfaz anidada **WebDriver.Timeouts**: Para especificar el tiempo de espera máximo para la carga de la página web completa o de sus elementos.
- Interfaz anidada **WebDriver.Window**: Para manejar y conocer la posición y el tamaño de la ventana de navegación de la aplicación web.
- Interfaz anidada **WebDriver.TargetLocator**: Para la gestión de frames/iframes y también de ventanas.

Para cualquier consulta referente al API de Selenium WebDriver existe abundante documentación en línea en la url:

- <http://seleniumhq.github.io/selenium/docs/api/java/>

CAPÍTULO 10

LA CLASE FIREFOXDRIVER.

Una de las características que hace más potente a Selenium WebDriver en comparación con su versión anterior, es el hecho de contar con una implementación específica para cada navegador. Todo lo estudiado hasta ahora es aplicable a cualquier browser, si bien en todos nuestros ejemplos hemos hecho uso de la clase FireFoxDriver, lanzando la ejecución de los scripts sobre Mozilla Firefox. Como se verá a continuación, existe una razón para ello.

En la parte práctica del PFC se desarrollarán unos test automáticos para poner en uso todo lo estudiado sobre Selenium y poder así obtener una valoración de la herramienta de automatización. Estos casos prácticos se desarrollarán para trabajar sobre el navegador Firefox.

En este capítulo realizaremos el estudio de las características que trae consigo la implementación específica realizada por la clase FireFoxDriver. Concretamente veremos:

- Cómo realizar las capturas de pantalla.
- Gestión del perfil de navegación creado por Firefox.
- Gestión de las preferencias del navegador.
- Gestión de la ejecución de múltiples versiones de Firefox.

10.1 CAPTURAS DE PANTALLA. INTERFAZ TAKESCREENSHOT

Durante la ejecución de los tests, en las pruebas que resulten falladas puede ser muy interesante realizar capturas de pantalla para recoger el aspecto que tenía la aplicación web en el momento de detectar el fallo.

Para poder realizar estas capturas de pantalla, Selenium define la interfaz TakesScreenshot. La clase FirefoxDriver, entre otras, implementa dicha interfaz.

MÉTODO getScreenshotAs()

Return	Método(Parámetros)
<X> X	getScreenshotAs (OutputType<X> target)
Descripción	
<p>- El método getScreenshotAs realiza una captura de la pantalla y devuelve dicha captura en el formato/tipo especificado en su parámetro de entrada "target".</p> <p>- EL objeto OutputType indica el tipo o formato de la captura de pantalla:</p> <ul style="list-style-type: none">• OutputType.FILE -> devuelve un fichero <java.io.File>. Este fichero es temporal, existe únicamente durante la ejecución del test.• OutputType.BYTES -> devuelve una array de bytes <byte[]> con el contenido de la captura.• OutputType.BASE64 -> devuelve una cadena <java.lang.String> con contenido de la captura. La información contenida en la cadena de texto esta codificada en base64.	

Ejemplo cap10.ScreenShot.java

```
/*
 * EJEMPLO: Screenshot
 * ACCIONES
 * Se carga la pagina web
 * Se realiza una captura de pantalla en fichero temporal
 * Se realiza una captura de pantalla en array de bytes
 * y se guarda el contenido en disco de forma parmanente
 * Cerrar la ventana del avegador
 */
WebDriver driver = new FirefoxDriver();

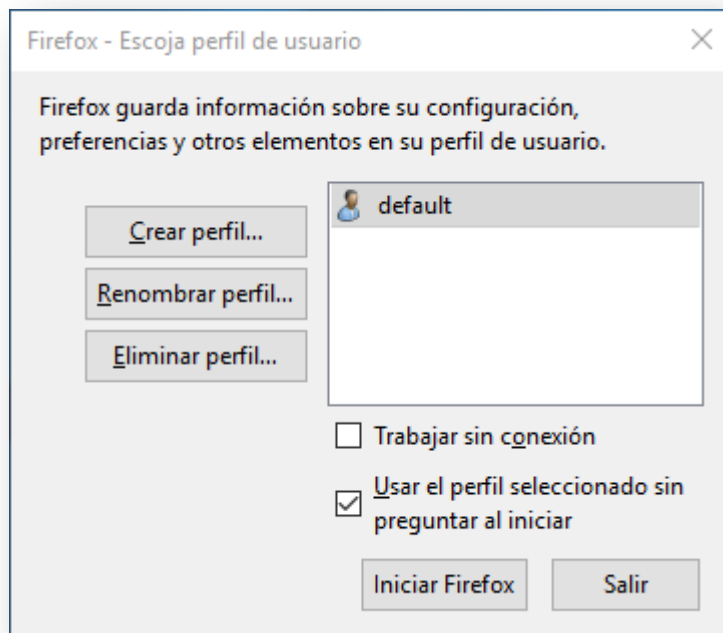
driver.get("file:///C:/WebSites/localhost/mantisbt-1.2.19/login_page.htm");
driver.manage().window().maximize();
//fichero temporal
File f = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
System.out.println(f.getAbsolutePath());
//array de bytes
try {
    FileOutputStream fileOutput = new FileOutputStream
        ("C:\\WebSites\\screenshots\\captura_pantalla_cap_9.png");
    BufferedOutputStream bufferedOutput = new
        BufferedOutputStream(fileOutput);
    bufferedOutput.write
        (((TakesScreenshot)driver).getScreenshotAs(OutputType.BYTES));
    bufferedOutput.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (WebDriverException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
driver.close();
```

10.2 LA CLASE FIREFOXPROFILE

El navegador Firefox utiliza un perfil de usuario para almacenar información relacionada con las contraseñas, preferencias, cookies, historial, apariencia, complementos y extensiones, marcadores (favoritos), etc.

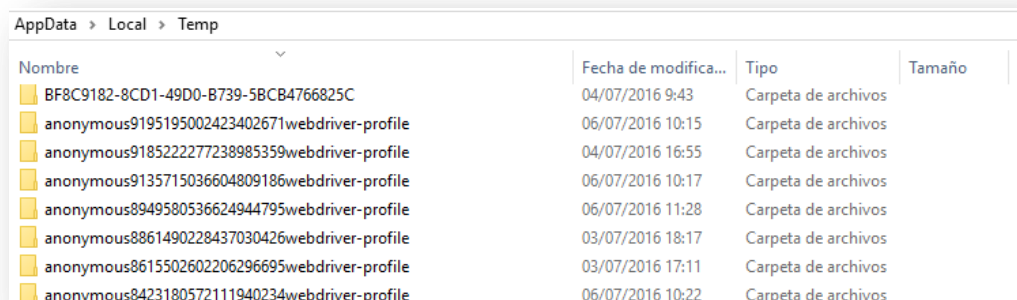
Para administrar los perfiles Firefox cuenta con la herramienta ProfileManager. Para poder acceder a ella es necesario cerrar todas las instancias de Firefox y desde la línea de comandos del símbolo de sistema ejecutar el comando

```
C:\Program Files\Mozilla Firefox>firefox.exe -ProfileManager
```



Cada vez que se lanza la ejecución del navegador desde una instancia de FirefoxDriver, éste crea un nuevo perfil en la carpeta:

C:\Users\usuarioX\AppData\Local\Temp.



Nombre	Fecha de modifica...	Tipo	Tamaño
BF8C9182-8CD1-49D0-B739-5BCB4766825C	04/07/2016 9:43	Carpeta de archivos	
anonymous9195195002423402671webdriver-profile	06/07/2016 10:15	Carpeta de archivos	
anonymous9185222277238985359webdriver-profile	04/07/2016 16:55	Carpeta de archivos	
anonymous9135715036604809186webdriver-profile	06/07/2016 10:17	Carpeta de archivos	
anonymous8949580536624944795webdriver-profile	06/07/2016 11:28	Carpeta de archivos	
anonymous8861490228437030426webdriver-profile	03/07/2016 18:17	Carpeta de archivos	
anonymous8615502602206296695webdriver-profile	03/07/2016 17:11	Carpeta de archivos	
anonymous8423180572111940234webdriver-profile	06/07/2016 10:22	Carpeta de archivos	

Para poder manejar los perfiles del navegador Firefox, el API Selenium WebDriver define la clase FirefoxProfile, de forma que gracias a ella se puede establecer el perfil del navegador para cada la ejecución realizada con FirefoxDriver.

La clase FirefoxProfile define gran cantidad de métodos, en nuestro caso, únicamente nos centraremos en estudiar cómo poder utilizar un perfil determinado para nuestras ejecuciones automáticas y cómo poder establecer ciertas preferencias.

10.1.1 CREACIÓN Y UTILIZACIÓN DE DIFERENTES PERFILES.

La creación del perfil que utilizará Firefox cuando se ejecute la instancia de FirefoxDriver se realiza mediante el constructor de la clase FirefoxProfile.

Constructor de la clase FirefoxProfile

```
public class FirefoxProfile
```

- FirefoxProfile() -> Crea un nuevo perfil vacío.
- FirefoxProfile(java.io.File profileDir) -> Crea un nuevo perfil a partir de uno ya existente.

