



INSTITUTO DE EDUCACION SUPERIOR
PEDRO P. DIAZ

ALUMNO: Renzo Flores Cornejo

TURNO: Diurno

SEMESTRE: V

DESARROLLO SISTEMAS DE INFORMACIÓN



PRUEBAS E IMPLEMENTACION DE UN PROYECTO DE SOFTWARE

SELENIUM

OBJETIVO:

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:

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.

CALIDAD DE SOFTWARE

¿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".

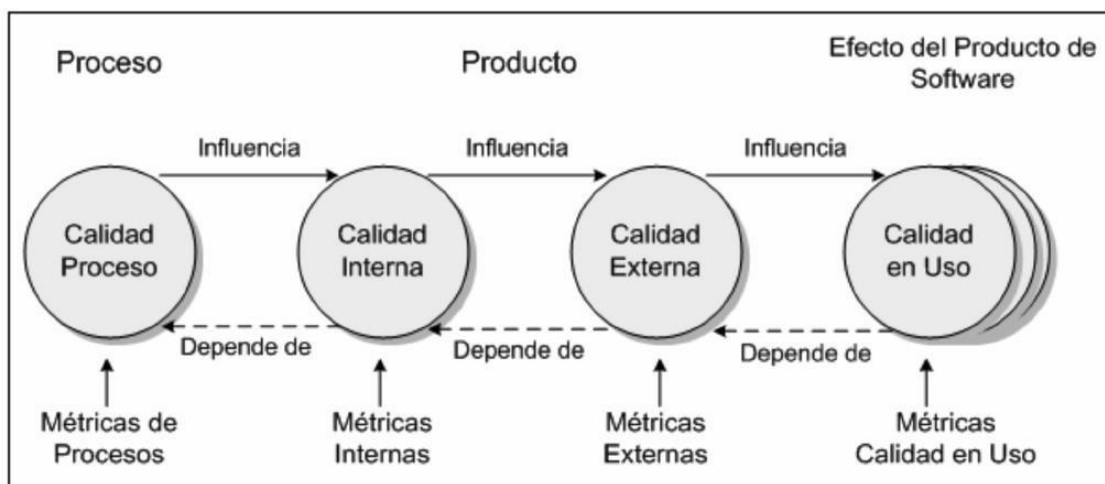
En el ámbito de la calidad de software, podemos hacer referencia a las siguientes definiciones formales:

- “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].

¿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.



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:
 - **Compleitud 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.
- **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 agrandar 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.

