

# Trabajo de Investigación: Pruebas de regresión

Chavarria Peña Jonathan Andrés

Estudiante Ing. en Sistemas de Computación

Universidad Fidélitas

San José, Costa Rica

[jonach1998@gmail.com](mailto:jonach1998@gmail.com)

Phillips Tencio Edmond

Estudiante Ing. en Sistemas de Computación

Universidad Fidélitas

Alajuela, Costa Rica

[ephillips10986@ufide.ac](mailto:ephillips10986@ufide.ac)

Morales Cordero Valeria

Estudiante Ing. en Sistemas de Computación

Universidad Fidélitas

San José, Costa Rica

[valemc0603@gmail.com](mailto:valemc0603@gmail.com)

Sánchez Camacho Carlos Daniel

Estudiante Ing. en Sistemas de Computación

Universidad Fidélitas

San José, Costa Rica

[csanchez20965@ufide.ac](mailto:csanchez20965@ufide.ac)

**Resumen—**In this investigation we found out important and relevant information about regression tests, and all the characteristics in this method like regression tools, the meaning, the functions, advantages, and disadvantages and most important how to develop this method. We studied this method to acknowledge the regression tests to be able to know how we will be able to use it in our projects and professional life. This is an important method because it helps us to verify a good base project, with minimum to no errors.

## 1. DESARROLLO

### 1.1. Qué es una Prueba de Regresión

Son pruebas para determinar si las aplicaciones existentes aún pueden funcionar como se espera después de que se hayan actualizado o modificado. Cada cambio pueden interrumpir la funcionalidad del software.

No se deben cambiar las pruebas de regresión hasta que las pruebas actuales pasen. Una fail en una prueba de regresión significa que una nueva funcionalidad ha afectado a otra que era correcta en el pasado.

Como también una falla en el test podría indicar que se ha vuelto a producir un error que ya había sido resuelto en el pasado.

**1.1.1. ¿Qué son las pruebas de software?:** Las pruebas del software pueden definirse como una actividad donde un sistema o una parte de un sistema es ejecutado de forma controlada es decir bajo unas condiciones específicas para observar su comportamiento y registrarlo.

**1.1.2. ¿Qué es validar un sistema? :** Validar un sistema es un proceso por el cual se evalúa una parte o la totalidad de un sistema para determinar si satisface todos los requisitos especificados, un sistema puede ser validado durante el desarrollo o al final del mismo.

**1.1.3. ¿Qué características tiene que tener un software de calidad? :** Los atributos de un buen software son: mantenibilidad, confiabilidad, eficiencia y usabilidad.

**1.1.4. Testing:** Verificación dinámica del comportamiento de un programa usando un conjunto finito de casos de prueba seleccionados desde el dominio infinito de ejecución contra el comportamiento esperado.

**1.1.5. ¿Qué es un tester?:** Un tester o probador es una persona que se encarga de realizar las pruebas de un sistema, por norma general suelen poseer buenas habilidades de desarrollo y codificación, conocimiento de algoritmos y lenguajes formales.

**1.1.6. Técnicas de testing:** Una técnica de testing provee distintos criterios para seleccionar el conjunto de casos de prueba que serán usados para visitar el software

**1.1.7. Tipos de pruebas:** Existen varios tipos de prueba, como por ejemplo: pruebas de aceptación, pruebas de desempeño, pruebas funcionales, pruebas del sistema, pruebas de caja blanca, pruebas de caja negra, pruebas de comportamiento y pruebas de regresión; en nuestro caso nos enfocaremos en las pruebas de regresión.

**1.1.8. Mantenimiento del software :** El mantenimiento del software implica modificarlo como resultado de errores o de alteraciones en los requerimientos del usuario, durante tarden las modificaciones nuevos errores se pueden introducir causando efectos secundarios adversos en el software. Entre los mantenimientos más comunes se encuentran la perspectiva que es aquella que mejora la funcionalidad del software, y la correctiva que es aquella que detecta y corrige defectos. Ya sea en la perspectiva o correctiva se deben asegurar que los cambios no afectan inadvertidamente las funcionalidades no modificadas, cuando esto ocurre se dice que hay un error de regresión.

**1.1.9. Objetivo de las pruebas de regresión:** El objetivo de las pruebas de regresión es verificar la no regresión de la calidad luego de un cambio, además asegurar que los cambios no introducen un comportamiento no deseado o errores adicionales que implican la re ejecución de algunas o todas las pruebas realizadas.