Ejemplo cap10.Profile1.java

```
/*
 * EJEMPLO: FireFoxProfile
 * ACCIONES
 * Crear un perfil de firefox
 * Abrir el navegador utilizando el nuevo perfil creado
 *
 * NOTA: Al no introducir ninguna modificación en el perfil, este ejemplo es
 * equivalente al haber utilizado el perfil vacío que se crea al ejecutar
 * directamente la sentencia "WebDriver driver = new FirefoxDriver();"
 */
FirefoxProfile p = new FirefoxProfile();

WebDriver driver = new FirefoxDriver(p);
driver.get("https://www.euitt.upm.es/");
driver.manage().window().maximize();
Thread.sleep(10000);

driver.close();
```

Ejemplo cap10.Profile3.java

```
/*
 * EJEMPLO: FireFoxProfile
 * ACCIONES
 * Crear un perfil de firefox
 * Se añade la extensión de FireBug
 * Abrir el navegador utilizando el nuevo perfil creado
 *
 * NOTA: En este ejemplo se observa como en la esquina superior derecha
 * aparece el complemento de FireBug
 */
FirefoxProfile p = new FirefoxProfile();
p.addExtension(new File("C:\\FireBug\\firebug@software.joehewitt.com.xpi"));

WebDriver driver = new FirefoxDriver(p);
driver.get("https://www.euitt.upm.es/");
driver.manage().window().maximize();
Thread.sleep(10000);
driver.close();
```

Ejemplo cap10.Profile3.java

```

/*
 * EJEMPLO: FireFoxProfile
 * ACCIONES
 * Crear un perfil de firefox a partir de un perfil existente
 * Abrir el navegador utilizando el perfil anteriormente cargado
 *
 * NOTA: En este caso el navegador de firefox se ejecutara con todos los
 * complementos y extensiones guardados en el perfil, asi como la
 * configuración de marcadores, apariencia, etc.
 */

FirefoxProfile p = new FirefoxProfile(new
File("C:\\Dropbox\\Proyecto\\Profiles\\default"));

WebDriver driver = new FirefoxDriver(p);
driver.get("https://www.euitt.upm.es/");
driver.manage().window().maximize();
Thread.sleep(10000);

driver.close();

```

10.1.2 GUARDADO Y RECUPERACIÓN DE PERFILES.

En muchas ocasiones, durante la ejecución de los tests, utilizaremos los mismos perfiles, teniendo una especie de colección con configuraciones conocidas. Para poder gestionar esta necesidad la clase `FirefoxProfile` dispone de un método que permite guardar y posteriormente recuperar un determinado perfil.

MÉTODO toJson()

El método `toJson()` devuelve una cadena de caracteres en formato JSON con toda la información del perfil siendo sencillo el poder almacenar el contenido de dicha cadena en un fichero.

Return	Método(Parámetros)
java.lang.String	toJson()
Descripción	
- Este método devuelve el contenido de un determinado perfil en una cadena de caracteres con formato JSON.	
Ejemplo:	
<pre> FirefoxProfile p = new FirefoxProfile(); String JsonProfile = p.toJson(); </pre>	

MÉTODO fromJson()

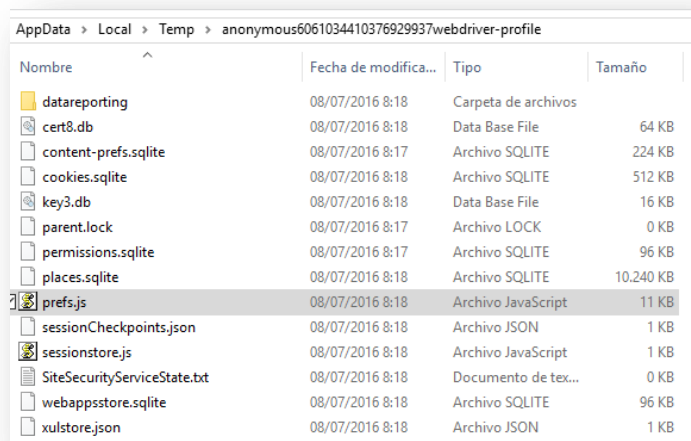
Para realizar la acción contraria, recuperar la información de un perfil existente, podemos hacer uso del método estático `fromJson(java.lang.String json)`.

Return	Método(Parámetros)
FirefoxProfile	fromJson()
Descripción	
- Este método estático devuelve un objeto de tipo FirefoxProfile a partir de una cadena de caracteres con formato JSON que recibe como parámetro de entrada.	
Ejemplo: String JsonProfile = pA.toJson(); FirefoxProfile pB = FirefoxProfile.fromJson(JsonProfile);	

Ejemplo cap10.Profile3.java
<pre>/* * EJEMPLO: FireFoxProfile JSON * ACCIONES * Crear un perfil vacio (profileA) * Añadir la extension de FireBug * Guardar el perfil en formato JSON * Crear un perfil (profileB) a partir de una cadena con info en formato * JSON * Utilizar el perfil profileB en la ejecucion del test */ FirefoxProfile profileA = new FirefoxProfile(); profileA.addExtension(new File("C:\\FireBug\\firebug@software.joehewitt.com.xpi")); String JsonProfile = profileA.toJson(); FirefoxProfile profileB = FirefoxProfile.fromJson(JsonProfile); WebDriver driver = new FirefoxDriver(profileB); driver.get("https://www.euitt.upm.es/"); driver.manage().window().maximize(); Thread.sleep(10000); driver.close();</pre>

10.1.3 CONFIGURACIÓN DE LAS PREFERENCIAS DEL NAVEGADOR.

En el directorio de perfil que crea cada ejecución la instancia de FirefoxDriver podemos encontrar el fichero “prefs.js”. En este fichero se detallan las preferencias del perfil configuradas.



Nombre	Fecha de modifica...	Tipo	Tamaño
datareporting	08/07/2016 8:18	Carpeta de archivos	
cert8.db	08/07/2016 8:18	Data Base File	64 KB
content-prefs.sqlite	08/07/2016 8:17	Archivo SQLITE	224 KB
cookies.sqlite	08/07/2016 8:18	Archivo SQLITE	512 KB
key3.db	08/07/2016 8:18	Data Base File	16 KB
parent.lock	08/07/2016 8:17	Archivo LOCK	0 KB
permissions.sqlite	08/07/2016 8:17	Archivo SQLITE	96 KB
places.sqlite	08/07/2016 8:18	Archivo SQLITE	10.240 KB
prefs.js	08/07/2016 8:18	Archivo JavaScript	11 KB
sessionCheckpoints.json	08/07/2016 8:18	Archivo JSON	1 KB
sessionstore.js	08/07/2016 8:18	Archivo JavaScript	1 KB
SiteSecurityServiceState.txt	08/07/2016 8:18	Documento de tex...	0 KB
webappsstore.sqlite	08/07/2016 8:18	Archivo SQLITE	96 KB
xulstore.json	08/07/2016 8:18	Archivo JSON	1 KB

A continuación podemos ver parte de su contenido:

```
# Mozilla User Preferences

/* Do not edit this file.
 *
 * If you make changes to this file while the application is running,
 * the changes will be overwritten when the application exits.
 *
 * To make a manual change to preferences, you can visit the URL about:config
 */

user_pref("accessibility.lastLoadDate", 1467958667);
user_pref("accessibility.loadedInLastSession", true);
user_pref("app.update.auto", false);
user_pref("app.update.enabled", false);
user_pref("app.update.lastUpdateTime.browser-cleanup-thumbnails", 0);
user_pref("app.update.lastUpdateTime.xpi-signature-verification", 0);
user_pref("browser.EULA.3.accepted", true);
user_pref("browser.EULA.override", true);
user_pref("browser.bookmarks.restore_default_bookmarks", false);
user_pref("browser.cache.disk.capacity", 1048576);
user_pref("browser.cache.disk.filesystem_reported", 1);
user_pref("browser.cache.disk.smart_size.first_run", false);
user_pref("browser.cache.frecency_experiment", 1);
user_pref("browser.displayedE10SNotice", 4);
user_pref("browser.dom.window.dump.enabled", true);
user_pref("browser.download.importedFromSqlite", true);
user_pref("browser.download.manager.showWhenStarting", false)
```

Existen dos tipos de preferencias, aunque a primera vista no existe modo alguno de diferenciarlas. Hay preferencias **alterables** y preferencias **NO alterables**.

Las no alterables son aquellas que crea la instancia FirefoxDriver por defecto. Por ejemplo al ejecutar el siguiente código:

```
WebDriver driver = new FirefoxDriver();
driver.get("https://www.euitt.upm.es/");
```

Al no especificar ningún perfil concreto o al utilizar un perfil vacío (sin aplicar ninguna modificación) la instancia de FirefoxDriver crea el fichero anteriormente mencionado "prefs.js" con la configuración por defecto. Todas las entradas de dicho fichero son preferencias **NO alterables**, es decir para modificar su contenido es necesario utilizar métodos específicos proporcionados por la clase FirefoxProfile.

Por otro lado, las preferencias creadas mediante el método `setPreference()`, se consideran **alterables**.

MÉTODO setPreference()

Return	Método(Parámetros)
void	setPreference(java.lang.String key, java.lang.String value) setPreference(java.lang.String key, boolean value) setPreference(java.lang.String key, int value)
Descripción	
Este método establece la preferencia alterable "key" con el valor "value". Si intentamos establecer un nuevo valor a una preferencia NO alterable el método lanzará una excepción:	
java.lang.IllegalArgumentException: Preference app.update.enabled may not be overridden: frozen value=false, requested value=true	
Ejemplo en cual saltaría una excepción:	
FirefoxProfile p = new FirefoxProfile(); p.setPreference("app.update.enabled", true);	

Para más información sobre la clase FirefoxProfile y sus métodos consultar la información en línea del API Selenium WebDriver:

- <http://seleniumhq.github.io/selenium/docs/api/java/>

Ejemplo cap10.Preferences.java

```
/*
 * EJEMPLO: FireFoxProfile edicion de preferencias
 * ACCIONES
 * Crear un nuevo perfil
 * Intentar modificar una preferencia NO alterable
 * Modificar la preferencia general.useragent.override
 * Lanzar el navegador y cargar la pagina web de la escuela
 *
 * NOTA: En el fichero "prefs.js" del perfil creado por la instancia de
 * FirefoxDriver deberá aparecer la entrada:
 * user_pref("general.useragent.override", "PFC Automatizacion de pruebas
 * con Selenium WebDriver");
 */

FirefoxProfile p = new FirefoxProfile();
try{
    p.setPreference("app.update.enabled", true);
}catch (Exception e){
    System.out.println(e);
}
p.setPreference("general.useragent.override", "PFC Automatizacion de pruebas
con Selenium WebDriver");

WebDriver driver = new FirefoxDriver(p);
driver.get("https://www.euitt.upm.es/");
driver.manage().window().maximize();

Thread.sleep(10000);
driver.close();
```

10.3 LA CLASE FIREFOXBINARY

Es bastante habitual en la ejecución de pruebas de una aplicación web, que dichas pruebas se realices sobre varios navegadores y sobre varias versiones de esos navegadores. Por ejemplo, es posible que en la versión 45 de Firefox la aplicación se comporte correctamente, pero no lo haga para versiones anteriores o para versiones posteriores.