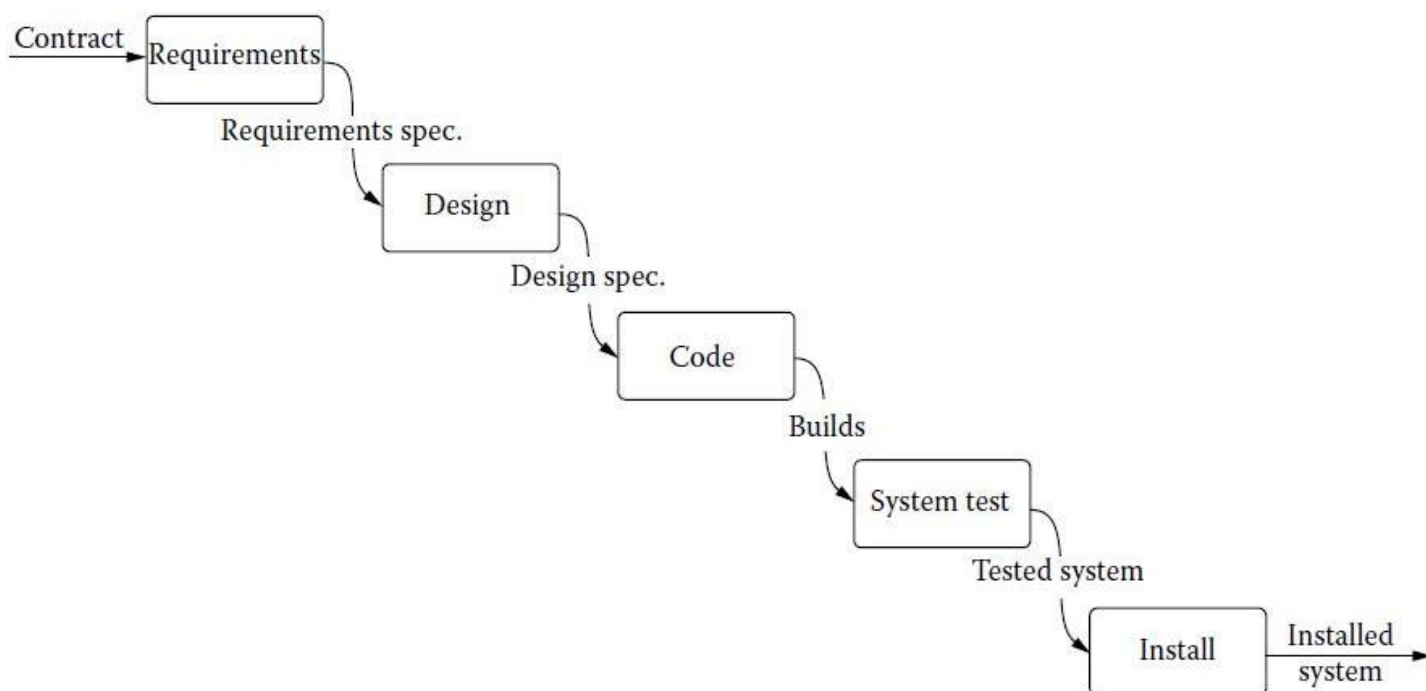
PRUEBAS SOFTWARE

INTRODUCCIÓN

Las 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.

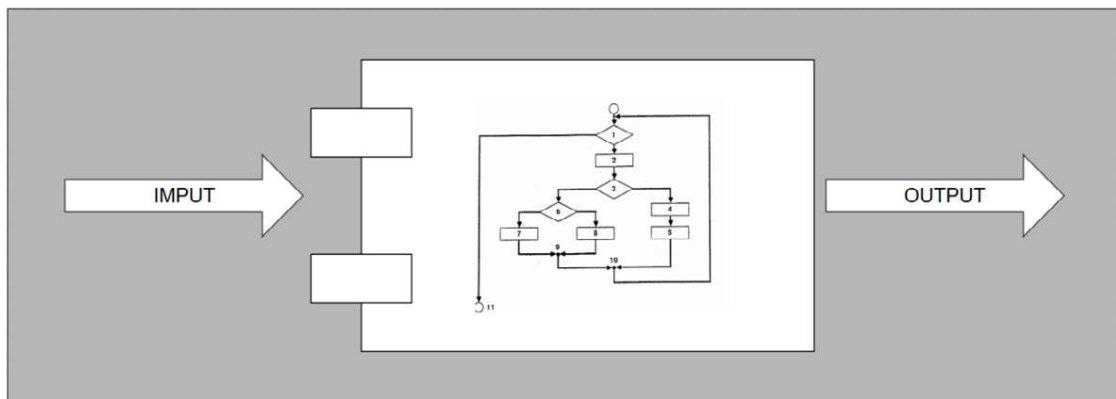


MÉTODOS Y TÉCNICAS DE PRUEBA

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.

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.

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.