**1.1.10. Pruebas de regresión:** Es una tarea necesaria pero cara, el test de regresión es efectuado sobre un programa modificado para confirmar que los cambios son correctos y no han afectado de forma adversa en las partes no alteradas del programa, una de las estrategias del test de regresión es el modelo “retest all” que vuelve ejecutar todos los test, pero esta estrategia puede consumir tiempo y recursos de modo excesivo.

**1.1.11. Técnicas de selección:** Las técnicas de selección de test de regresión intentan reducir el tiempo requerido para volver a probar un programa modificado seleccionando algún conjunto del total de los tests existentes, por lo tanto, el testing de regresión tiene como objetivo principal proporcionar la suficiente confianza en que las modificaciones efectuadas están correctas y no han afectado otras partes del software.

**1.1.12. Testing de regresión:** El testing de regresión puede ser progresivo o correctivo, para el testing de regresión sería costoso repetir todo el conjunto de casos de prueba usados en el desarrollo inicial del software y no sería óptimo elegir un subconjunto al azar de estos casos de prueba, por lo tanto, es importante seleccionar un subconjunto conveniente de los casos de prueba que logren los objetivos del testing de revisión.

**1.1.13. Selección:** La selección de los casos convenientes se puede hacer de diversas maneras y se han propuesto un número de acercamientos y de algoritmos de testing de regresión. El testing de regresión tiene importancia crucial a nivel de pruebas de unidad, integración y sistema.

**1.1.14. Clasificación de las técnicas:** Algunas técnicas seleccionan pruebas basadas en información recolectada de especificaciones del programa, la mayoría lo hace basadas en información acerca del código del programa y de la versión modificada, estas técnicas basadas en el código persiguen tres metas diferentes:

- Técnicas de cobertura: localizan componentes del programa que han sido modificados o afectados por modificaciones, y seleccionan pruebas del conjunto original de pruebas que ejercitan esos componentes.
- Técnicas de minimización: trabajan como las de cobertura, pero seleccionan conjuntos mínimos de pruebas a través de componentes del programa modificados o afectados.
- Técnicas seguras: seleccionan cada prueba en el conjunto original de pruebas que pueden exponer una o más fallas de mejora.

**1.1.15. Importancia de las pruebas de regresión:** Las pruebas de regresión se realizan generalmente después de la corrección de un defecto o después de la adición de nuevas funcionalidades, su objetivo es asegurar que ningún defecto se añadió al sistema después de la modificación; las llamamos nuevas de regresión porque tenemos que hacer nuevas pruebas donde ya se han realizado antes, por lo general este tipo de pruebas se realizan a través de herramientas automatizadas, pues muchas veces hay falta de tiempo para volver a ejecutar dichas pruebas así éstas se dejan en segundo plano. En las pruebas de regresión a veces se pueden encontrar más defectos que la primera vez, esto es debido a que el tester tiene más familiaridad con el sistema y al volver a ejecutar los casos de prueba es posible detectar algunos defectos que en la primera ejecución pasaron inadvertidos.

**1.1.16. Plan de casos de prueba:** El plan de casos de prueba de regresión se puede clasificar en tres tipos: los casos de prueba que descubren toda la funcionalidad del sistema, los casos de prueba sólo para las características que han sido modificadas, y nuevos casos de prueba para las características que se vieron afectadas probablemente por el cambio. Las pruebas de regresión son una forma efectiva de reducir la cantidad de defectos que se pueden encontrar en un sistema.

La etapa de mantenimiento en todo proceso de desarrollo de software es una fase que ocupa la mayor parte del proceso y es la etapa en donde el software puede sufrir muchos cambios debido a nuevos requerimientos, cambios de plataformas o dispositivos empleados por defectos encontrados; por lo tanto, es crucial poseer un mecanismo de prueba para validar que todos los cambios introducidos al software no hayan perjudicado las funcionalidades existentes, de allí que el testing de regresión sea una actividad muy importante en la etapa de mantenimiento. El testing de regresión posee una base que lo diferencia de los demás tipos de testing, este tiene como fundamento la existencia de un conjunto original en casos de pruebas correspondientes al software no modificado.

El testing de regresión es una tarea costosa pero necesaria la mayoría de las veces no se realizan por problemas como falta de recursos y tiempo, el testing de regresión permite mantener la calidad de las aplicaciones; la técnica más segura para encontrar defectos de regresión es el método de volver a probar todo, es decir, probar la aplicación modificada con el conjunto original de casos de prueba, esta es una técnica segura y muy cara además

es ineficiente debido a que en una modificación no todo el código cambia sólo una parte y por lo tanto sólo habría que probar dicho sector. Las métricas más empleadas para medir la efectividad de las técnicas del testing de regresión son la inclusividad, precisión, eficiencia y la generalidad.