Para poder lanzar la ejecución de los test sobre una versión concreta de Firefox, debemos ayudarnos de la clase `FirefoxBinary`, la cual se encarga de especificar que ejecutable del navegador se debe lanzar cuando se instancie un objeto de la clase `FirefoxDriver`.

A continuación, veremos un ejemplo donde se especifica que versión de Firefox debe ejecutar la instancia de `FirefoxDriver`.

Ejemplo cap10.Binary.java

```
/*
 * EJEMPLO: FireFoxProfile
 * ACCIONES:
 * Crear un perfil de firefox a partir del perfil por defecto
 * Cargar la versión 47 de firefox
 * Abrir el navegador utilizando el perfil creado y el ejecutable de la
 * versión 47 de firefox
 */
ProfilesIni profile = new ProfilesIni();
FirefoxProfile p = profile.getProfile("default");

FirefoxBinary firefox47 = new FirefoxBinary(new File("C:\\Program
Files\\Mozilla Firefox 47\\firefox.exe"));
FirefoxDriver driver = new FirefoxDriver(firefox47,p);

driver.get("https://www.euitt.upm.es/");
driver.manage().window().maximize();

Thread.sleep(5000);
driver.close();
```

PARTE 3ª. AUTOMATIZACIÓN DE PRUEBAS FUNCIONALES CON SELENIUM WEBDRIVER.

CAPÍTULO 11

DESCRIPCIÓN DEL CASO PRÁCTICO

En este capítulo que abre la parte práctica del PFC aclararemos los siguientes temas:

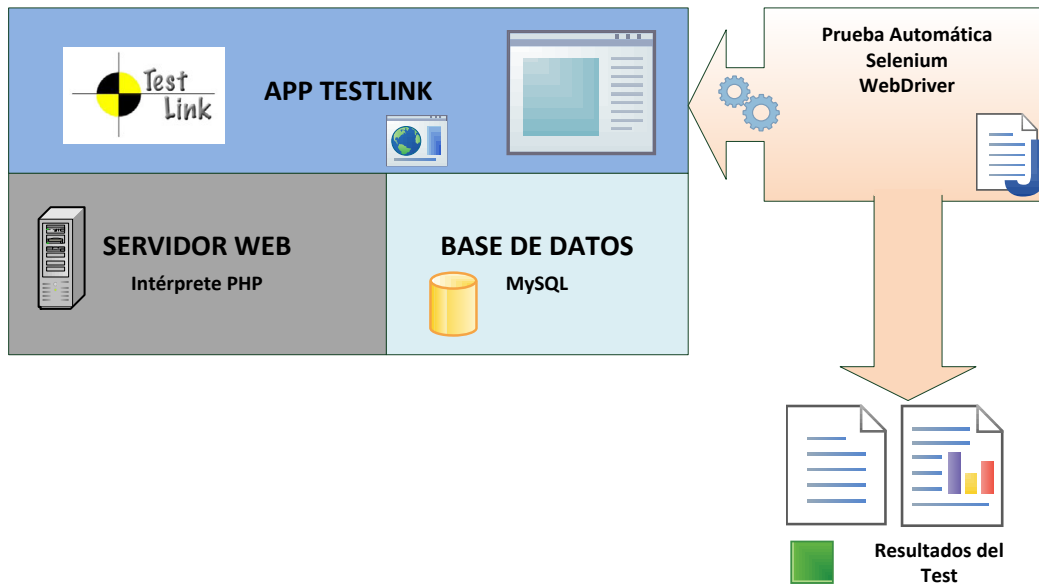
- Sobre qué aplicación web enfocaremos nuestros test automáticos. Sobre qué navegador.
- Qué propósito tendrán las pruebas realizadas y en qué condiciones se ejecutarán.
- Que objetivos persigue la tercera parte del proyecto.
- Que software necesitamos para poder llevar a cabo ésta parte práctica.

11.1 APLICACIÓN WEB UTILIZADA PARA LOS TEST AUTOMÁTICOS.

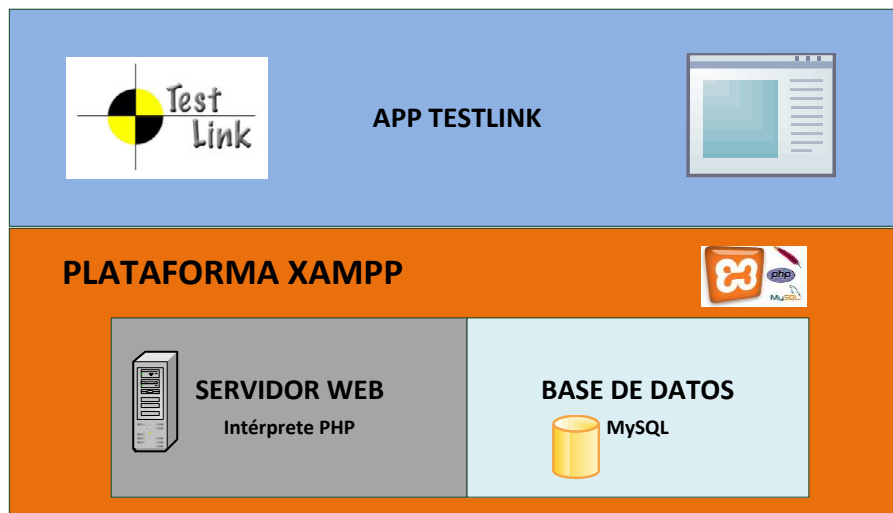
Para poder poner en práctica todo lo aprendido durante la fase de estudio de la herramienta Selenium WebDriver, necesitaremos establecer un escenario que nos permita poder operar sobre una determinada aplicación web. En los ejemplos de capítulos anteriores se ha hecho referencia fundamentalmente a dos aplicaciones, Mantis y TestLink. Para ésta parte práctica nos centraremos únicamente en TestLink, de forma que nuestros test automáticos probarán ciertas funcionalidades de esta aplicación web.

TestLink es una herramienta gratuita para la gestión de casos de prueba que permite definir tests, organizarlos en planes de prueba, ejecutarlos, generar informes, etc. Para ello, la aplicación nos ofrece una interfaz web donde cada usuario accede a ella a través de una URL, logándose mediante su nombre y contraseña.

La siguiente imagen muestra de manera global la arquitectura del escenario para esta parte práctica.



Testlink está desarrollada en PHP y necesita ser desplegada como parte de un servicio sobre un servidor web. También necesita hacer uso de una base de datos para ir almacenando la información que maneja. Para facilitar su despliegue utilizaremos la plataforma XAMPP que nos proporciona el intérprete PHP, un servidor web Apache y una base de datos MySQL.



Queda fuera del alcance del capítulo y también de proyecto los detalles sobre la instalación de todas estas aplicaciones. En diferentes recursos web existe abundante información sobre los pasos y las acciones necesarias para la instalación de todas estas herramientas.

11.2 SUPUESTO PRÁCTICO.

Toda la parte práctica del PFC se desarrollará en base a un escenario que intentará simular una situación real durante el proceso de desarrollo/mantenimiento de una aplicación web.

Concretamente partiremos del siguiente supuesto:

Teniendo una versión beta de la aplicación TestLink (1.9.11) se efectúan una serie de cambios y mejoras en el código de algunos componentes que interactúan con la base de datos y que intervienen en la funcionalidad de creación de usuarios/roles (CREATE_USERS/ROLES) y en la funcionalidad de gestión de los test cases y test suits que componen un proyecto (TESTCASE_MANAGEMENT). Estos cambios no deberán afectar a la interfaz de usuario ni a la funcionalidad prestada.

Para realizar el seguimiento de las modificaciones implementadas, el equipo de calidad decide empezar por diseñar una serie de pruebas de humo para verificar que no existen bloqueos o errores graves en las funcionalidades mencionadas (CREATE_USER/ROLES y TESTCASE_MANAGEMENT).

Las pruebas de humo deben ser ejecutadas sobre varias versiones del navegador Firefox, ya que la aplicación debe ser capaz de funcionar con normalidad en todas ellas. Concretamente, los test se lanzarán sobre las versiones:

- Firefox 43.0
- Firefox 44.0
- Firefox 45.0

Estos test automáticos se deberán ejecutar cada vez que se apruebe un cambio en los componentes afectados. Cuando el proceso de refactorización concluya, el equipo de calidad efectuará una batería de pruebas de regresión más exhaustiva antes de dar por buena la nueva versión de la aplicación (Esta última parte quedaría fuera del alcance del supuesto práctico, únicamente se menciona para qué el lector conozca los procesos que se desencadenarían en una situación real)

11.3 OBJETIVOS PRINCIPALES.

Los objetivos principales de la tercera parte del PFC serán:

- Crear una clase que defina de forma general un test automático.
- Crear el modelo de clases necesario para poder soportar la ejecución de un conjunto determinado de pruebas bajo unas condiciones definidas (Definir el modelo de Automatización).
- Utilizar Selenium WebDriver para poder llevar a cabo las diferentes acciones que tomaría un usuario real sobre la aplicación web para comprobar una funcionalidad determinada.
- Conseguir que un mismo test pueda ser ejecutado sobre varias versiones del navegador de forma transparente para él (Desacoplar la definición del test del entorno de ejecución).
- Evaluar la experiencia obtenida durante el uso de Selenium WebDriver y obtener conclusiones.