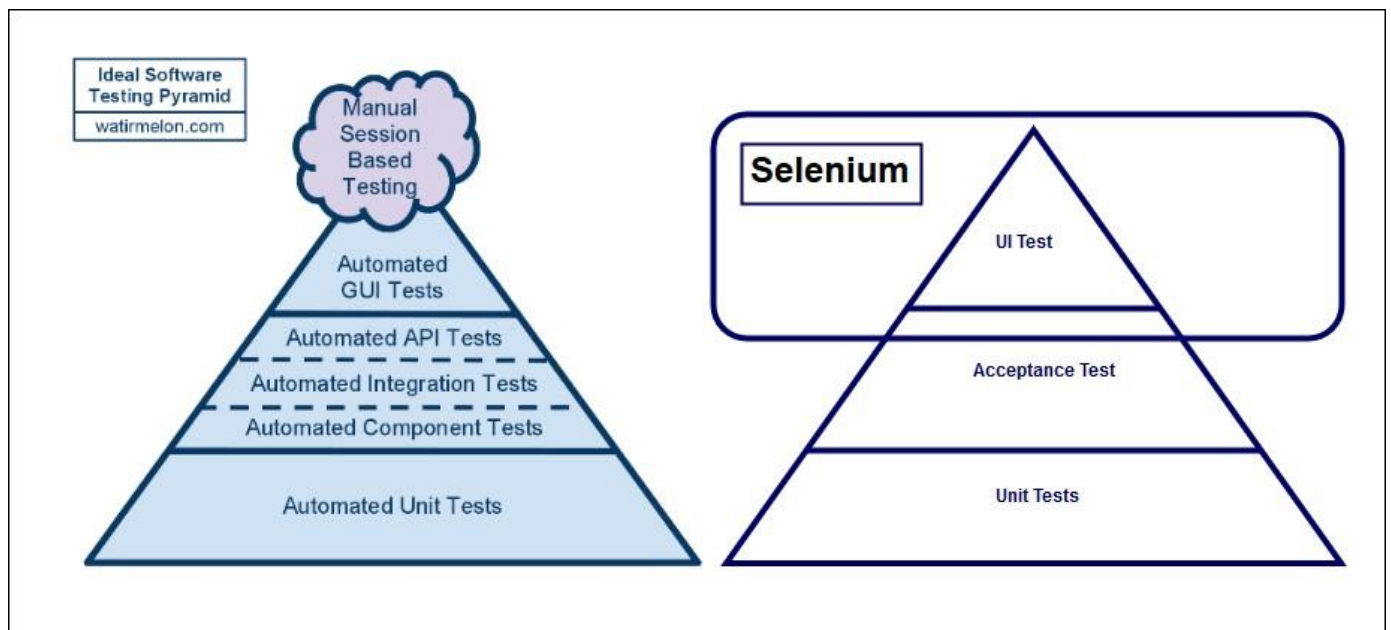
AUTOMATIZACIÓN DE PRUEBAS

¿Qué es exactamente la automatización de pruebas?

En el ámbito de las pruebas software, cuando utilizamos el término 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.

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.



SELENIUM Y SUS HERRAMIENTAS

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.

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.