### 1.2. ¿Cuándo debe realizarse la prueba de regresión?

Es importante ejecutar este tipo de pruebas cada vez que cambia su código. Los cambios de aplicaciones que requieren pruebas de regresión son:

1. Mejoras.
2. Parches.
3. Cambios de configuración.
4. Integración con otro software.
5. Un problema de rendimiento.
6. Hay un cambio en los requisitos y el código se modifica de acuerdo con el requisito.

Es necesario asegurarse de que las funciones del código antiguo puedan seguir funcionando normalmente después de que se incluya el nuevo código.

Se debe tener en consideración varios aspectos a la hora de implementar las pruebas de regresión:

- Los cambios que se implementaron en el código y cuales otras partes son afectadas debido a esos cambios.
- Conocer la cantidad de casos que se realizan en las pruebas regresión.
- Seleccionar los casos necesarios para comprobar el buen funcionamiento del programa.
- Si son muchos casos los que se necesitan realizar considerar si lo recomendado son las pruebas automáticas o las manuales.

Es necesario especificar que toda modificación, por mínima que sea, necesitará de pruebas de regresión. Para determinar la cantidad de casos en las pruebas de regresión a realizar se debe saber que tanto riesgo de error corre el programa después de realizados los cambios o actualizaciones, si el riesgo es alto hay mayor necesidad de realizar las pruebas, si es riesgo es bajo o nulo se podrían realizar de igual forma o hacer menos cantidad de casos.

Es importante tener en consideración que en el más pequeño de los cambios el riesgo puede ser alto, por lo mismo es importante estudiar el impacto de cada cambio. Es importante saber que lo primero que se debe realizar es el retesteo de todos los casos de la prueba de regresión, posterior a esto es necesario priorizar los casos en específico que tienen modificaciones.

### 1.3. ¿Cómo hacer una prueba de regresión?

Se puede realizar de distintas formas, depende de los cambios realizados y de que tanta confiabilidad se necesita en los datos de la prueba realizada.

- Realizar pruebas a todo el código: Es la forma de garantizar que todos los casos de prueba en el programa sean realizados para su comprobar su integridad y funcionamiento. Este método suele ser costoso ya que a menudo requiere una gran inversión de tiempo y recursos.
- Realizar las pruebas en una parte específica del código: Permite seleccionar una parte específica de las pruebas que se ejecutaban, por lo tanto, no es tan costosa.
- Priorización de casos de prueba: Prioriza los casos de prueba de acuerdo con su impacto normalmente se prueban aspectos críticos y funcionalidades de uso frecuente.

#### 1.4. Software a utilizar

```
@echo off
python -c "import selenium"
if ERRORLEVEL 1 (pip check selenium & pip install selenium)

python -c "import webdriver_manager"
if ERRORLEVEL 1 (pip check webdriver-manager & pip install
    webdriver-manager)

python -c "import HtmlTestRunner"
if ERRORLEVEL 1 (pip check html-testRunner & pip install html-
    testRunner)

if ERRORLEVEL 0 (echo Done) else (echo Some errors, please check
    )
```

#### 1.5. Herramienta para las pruebas

Para poder realizar el trabajo de una prueba de regresión en algún proyecto de revisión de la calidad del software existen herramientas que nos facilitan un poco el trabajo, estos programas o softwares nos ayudan a hacer las pruebas de regresión tanto automáticas como manuales facilitando este trabajo. Si bien los usuarios tienen la opción de utilizar herramientas gratuitas de prueba de regresión de código abierto, la mayoría de los expertos en tecnología argumentan que es prudente adoptar una visión a más largo plazo y considerar las ventajas de adoptar una herramienta de prueba comercial y de esta manera que sea una versión más completa y confiable. Ver 1

**1.5.1. Ejemplos de herramientas:** Según se dice en investigaciones la herramienta Selenio es una de las herramientas más usadas en la actualidad, pero poco a poco siendo más cuestionada por los usuarios ya que se dice que está siendo superada por otras herramientas tales como:

- Testingwhiz: Herramienta de prueba de código automática y fácil de usar. ver 2
- Sahi Pro: Es una herramienta de automatización de pruebas única que se centra en el evaluador y, al mismo tiempo, se basa en el diseño y la funcionalidad de los negocios ver 3
- Silk Test: Herramienta de la empresa Micro Focus centrada en las pruebas de regresión tanto para desarrollo web como para desarrollo móvil.
- Junit: Es un entorno de pruebas para Java creado por Erich Gamma y Kent Beck. Se encuentra basado en SUnit creado originalmente para realizar pruebas unitarias para el lenguaje Smalltalk.
- Unit tests: Este tipo de tests consiste en probar de forma individual las funciones o métodos. Generalmente son pruebas automatizadas de menor costo, y pueden ejecutarse rápidamente por un servidor de integración continua.