11.4 SOFTWARE Y APLICACIONES UTILIZADAS.

A continuación se lista el software utilizado durante la parte práctica del PFC:

- XAMPP Versión 5.6.15
 - APACHE 2.4.4 + MYSQL 5.5.32 (COMMUNITY SERVER) + PHP 5.6.15
- TESTLINK 1.9.11(THE ROBOTS OF DAWN)
- Complementos del navegador Firefox:
 - FireBug versión 2.0.17
 - FirePath 0.9.7.1.1
- Selenium WebDriver (Java API) 2.53.0
- ECLIPSE IDE FOR JAVA DEVELOPERS, VERSION: MARS.1 RELEASE (4.5.1)
- Java SE versión 8 (1.8.0_101)

CAPÍTULO 12

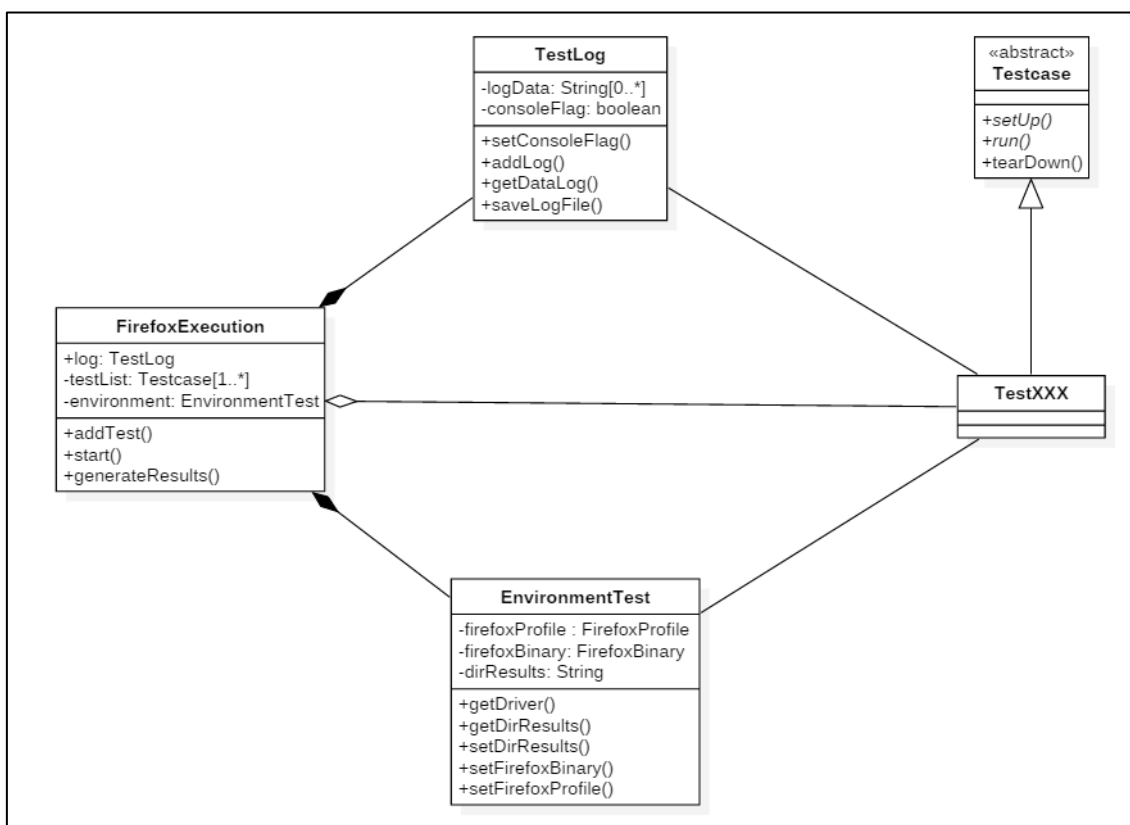
MODELO DE AUTOMATIZACIÓN

La ejecución automática de un conjunto o set de tests requiere de un mecanismo que la soporte. A continuación, describiremos el modelo de automatización diseñado, según el cual, podremos definir los test, seleccionar cuáles de ellos queremos ejecutar, lanzar su ejecución y por último registrar los resultados obtenidos.

Las clases que conforman nuestro modelo de automatización son 5:

- *FirefoxExecution*
- *Testcase*
- *FailedTest*
- *EnvironmentTest*
- *TestLog*

Todas estas clases pertenecen al paquete *automation*. El siguiente diagrama UML representa la relación existente entre cada una de ellas.



12.1 LA CLASE TESTCASE

El primer paso que debemos tomar a la hora de ir construyendo nuestro sistema de automatización es definir de una manera estándar la entidad que represente a un test. En nuestro caso, las diferentes pruebas automáticas estarán representadas por la clase Testcase que define el modelo genérico de un caso de prueba.

automation.Testcase.java

Clase abstracta que define el patrón que deberán tener todos los casos de prueba. Cualquier test debe heredar siempre se esta clase.

```
public abstract class Testcase {  
  
    abstract public void setUp(EnvironmentTest environment, TestLog log)  
    throws FailedTest;  
    abstract public void run() throws FailedTest;  
    abstract public void tearDown();  
  
}
```

Métodos de la clase Testcase:

Return	Método(Parámetros)
void	setUp(EnvironmentTest environment, TestLog log)
Descripción	
<p>Método que prepara las precondiciones necesarias para la ejecución del test, estableciendo la configuración y el entorno se ejecutará la prueba.</p> <p>Params:</p> <ul style="list-style-type: none">- environment: parámetro de tipo <EnvironmentTest> que contiene información sobre el entorno de ejecución.- log: parámetro de tipo <TestLog> que permite que el test pueda registrar información de log durante su ejecución. <p>El método setup() si detecta cualquier problema durante su ejecución, lanza una excepción del tipo <FailedTest></p>	

Return	Método(Parámetros)
void	run()
Descripción	
<p>En este método se definen las acciones que ha de ir realizando la prueba. Por ejemplo, si el test consiste en hacer click sobre un determinado elemento de la aplicación web (botón, checkbox, etc.) y después comprobar el resultado de esa acción, la definición de todos los pasos necesarios para ello deben ir en el método run() del testcase.</p> <p>Si durante su ejecución, éste método se encuentra con algún resultado no esperado lanza una excepción del tipo <FailedTest></p>	

Return	Método(Parámetros)
void	tearDown()
Descripción	
Método que realiza las acciones necesarias después de la ejecución de una prueba. Estas acciones pueden consistir en liberar recursos utilizados por el test, generación de resultados, etc.	

12.2 LA CLASE FIREFOXEXECUTION

Durante el diseño del modelo de automatización, se ha tenido en cuenta que una aplicación web real debe ser capaz de ejecutarse correctamente en diferentes navegadores. Por ejemplo, TestLink deberá funcionar para los navegadores más comunes, como pueden ser Chrome, IE, Firefox, etc., y dentro de esos navegadores deberá ser compatible con sus diferentes versiones. Por tanto, es necesario que las pruebas de la aplicación estén totalmente desacopladas del entorno de ejecución, de forma que un mismo test pueda ser ejecutado sobre diferentes navegadores de forma transparente para él.

La clase FirefoxExecution representa la ejecución de un set o batería de tests dentro de un entorno de pruebas específico.

automation.FirefoxExecution.java
Clase que representa la ejecución de una batería de pruebas sobre el navegador Firefox.
<pre> public class FirefoxExecution{ public TestLog log = null; private List<Testcase> testList = null; private EnvironmentTest environment = null; public FirefoxExecution(String browser, String profile, String dirResults){} public void addTest(Testcase t){} public void start(){} public void generateResults(){} }</pre>

Métodos de la clase FirefoxExecution:

Return	Método(Parámetros)
void	addTest(Testcase t)
Descripción	
Método para añadir tests a la ejecución de la batería de pruebas.	
Params:	
- t : Objeto de tipo <Testcase>. Representa el test que se quiere añadir a la ejecución.	

Return	Método(Parámetros)
void	start()
Descripción	
Este método inicia la ejecución del set de tests. La ejecución se realiza en el mismo orden en el cual fueron añadidas las pruebas.	

Return	Método(Parámetros)
void	generateResults()
Descripción	
Este método genera el archivo de log con el resultado de la ejecución. En él se pueden consultar los resultados obtenidos por cada uno de los test ejecutados.	
El fichero de log que genera éste método se guarda en el directorio indicado por el atributo environment.	