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

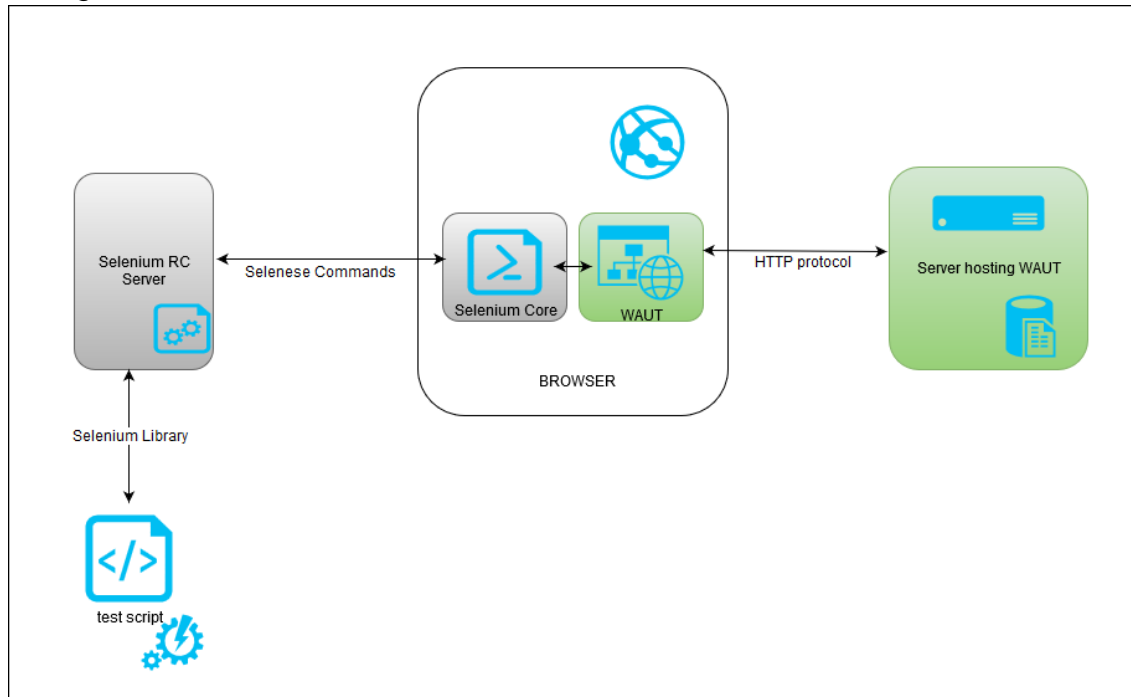
INTRODUCCION A SELENIUM

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)-

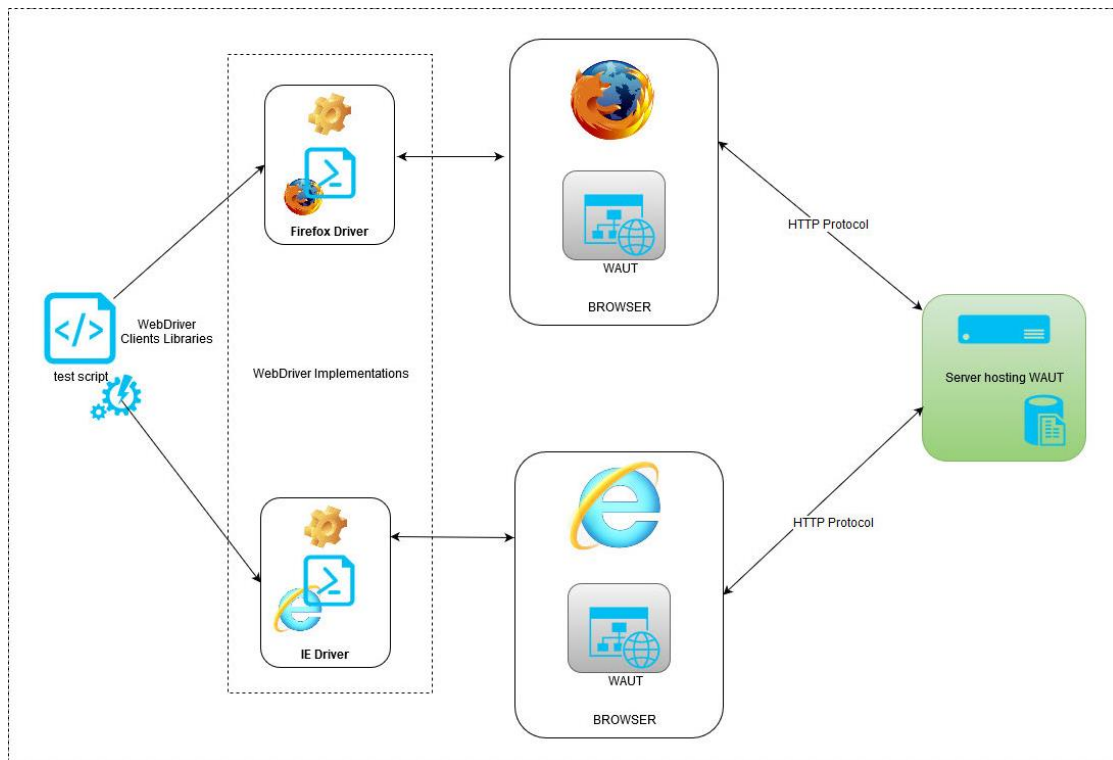
Por otro lado, Selenium RC Server es el encargado de inyectar el código de Selenium Core en el navegador y traducir las funciones de librería a comandos que intercambia con Selenium Core (Selenese Commands) para dirigir la navegación sobre una determinada WAUT.



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.