12.3 LA CLASE ENVIRONMENTTEST

En el apartado anterior se ha mencionado la importancia que tiene poder desacoplar la definición del test del entorno sobre el cual se ejecuta, por ello, uno de los atributos de la clase `FirefoxExecution` es un objeto de tipo `EnvironmentTest`. La clase `EnvironmentTest` es la encargada de proporcionar la información que necesita el `Testcase` para conocer las condiciones sobre las que se ejecutará, con la ventaja de que éste proceso se producirá en tiempo de ejecución.

automation.EnvironmentTest.java

Clase que representa el entorno de ejecución de la prueba.

```
public class EnvironmentTest{

    private FirefoxProfile firefoxProfile = null;
    private FirefoxBinary firefoxBinary = null;
    private String dirResults = "";

    public EnvironmentTest(FirefoxBinary binary, FirefoxProfile profile,
        String dir){}

    public WebDriver getDriver() {}

    public String getDirResults(){}

    public void setDirResults(String dir){}

    public void setFirefoxBinary(FirefoxBinary binary){}

    public void setFirefoxProfile(FirefoxProfile profile){}

}
```

Métodos de la clase EnvironmentTest:

Return	Método(Parámetros)
WebDriver	getDriver()
Descripción	
Método para obtener el objeto <code>WebDriver</code> que controla la interacción con el navegador web.	
Se devuelve un objeto de tipo <code>WebDriver</code> caracterizado por los atributos <code>firefoxBinary</code> y <code>firefoxProfile</code> .	

Return	Método(Parámetros)
String	getDirResults()
Descripción	
Método que devuelve la dirección del directorio donde se guardan los resultados de la ejecución.	

Return	Método(Parámetros)
void	setDirResults(String dir)
Descripción	
<p>Método para establecer la dirección del directorio donde guardar los resultados de ejecución.</p> <p>Params:</p> <ul style="list-style-type: none"> - dir: Objeto de tipo <String> que indica el directorio donde guardar los resultados de ejecución. 	

Return	Método(Parámetros)
void	setFirefoxBinary(FirefoxBinary binary)
Descripción	
<p>Método que establece que versión (archivo ejecutable) del navegador Firefox soportará la ejecución de los tests.</p> <p>Params:</p> <ul style="list-style-type: none"> - binary: Objeto de tipo <FirefoxBinary> que representa el fichero ejecutable de la versión del navegador Firefox que se utilizará durante la ejecución de los tests. 	

Return	Método(Parámetros)
void	setFirefoxProfile(FirefoxProfile profile)
Descripción	
<p>Método que establece el perfil del navegador que utilizará Firefox durante la ejecución de los tests.</p> <p>Params:</p> <ul style="list-style-type: none"> - profile: Objeto de tipo <FirefoxProfile> que representa el perfil del navegador Firefox que se utilizará durante la ejecución de los tests. 	

12.4 LA CLASE TESTLOG

La clase TestLog ofrece un servicio de registro de mensajes de log para que los diferentes tests puedan ir grabando información relacionada con su ejecución. Esta información posteriormente puede ser guardada en un fichero.

automation.TestLog.java

La clase TestLog va registrando en una lista los mensajes de log de diferentes fuentes. Posteriormente esa información puede ser guardada en un fichero o ser devuelta como una cadena de caracteres.

```
public class TestLog {  
  
    private List<String> logData = null;  
    private boolean consoleFlag = false;  
  
    public TestLog(){}  
  
    public void setConsoleFlag(boolean consoleFlag){}  
  
    public void addLog(Object sourceLog, String logMsg){}  
  
    public String getDataLog(){}  
  
    public void saveLogFile(String logFile){}  
  
}
```

Métodos de la clase TestLog:

Return	Método(Parámetros)
void	setConsoleFlag (boolean consoleFlag)
Descripción	
<p>Método que permite activar o desactivar los logs por consola. Este método modifica el valor del atributo consoleFlag.</p> <p>Params:</p> <ul style="list-style-type: none">- consoleFlag: Objeto de tipo <boolean> que indica si se desea mostrar o no por consola, los mensajes de log registrados por el objeto TestLog.	

Return	Método(Parámetros)
void	addLog (Object sourceLog, String logMsg)
Descripción	
<p>Este método es el encargado de registrar un mensaje de log. Para ello, es necesario que se le indique el mensaje a registrar y el autor o fuente de dicho mensaje.</p> <p>Params:</p> <ul style="list-style-type: none">* sourceLog: Objeto que quiere registrar un mensaje de log. Por ejemplo, podría ser un objeto de tipo Testcase que necesita registrar algún mensaje de log durante su ejecución.* logMsg: Objeto de tipo <String> que representa el mensaje (la información) a registrar.	

Return	Método(Parámetros)
String	getDataLog ()
Descripción	
<p>Método que devuelve toda la información de log registrada hasta el momento en el cual se invoca. La información se devuelve en un objeto de tipo <String></p>	

Return	Método(Parámetros)
void	saveLogFile (String logFile)
Descripción	
<p>Método que guarda en un fichero toda la información de log registrada hasta el momento en el cual se invoca. El fichero donde se desea guardar la información de log se le pasa como parámetro.</p> <p>Params:</p> <ul style="list-style-type: none">-* logFile: Objeto de tipo <String> que indica la localización del fichero donde se desea guardar la información de log.	

CAPÍTULO 13

DEFINICIÓN DE LOS CASOS DE PRUEBA

Partiendo del supuesto descrito en el apartado 11.3 del capítulo 11, se desarrollarán ciertas pruebas de humo con el fin de poder constatar que los trabajos de mejora que se están realizando en ciertos componentes software no afectan a las funcionalidades:

CREATE_USERS/ROLES y TESTCASE/TESTSUITE_MANAGEMENT.

Estos tests se ejecutarán gran número de veces, por lo que el equipo de calidad ha decidido invertir tiempo en automatizarlos.

Normalmente las pruebas de humo consisten en operaciones completas (end-to-end) de una funcionalidad/funcionalidades prestadas por el software que se quiere testear, y su principal objetivo es comprobar que no existen fallos graves que puedan provocar bloqueos o comportamientos no esperados del software.

Este tipo de pruebas suelen ser las primeras en ejecutarse, ya que si se detectan fallos ya en este nivel, el desarrollo de la aplicación debe detenerse para realizar las correcciones necesarias.

A continuación, se llevará a cabo una descripción de los casos de pruebas que se aplicaran para cada una de las dos funcionalidades mencionadas anteriormente.

13.1 TESTS PARA CUBRIR LA FUNCIONALIDAD “CREATE_USERS/ROLES”

La funcionalidad “CREATE_USERS/ROLES” permite a un usuario con rol de Administrador crear nuevos usuarios en el sistema. Estos nuevos usuarios son añadidos a la base de datos y pueden acceder a la aplicación logándose con su username y password. En función del rol asignado, el usuario tendrá disponibles unas opciones u otras.

Para cubrir esta funcionalidad, durante el proceso de refactorización se han diseñado 3 pruebas de humo end-to-end que se describirán a continuación:

testsuit.createroles.TestAdminRole
Descripción
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>Prueba que cubre todo el proceso de creación de un usuario nuevo con rol de "Administrator". Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "User/Roles" de la barra de opciones superior. • Captura de pantalla pre-test. • Crear usuario con los datos de entrada. • Captura de pantalla post-test. • Logout: admin [admin]. • Login: newUser [role]. • Comprobar inicio de sesión newUser. • Captura de pantalla de main-screen (newUser). • Guardar estado de la base de datos post-test. • Comprobar opciones propias del rol de administrador mostradas en la página principal del usuario.
Objetivo
<p>Completar el proceso completo de creación de un usuario con rol de administrador y comprobar que no se detectan errores bloqueantes.</p>
Datos de entrada
<ul style="list-style-type: none"> • Login name: adminPFC • First name: User • Last name: Roles • Password: 12345 • Email: admin_pfc@mymail.com • Role: admin • Locale: English (wide/UK) • Authentication method: Default (DB)
Datos de salida. Resultados
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Captura de pantalla main-screen (newUser). • Base de datos post-test. • Log de ejecución.
Condiciones Iniciales
<p>Base de datos únicamente con un usuario dado de alta -> admin [admin]</p> <p>ddbb/escenario/init_DDBB.sql</p>

testsuit.createroles.TestSeniorTesterRole
Descripción
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>Prueba que cubre todo el proceso de creación de un usuario nuevo con rol "senior tester". Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "User/Roles" de la barra de opciones superior. • Captura de pantalla pre-test. • Crear usuario con los datos de entrada. • Captura de pantalla post-test. • Logout: admin [admin]. • Login: newUser [role]. • Comprobar inicio de sesión newUser. • Captura de pantalla de main-screen (newUser). • Guardar estado de la base de datos post-test.
Objetivo
<p>Completar el proceso completo de creación de un usuario con rol "senior tester" y comprobar que no se detectan errores bloqueantes.</p>
Datos de entrada
<ul style="list-style-type: none"> • Login name: seniorPFC • First name: User • Last name: Roles • Password: 12345 • Email: senior_pfc@mymail.com • Role: senior tester • Locale: English (wide/UK) • Authentication method: Default (DB)
Datos de salida. Resultados
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Captura de pantalla main-screen (newUser). • Base de datos post-test. • Log de ejecución.
Condiciones Iniciales
<p>-> Haber ejecutado previamente el test TestAdminRole</p> <p>Esta prueba se debe ejecutar de manera encadenada con el test anterior (TestAdminRole) ya que el nuevo usuario debe ser el creado por adminPFC.</p>

testsuit.createroles.TestGuestRole
Descripción
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>Prueba que cubre todo el proceso de creación de un usuario nuevo con rol de "guest". Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "User/Roles" de la barra de opciones superior. • Captura de pantalla pre-test. • Crear usuario con los datos de entrada. • Captura de pantalla post-test. • Logout: admin [admin]. • Login: newUser [role]. • Comprobar inicio de sesión newUser. • Captura de pantalla de main-screen (newUser). • Guardar estado de la base de datos post-test.
Objetivo
<p>Completar el proceso completo de creación de un usuario con rol "guest" y comprobar que no se detectan errores bloqueantes.</p>
Datos de entrada
<ul style="list-style-type: none"> • Login name: guestPFC • First name: User • Last name: Roles • Password: 12345 • Email: guest_pfc@mymail.com • Role: guest • Locale: English (wide/UK) • Authentication method: Default (DB)
Datos de salida. Resultados
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Captura de pantalla main-screen (newUser). • Base de datos post-test. • Log de ejecución.
Condiciones Iniciales
<p>-> Haber ejecutado previamente los tests TestAdminRole y TestSeniorTesterRole</p>

13.2 TESTS PARA CUBRIR LA FUNCIONALIDAD “TESTCASE/TESTSUIT_MANAGEMENT”

Un proyecto en Testlink contiene una serie de casos de prueba (Test cases) organizados en diferentes carpetas o contenedores (Test suits).

La funcionalidad TESTCASE/TESTSUIT_MANAGEMENT se encarga de gestionar los diferentes ítems (test cases y test suits) permitiendo añadir, borrar, mover, etc. cada uno de ellos. Esta funcionalidad debe estar en continua comunicación con la base de datos para registrar cualquier cambio en la información que maneja.

Para cubrir esta característica, durante el proceso de refactorización se han diseñado 4 pruebas de humo que se describirán a continuación:

testsuit.testmanagement.TestMoveTestcase	
Descripción	
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>La prueba consiste en mover un caso de prueba de un testsuit a otro. Esta acción se realiza simulando el ratón del PC, de forma que se selecciona el testcase con el botón izquierdo y dejándolo presionado se desplaza hasta el testsuit donde quiere ser movido. Finalmente se libera el botón izquierdo del ratón y el testcase debe quedar en el interior del nuevo testsuit.</p> <p>Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "Test Specification" de la barra de opciones superior. • Despliegue de todos los testsuits para muestren su contenido. • Captura de pantalla pre-test. • Seleccionar con el ratón el test "Test_A1_001" y arrastrarlo dentro del testsuit "Testsuit_A1" • Captura de pantalla post-test. • Guardar estado de la base de datos post-test. 	
Objetivo	
Comprobar que la funcionalidad de mover un testcase de un testsuit a otro no presenta errores ni bloqueos.	
Datos de salida. Resultados	
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Base de datos post-test. • Log de ejecución. 	
Condiciones Iniciales	
<p>Es necesario cargar la base de datos indicada para que la aplicación de Testlink tenga registrado el usuario admin [admin] junto con el proyecto que contiene los testsuits y testcases necesarios para la prueba.</p> <p>dldb/escenarioPFC/tc_DDBB.sql</p>	

testsuit.testmanagement.TestMoveTestsuit
Descripción
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>La prueba consiste en mover un testsuit dentro de otro. Esta acción se realiza mediante el menú derecho de opciones del propio testsuit.</p> <p>Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "Test Specification" de la barra de opciones superior. • Despliegue de todos los testsuits para muestren su contenido. • Captura de pantalla pre-test. • Seleccionar con el ratón el testsuit "Testsuit_A1" • Desplegar el menú de edición del testsuit. • Hacer click (con el botón izquierdo del ratón) en el botón [Move/Copy] de las opciones del testsuit (Test Suite Operations). • Seleccionar el destino donde será movido el testsuit. • Hacer click (con el botón izquierdo del ratón) en el botón [Move] • Captura de pantalla post-test. • Guardar estado de la base de datos post-test.
Objetivo
Comprobar que la funcionalidad de mover un testsuit no presenta errores ni bloqueos.
Datos de salida. Resultados
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Base de datos post-test. • Log de ejecución.
Condiciones Iniciales
Esta prueba esta diseñada para ser ejecutada de forma encadenada tras el test TestMoveTestcase.

testsuit.testmanagement.TestDeleteTestcase
Descripción
<p>TIPO DE PRUEBA: Prueba de humo end-to-end.</p> <p>La prueba consiste borrar un determinado testcase utilizando las opciones disponibles en el menú derecho del propio caso de prueba.</p> <p>Pasos principales:</p> <ul style="list-style-type: none"> • Login: admin [admin] • Selección de la opción "Test Specification" de la barra de opciones superior. • Despliegue de todos los testsuits para muestren su contenido. • Captura de pantalla pre-test. • Seleccionar con el ratón el test "PFC_P01-2:Test_A002" • Desplegar el menú de edición del testcase. • Hacer click (con el botón izquierdo del ratón) en el botón [Delete] de las opciones del testsuit (Test Suite Operations). • Confirmar el borrado. • Captura de pantalla post-test. • Guardar estado de la base de datos post-test.
Objetivo
Comprobar que la funcionalidad de eliminar un testcase no presenta errores ni bloqueos.
Datos de salida. Resultados
<ul style="list-style-type: none"> • Captura de pantalla pre-test. • Captura de pantalla post-test. • Base de datos post-test. • Log de ejecución.
Condiciones Iniciales
Esta prueba esta diseñada para ser ejecutada de forma encadenada tras los tests TestMoveTestcase y TestMoveTestsuit.

testsuit.testmanagement.TestDeleteTestsuit

Descripción

TIPO DE PRUEBA: Prueba de humo end-to-end.

La prueba consiste borrar un determinado testsuit utilizando las opciones disponibles en el menú derecho del propio caso de prueba.

Pasos principales:

- Login: admin [admin]
- Selección de la opción "Test Specification" de la barra de opciones superior.
- Despliegue de todos los testsuits para muestren su contenido.
- Captura de pantalla pre-test.
- Seleccionar con el ratón el testsuit "Testsuit_A1"
- Desplegar el menú de edición del testsuit.
- Hacer click (con el botón izquierdo del ratón) en el botón [Delete] de las opciones del testsuit (Test Suite Operations).
- Confirmar el borrado.
- Captura de pantalla post-test.
- Guardar estado de la base de datos post-test.

Objetivo

Comprobar que la funcionalidad de eliminar un testsuit no presenta errores ni bloqueos.

Datos de salida. Resultados

- Captura de pantalla pre-test.
- Captura de pantalla post-test.
- Base de datos post-test.
- Log de ejecución.

Condiciones Iniciales

Esta prueba esta diseñada para ser ejecutada de forma encadenada tras los tests TestMoveTestcase, TestMoveTestsuit y TestDeleteTestcase.

CAPÍTULO 14

DEFINICIÓN DE LOS ENTORNOS DE EJECUCIÓN

En el capítulo anterior se realizó la definición de lo que se quiere probar, es decir, los casos de prueba, ahora el siguiente paso es definir en qué condiciones se quieren ejecutar dichos casos de prueba.

Cada ejecución irá enfocada a probar una funcionalidad concreta (CREATE_USERS/ ROLES o TESTCASE/TESTSUIT_MANAGEMENT) sobre una versión determina de Firefox (v43, v44 o v45).

CREATE_USERS/ ROLES

Ejecuciones:

- **Firefox43Default:** Ejecución de las pruebas utilizando la versión 43 del navegador Firefox. Perfil utilizado -> Default
- **Firefox44Default:** Ejecución de las pruebas utilizando la versión 44 del navegador Firefox. Perfil utilizado -> Default
- **Firefox45Default:** Ejecución de las pruebas utilizando la versión 45 del navegador Firefox. Perfil utilizado -> Default

Tests:

- TestAdminRole.java
- TestGuestRole.java
- TestSeniorTesterRole.java

TESTCASE/TESTSUIT_MANAGEMENT

Ejecuciones:

- **Firefox43Default:** Ejecución de las pruebas utilizando la versión 43 del navegador Firefox. Perfil utilizado -> Default
- **Firefox44Default:** Ejecución de las pruebas utilizando la versión 44 del navegador Firefox. Perfil utilizado -> Default
- **Firefox45Default:** Ejecución de las pruebas utilizando la versión 45 del navegador Firefox. Perfil utilizado -> Default

Tests:

- TestDeleteTestcase.java
- TestDeleteTestsuit.java
- TestMoveTestcase.java
- TestMoveTestsuit.java

Para que las pruebas de humo sean satisfactorias, se debe obtener los mismos resultados con independencia de la versión del navegador Firefox utilizado.

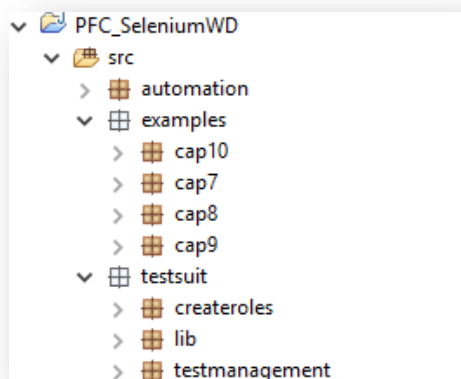
CAPÍTULO 15

PROYECTO EN ECLIPSE

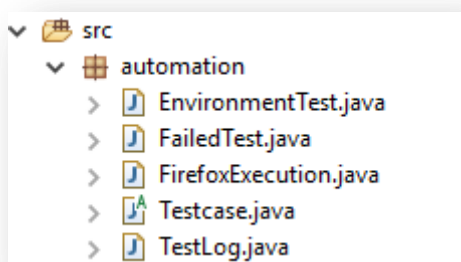
Para el desarrollo de la parte práctica del PFC y también para la implementación de los diferentes ejemplos mostrados durante el estudio de Selenium, se ha creado un proyecto en Eclipse. En éste capítulo se detalla la estructura del mismo.

15.1 ESTRUCTURA DE LOS PAQUETES JAVA

El contenido de las diferentes clases implementadas en eclipse se organiza en 3 paquetes de primer nivel: *automation*, *examples* y *testsuit*.