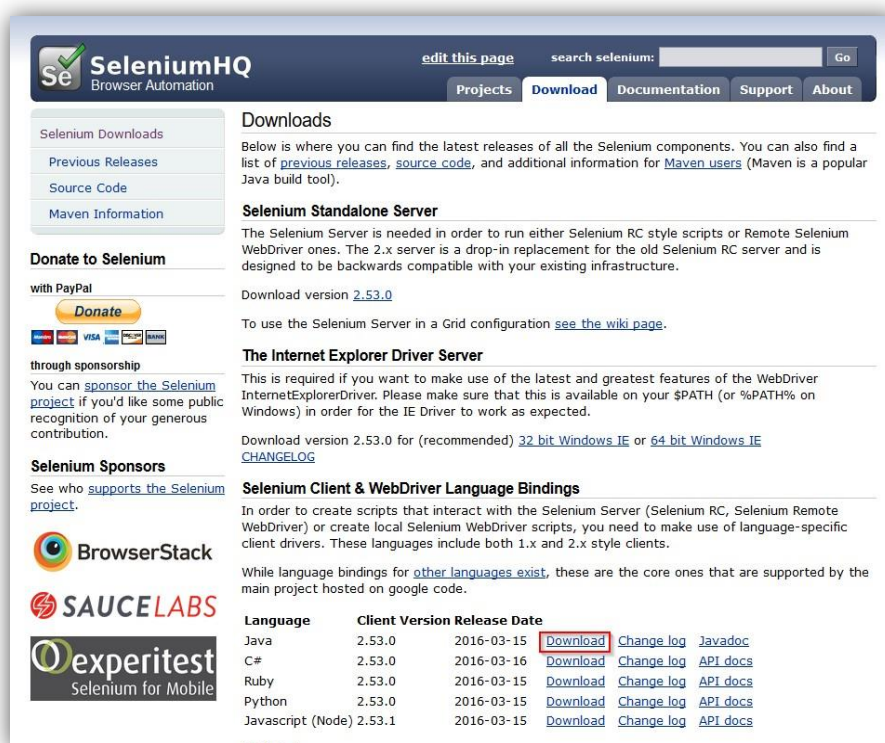
ENTORNO Y HERRAMIENTAS NECESARIAS

Para poder empezar a trabajar con Selenium, necesitamos primero descargarla 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.

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.

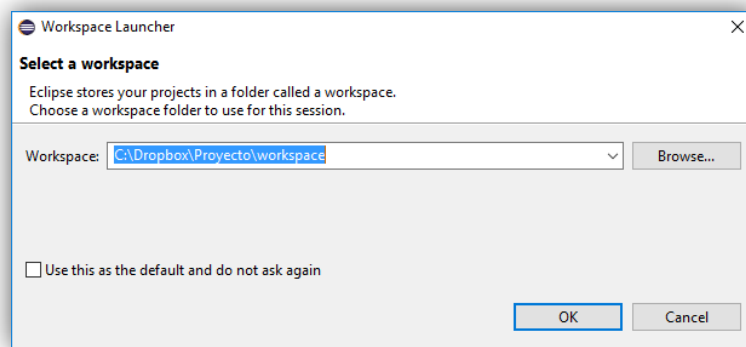
Language	Client Version	Release Date	Download	Change log	Javadoc
Java	2.53.0	2016-03-15	Download	Change log	Javadoc
C#	2.53.0	2016-03-16	Download	Change log	API docs
Ruby	2.53.0	2016-03-15	Download	Change log	API docs
Python	2.53.0	2016-03-15	Download	Change log	API docs
Javascript (Node)	2.53.1	2016-03-15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

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.

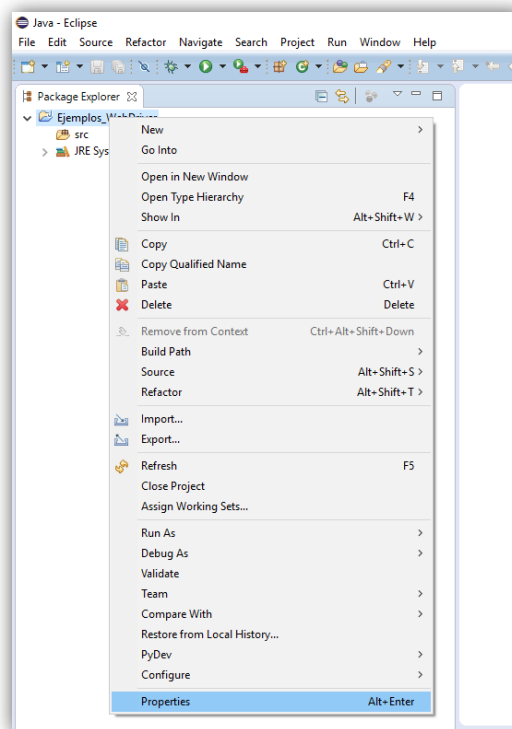
Desde la URL < <https://eclipse.org/downloads/> > podemos descargar la versión que necesitamos. En nuestro caso se ha optado por utilizar la versión Mars 1, del paquete Eclipse IDE for Java Developers.

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.