El paquete ***automation*** contiene las clases que componen el modelo de automatización:

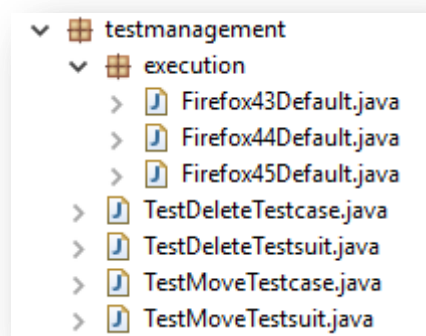
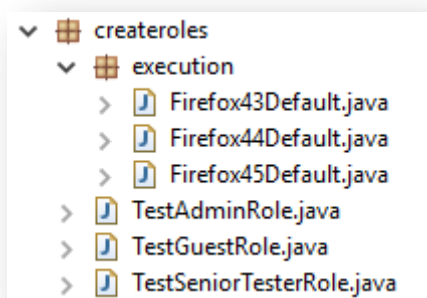


Por otro lado, el paquete ***examples*** contiene los ejemplos mostrados en los capítulos 7, 8, 9 y 10 correspondientes a la parte de estudio de Selenium.

Finalmente, en el paquete **testsuit** podemos encontrar la implementación de los casos de prueba definidos en el capítulo 13. Estos tests se organizan a su vez en los paquetes **testsuit.createroles** (implementación de los casos de prueba referentes a la funcionalidad CREATE_USERS/ROLES) y **testsuit.testmanagement** (implementación de los tests referentes a la funcionalidad TESTCASE/TESTSUIT_MANAGEMENT).

Existe también el paquete **testsuit.lib** donde se ubican diferentes clases que ofrecen funcionalidades comunes que utilizan los testcases, sobre todo los relacionados con la creación de usuarios, login/logout, captura de pantalla y gestión de la base de datos que utiliza la aplicación de Testlink.

Antes de terminar este apartado, es necesario comentar que tanto en el paquete **testsuit.createroles** como en el **testsuit.testmanagement** se diferencia la implementación de los diferentes casos de prueba de la ejecución de los mismos. Las clases encargadas de lanzar la ejecución de los tests, dentro de los diferentes entornos de prueba, se encuentran en los paquetes **execution**.



15.2 DATOS DE ENTRADA NECESARIOS PARA LOS TESTCASES.

Para la ejecución de los casos de prueba, en muchas ocasiones son necesarios algunos datos de entrada para poder realizar su ejecución. Por ejemplo, en las pruebas TestAdminRole y TestMoveTestcase se necesita tener cargada en la aplicación de testlink una base de datos concreta.

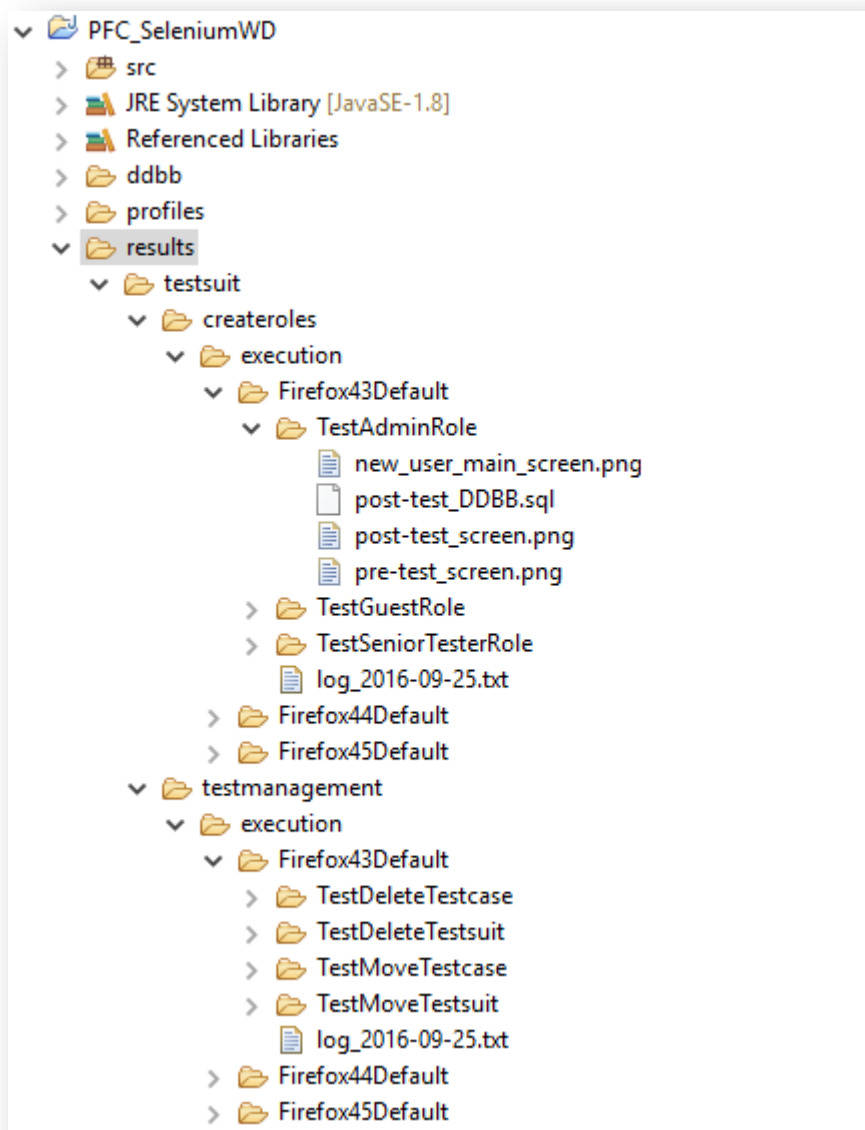
Otro ejemplo son los perfiles de Firefox, cuando lanzamos una ejecución sobre un navegador podemos indicarle el perfil que debe usar. En el perfil se guardan ciertas configuraciones y preferencias del navegador.

Para que los test tengan acceso a estos datos de entrada, en el proyecto de eclipse existen dos carpetas: **ddb** para la bases de datos y **profiles** para los perfiles del navegador.

15.3 INFORMACIÓN GENERADA POR LOS TESTCASES. DATOS DE SALIDA.

Cada test, durante su ejecución, genera una serie de datos de salida que se van registrando en unos directorios concretos. La dirección donde almacenar toda esa información la proporciona el objeto FirefoxExecution por medio de su atributo environment.

Concretamente, tras las distintas ejecuciones de los test diseñados para esta parte del PFC, los resultados obtenidos se organizan de la siguiente manera:



Para cada funcionalidad (**createroles**, **testmanagement**) aparecen los resultados agrupados por ejecución (**Firefox43Default**, **Firefox44Default**, **Firefox45Default**) y dentro de cada ejecución vemos que cada caso de prueba tiene su propio directorio donde guarda los ficheros de su resultado. Además se genera un fichero de log, el cual muestra cómo ha ido la ejecución en su conjunto (**log_2016-09-25.txt** en la captura anterior). El nombre del fichero contiene la fecha de ejecución y a continuación podemos ver un extracto de su contenido:

```
13:38:23.647 [automation.FirefoxExecution]
START execution Firefox43Default
-----

13:38:23.658 [testsuit.createroles.TestAdminRole] [START TEST]
13:38:23.658 [testsuit.createroles.TestAdminRole] *Setting database
13:38:27.891 [testsuit.createroles.TestAdminRole] *Database OK
13:38:27.894 [testsuit.createroles.TestAdminRole] *Dir result:
results/testsuit/createroles/execution/Firefox43Default/TestAdminRole

13:38:35.545 [testsuit.createroles.TestAdminRole] Functionality under test:
CREATE USERS/ROLES
13:38:35.545 [testsuit.createroles.TestAdminRole] [STEP 1]: login as admin
13:38:35.546 [UserUtils.login] *Open url -> http://localhost:8080/testlink-
1.9.11/login.php
13:38:38.118 [UserUtils.login] *User: admin
13:38:39.557 [UserUtils.login] *Password: 1234
13:38:40.147 [testsuit.createroles.TestAdminRole] [STEP 2]: click Users/Roles
menu bar option
13:38:42.413 [testsuit.createroles.TestAdminRole] *Waiting for
UserUtils.takeScreenShoot() -> pre-test_screen.png
13:38:44.211 [testsuit.createroles.TestAdminRole] [STEP 3]: click Create
button
13:38:44.419 [testsuit.createroles.TestAdminRole] [STEP 4]: set Login
13:38:45.817 [CreateRole.userDetailsSetLogin] adminPFC
13:38:45.882 [testsuit.createroles.TestAdminRole] [STEP 5]: set First Name
13:38:45.959 [CreateRole.userDetailsSetFirstName] User
13:38:46.018 [testsuit.createroles.TestAdminRole] [STEP 6]: set Last Name
...
13:38:50.887 [UserUtils.login] *User: adminPFC
13:38:52.043 [UserUtils.login] *Password: 12345
13:38:52.483 [testsuit.createroles.TestAdminRole] [STEP 15]: Check login
13:38:52.483 [UserUtils.checkLoggedUser] *check logged user -> adminPFC
13:38:54.417 [UserUtils.checkLoggedUser] *Displayed -> [OK]
13:38:54.507 [UserUtils.checkLoggedUser] *Position: (138, 3) -> [OK]
13:38:54.567 [UserUtils.checkLoggedUser] *Text: adminPFC [admin] -> [OK]
13:38:54.567 [testsuit.createroles.TestAdminRole] *Waiting for
UserUtils.takeScreenShoot() -> new_user_main_screen.png
...
13:38:57.053 [AdminRole.checkTestProjectTopics] *Platform Management -> [OK]
13:38:57.053 [testsuit.createroles.TestAdminRole] [STEP 17]: click Logout
option
13:38:57.254 [testsuit.createroles.TestAdminRole] [END TEST]: PASSED

13:38:58.063 [testsuit.createroles.TestSeniorTesterRole] [START TEST]
13:38:58.064 [testsuit.createroles.TestSeniorTesterRole] *Dir result:
results/testsuit/createroles/execution/Firefox43Default/TestSeniorTesterRole
...

13:39:53.526 [automation.FirefoxExecution]
-----
END execution
```

Este fichero aporta gran cantidad de información sobre el proceso de ejecución de los tests, como por ejemplo:

- Inicio y final de la ejecución, indicando la hora a la que ocurre y el tipo de ejecución:

```
13:38:23.647 [automation.FirefoxExecution]
START execution Firefox43Default
-----
...
13:39:53.526 [automation.FirefoxExecution]
-----
END execution
```

- Inicio y final de cada test, indicando el nombre del caso de prueba que se está ejecutando y su resultado (pasado o fallado):

```
13:38:23.658 [testsuit.createroles.TestAdminRole] [START TEST]
...
13:38:57.254 [testsuit.createroles.TestAdminRole] [END TEST]: PASSED
```

- La dirección de donde se almacenarán los resultados obtenidos de la ejecución:

```
13:38:27.894 [testsuit.createroles.TestAdminRole] *Dir result:
results/testsuite/createroles/execution/Firefox43Default/TestAdminRole
```

- La funcionalidad que se está probando:

```
13:38:35.545 [testsuit.createroles.TestAdminRole] Functionality under
test: CREATE_USERS/ROLES
```

- Los pasos que va dando el caso de prueba:

```
13:38:40.147 [testsuit.createroles.TestAdminRole] [STEP 2]: click
Users/Roles menu bar option
```

- Las diferentes comprobaciones que se van realizando:

```
13:38:54.417 [UserUtils.checkLoggedInUser] *Displayed -> [OK]
```

Toda esta información es la que ha ido registrando el objeto de tipo **TestLog** durante el proceso de ejecución (ver apartado **12.4 La clase TestLog** para más información sobre la funcionalidad de este tipo de objetos).

CAPÍTULO 16

CONCLUSIONES

En el último capítulo del PFC haremos un análisis del trabajo realizado, evaluaremos Selenium WebDriver como herramienta de automatización de pruebas funcionales y expondremos las líneas de trabajo futuras.

16.1 ANALISIS DEL TRABAJO REALIZADO.

Uno de los principales objetivos del PFC era el de presentar los conceptos básicos relacionados con la calidad de software indicando la importancia que tiene dentro del desarrollo de aplicaciones y los beneficios que puede aportar diseñar software con un alto grado de calidad. Quizás este objetivo haya sido uno de los más difíciles de alcanzar, ya que sintetizar toda la información que existe referente a la calidad y las pruebas software no es tarea fácil. Además, la calidad como concepto puede ser algo subjetiva debido a que, en función de la naturaleza del producto que estemos analizando, puede representar unas características u otras.

En general, podríamos decir que se han presentado las ideas fundamentales para entender lo que es la calidad de software y lo que persigue, describiendo además algunos tipos y técnicas de pruebas.

Referente a la automatización de pruebas con Selenium WebDriver, tema principal del PFC, primero se ha realizado un estudio del API WebDriver y posteriormente se ha diseñado un supuesto práctico donde se han implementado cierto número de pruebas automáticas con el objetivo de poder comprobar el potencial de la herramienta y evaluar, de alguna manera, la experiencia obtenida con ella.

En términos generales, la segunda parte del PFC puede servir de guía para un tester que esté interesado en iniciarse en la automatización de pruebas mediante Selenium WebDriver. Con los métodos, clases e interfaces estudiadas se puede simular la mayoría de las acciones que un usuario real puede realizar tanto con el teclado como con el ratón, cuando opera sobre una aplicación web a través de su navegador.

Por último, el supuesto práctico ha servido para enfrentarse a las dificultades que entraña el implementar casos de prueba automáticos que persiguen ciertos objetivos y que pueden aportar un valor real de cara a realizar el testeo de una aplicación.

16.2 ANALISIS DE SELENIUM WEBDRIVER COMO HERRAMIENTA DE AUTOMATIZACIÓN.

De la experiencia obtenida con Selenium podemos extraer algunas conclusiones. En general, Selenium WebDriver ofrece un API de programación muy extenso que cubre muchas de las necesidades que puede tener un tester a la hora de automatizar pruebas funcionales de aplicaciones con interfaz web. Concretamente permite:

- Simular la interacción usuario-aplicación por medio del ratón.
- Simular la interacción usuario-aplicación por medio del teclado.
- Conocer en todo momento lo que la aplicación web está mostrando. Podemos saber el contenido HTML mostrado, la posición de cada elemento web, el tamaño, el texto que contiene, etc.
- Realizar capturas de pantalla.
- Gestionar la configuración del navegador (perfil, complementos, etc.), de forma que permite controlar el entorno de ejecución de los test automáticos.
- Permite simular diferentes acciones del navegador, como por ejemplo, ir a la página anterior y posterior, actualizar el contenido de la web, maximizar y minimizar la ventana de navegación, etc.

Sin embargo, lo que Selenium WebDriver no ofrece es un framework de automatización que permita crear y ejecutar test automáticos, es decir, no proporciona un mecanismo para representar las pruebas, ejecutarlas y obtener los resultados. Desde nuestro punto de vista, esto no supone una ventaja ni un inconveniente, simplemente se trata de una característica, quedando fuera del alcance de Selenium ofrecer dicho framework. Por este motivo, ha sido necesario diseñar ciertas clases durante la tercera parte del PFC, con el objetivo de dar soporte a la definición y ejecución de los test automáticos.

A continuación, para poder sintetizar todo lo posible la valoración de Selenium WebDriver listaremos los pros y contras que, desde nuestro punto de vista, tiene esta herramienta.

Principales ventajas:

- Herramienta en auge. La comunidad que existe detrás del proyecto de Selenium cada vez es más grande y es más fácil encontrar documentación y soporte. También está cobrando bastante importancia dentro del ámbito laboral, ya que es un área de conocimiento que empieza a ser demandada por las empresas.
- API completa, documentada y disponible para varios lenguajes de programación. El API ofrecida por Selenium WebDriver permite simular cualquier acción que pueda realizar un usuario sobre una aplicación web por medio de un navegador. Además, dicho API está disponible en varios lenguajes de programación como pueden ser Java, C#, Python o Ruby.
- Existe soporte para los principales navegadores web. Selenium WebDriver ofrece una implementación específica para los navegadores Firefox, Internet Explorer, Chrome, Safari y Opera.
- Desde un punto de vista técnico, Selenium WebDriver permite desacoplar la implementación del test del entorno de ejecución. Un mismo test puede ser ejecutado sobre Firefox, Internet Explorer o cualquiera de los navegadores soportados.

Principales desventajas:

- La aplicación web sólo se prueba a través de su interfaz. Selenium WebDriver es poco efectivo para pruebas de estrés o de carga.
- No sirve para medir el tiempo de respuesta de la aplicación web. Como se ha comentado en el punto anterior, Selenium WebDriver es poco efectivo para algunos tipos de pruebas, como por ejemplo, todo las que van enfocadas a comprobar tiempos de respuesta. La propia ejecución de Selenium introduce cierto retardo lo que dificulta la toma exacta de tiempos.
- Selenium por sí solo no permite conocer la cobertura de código (cantidad de código probado) alcanzada con las pruebas. La herramienta no ofrece ningún mecanismo en este sentido.
- Requiere conocer con exactitud el código HTML que genera la aplicación web. Para la localización de los diferentes componentes de la página web, es necesario tener acceso al código HTML con el objetivo de saber a qué elemento hacer referencia. En ocasiones, es necesario conocer el nombre exacto de la etiqueta, su localización XPath, ciertos atributos, etc. Esto dificulta el mantenimiento de los test, ya que una pequeña variación en el código HTML de la aplicación web puede hacer que nuestra prueba falle, aunque se esté ofreciendo la funcionalidad deseada.

16.3 LÍNEAS DE TRABAJO FUTURAS

El estudio realizado de Selenium WebDriver durante el desarrollo del PFC representa únicamente una introducción a la herramienta de automatización. Se han quedado fuera características importantes y funcionalidades que pueden aportar mayor potencial a la hora de programar test automáticos.

A continuación citaremos algunas de las posibles líneas de trabajo futuras, con el objetivo de poder ampliar los conocimientos sobre Selenium:

- **Gestión de cookies.** Utilización de las cookies del navegador. Para más información es necesario consultar la interfaz anidada *WebDriver.Options*. Ver métodos: *addCookie()*, *deleteAllCookies()*, *deleteCookie()*, *getCookieNamed()*, *getCookies()*, etc.
- **RemoteWebDriver.** La clase RemoteWebDriver permite ejecutar test automáticos sobre navegadores que se encuentren ejecutándose en máquinas remotas. Hasta ahora, la ejecución del test y la instancia del navegador se ejecutaban en la misma máquina. Con el uso de RemoteWebDriver, por ejemplo, podríamos observar el comportamiento de la aplicación operando sobre navegadores que corren en diferentes sistemas operativos.
- **Selenium Grid.** Para controlar la ejecución distribuida de los test, Selenium ofrece la posibilidad de utilizar una configuración donde una máquina actúa como Hub y el resto como Nodos. El “Hub” recibe las instrucciones de la máquina que ejecuta el test y es el encargado de sincronizar la ejecución de las diferentes instancias de WebDriver en cada uno de los “Nodos”. De esta forma, el test está totalmente desacoplado del entorno en el que se ejecuta, permitiendo realizar pruebas en diferentes escenarios de una manera eficiente.
- **Automatización de aplicaciones en dispositivos móviles.** Herramientas basadas en Selenium WebDriver para dispositivos móviles: AndroidDriver, iPhoneDriver, iOSDriver, Selendroid o Appium.