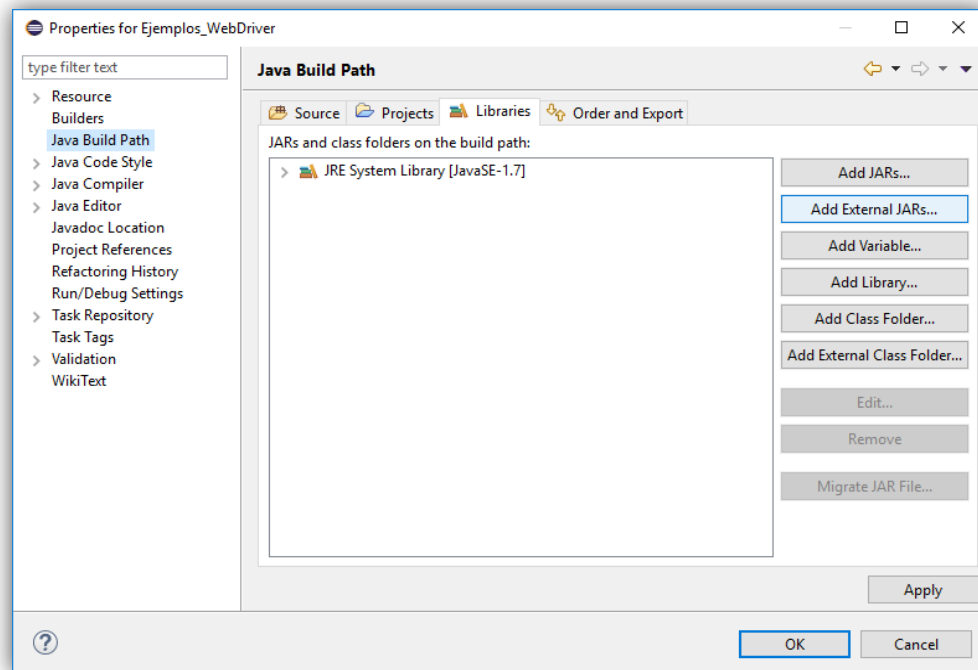
Con la perspectiva Java activa, crearemos un nuevo proyecto desde la barra de menú superior.

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.



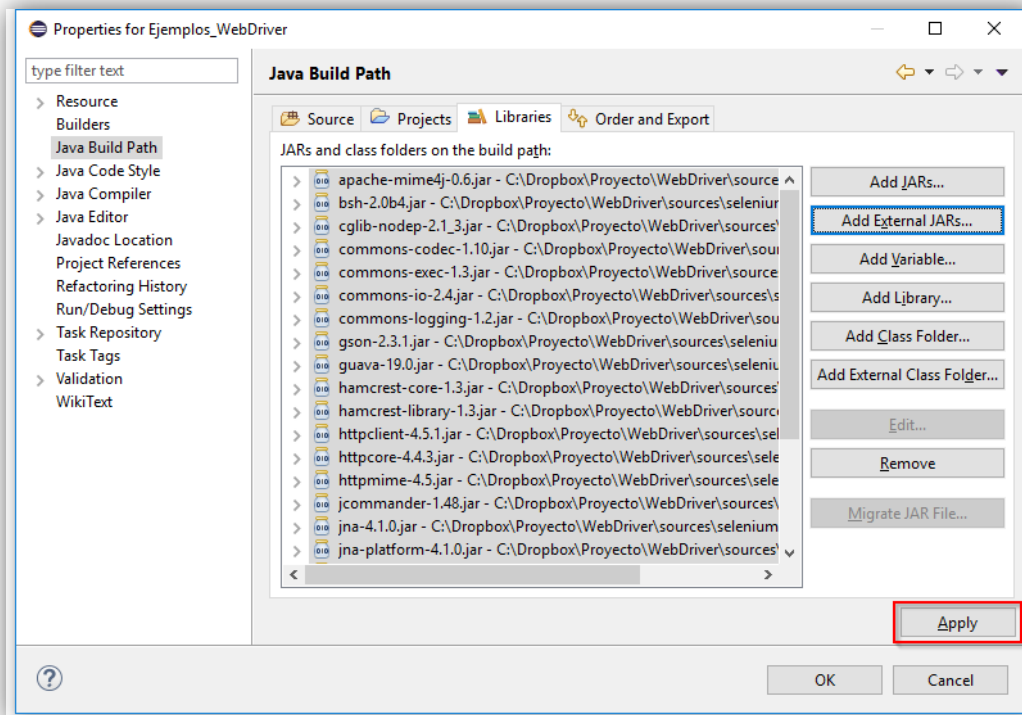
Dentro de la pantalla de propiedades, en el menú derecho, seleccionamos la opción "Java Build Path".

AHORA, EN LA PESTAÑA “LIBRARIES”,
PULSAMOS EL BOTÓN DERECHO “ADD
EXTERNALJARS...”



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

PULSANDO EL BOTÓN “APPLY” TERMINAMOS ESTE PROCESO.



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:

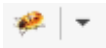
- < <https://addons.mozilla.org/es/firefox/addon/firebug/> >

Y FIREPATH DE:

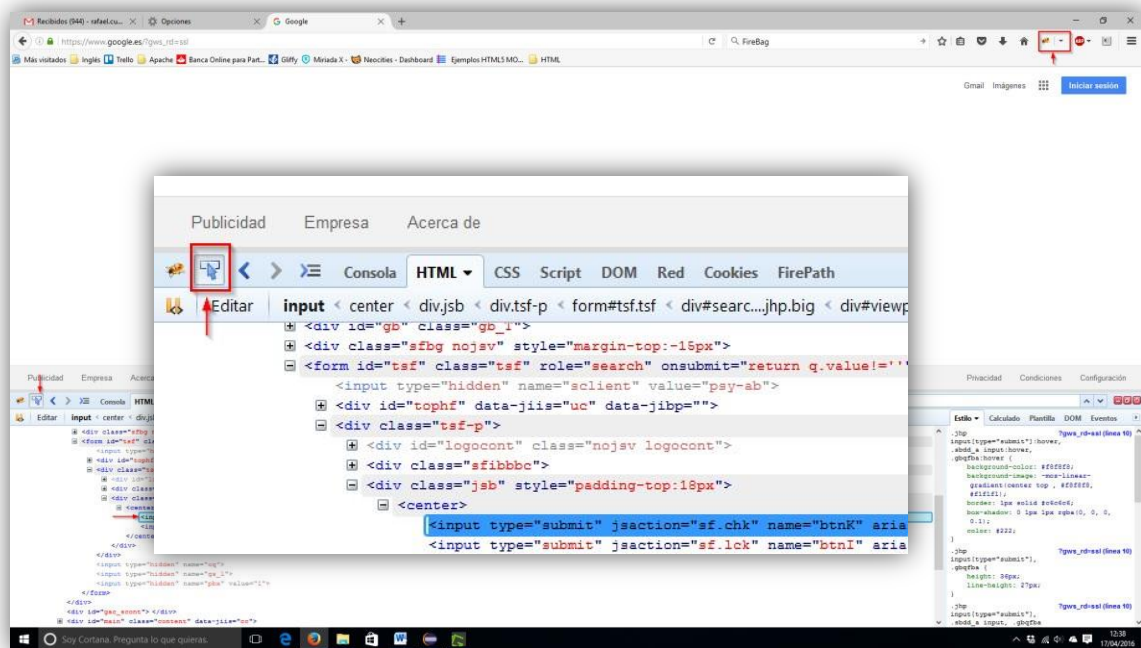
- < <https://addons.mozilla.org/es/firefox/addon/firepath/> >.

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”.

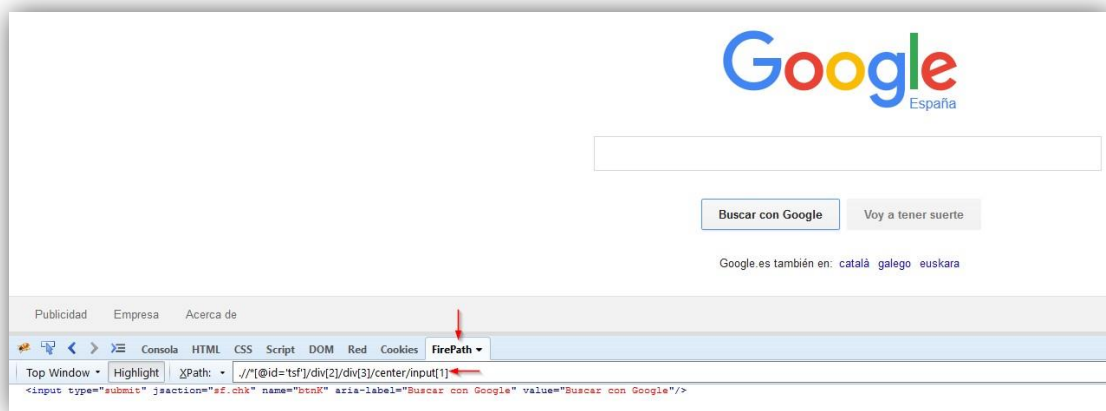


Para ello, después de activar el complemento, pulsando sobre el icono del insecto en el menú de la ventana inferior, debemos seleccionar la función de localización.

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.