Entre otras.

#### 1.5.2. Ventajas y Desventajas de las pruebas de regresión:

Como en muchos temas los criterios buenos o malos deben de existir y este caso no es una excepción, en este caso también existen ventajas que hacen que este método sea muy utilizado y las desventajas para tomar en cuenta.

##### 1. Ventajas

Algunas de las ventajas del método de pruebas de regresión son los siguientes:

- 1.1 Primeramente, nos permite asegurarnos de que se pueda hacer cualquier corrección o cambio de código en un módulo o en la aplicación sin afectar el resto de código que ya fue probado exitosamente.
- 2.2 Ayudan a mejorar el producto que vamos a entregar al cliente ya que nos aseguramos de que el código o la aplicación funcione correctamente.

- 3.3 Estas pruebas de regresión se pueden hacer utilizando herramientas de automatización haciendo el trabajo más fácil y asegurándonos que se esté haciendo correctamente.
- 4.4 Se pueden identificar fácilmente los errores que se pudieran dar por algún cambio o actualización.
- 5.5 Asegura que todos los errores que se encontraron previamente no sean creables nuevamente.
- 6.6 Durante las pruebas de regresión, los casos de prueba se priorizan según los cambios realizados en la función o el módulo de la aplicación. La característica o módulo donde se realizan los cambios o modificaciones que toda la característica se toma en prioridad para la prueba.

##### 2. Desventajas

- 2.1 Si las pruebas de regresión no se realizan con una herramienta de automatización se pueden volver muy tediosas ya que se tienen que hacer las pruebas una y otra vez.
- 2.2 Se tiene que hacer una prueba de regresión con cualquier cambio que se realice no importa que tan pequeño sea el cambio igual puede generar errores en el código y complicar el proyecto.
- 2.3 Las herramientas de automatización de pruebas de regresión se recomiendan utilizar las más completas por lo que en algunos casos las herramientas tienden a ser bastante costosas dependiendo de lo que se quiera.

#### 1.6. Prueba de regresión utilizando framework de Selenium

En el siguiente apartado se mostrará un ejemplo de una prueba de regresión.

Es importante actualizar las dependencias del lenguaje a utilizar; en este ejemplo las dependencias a utilizar se establecieron en un archivos de texto sin formato, guardados con la extensión .BAT que contienen un conjunto de instrucciones que se utilizaran con el cmd de Windows.

```
C:\git\calidad_software_proyecto\
    ↪ Investigacion_1_Puebas_de_Regresion\Ejemplo_Practico>
    ↪ python_dependencies.bat

C:\git\calidad_software_proyecto\
    ↪ Investigacion_1_Puebas_de_Regresion\Ejemplo_Practico>
    ↪ python -c "import selenium"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'selenium'

C:\git\calidad_software_proyecto\
    ↪ Investigacion_1_Puebas_de_Regresion\Ejemplo_Practico>if
    ↪ ERRORLEVEL 1 (pip check selenium & pip install selenium
    ↪ )
No broken requirements found.
Collecting selenium
  Using cached selenium-3.141.0-py2.py3-none-any.whl (904 kB)
Requirement already satisfied: urllib3 in c:\python39\lib\site-
    ↪ packages (from selenium) (1.26.5)
Installing collected packages: selenium
Successfully installed selenium-3.141.0

C:\git\calidad_software_proyecto\
    ↪ Investigacion_1_Puebas_de_Regresion\Ejemplo_Practico>
    ↪ python -c "import webdriver_manager"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'webdriver_manager'

C:\git\calidad_software_proyecto\
    ↪ Investigacion_1_Puebas_de_Regresion\Ejemplo_Practico>if
    ↪ ERRORLEVEL 1 (pip check webdriver-manager & pip install
    ↪ webdriver-manager )
No broken requirements found.
Collecting webdriver-manager
  Using cached webdriver_manager-3.4.2-py2.py3-none-any.whl (16
    ↪ kB)
Requirement already satisfied: crayons in c:\python39\lib\site-
    ↪ packages (from webdriver-manager) (0.4.0)
```

```

Requirement already satisfied: requests in c:\python39\lib\site-
    ↪ packages (from webdriver-manager) (2.25.1)
Requirement already satisfied: configparser in c:\python39\lib\
    ↪ site-packages (from webdriver-manager) (5.0.2)
Requirement already satisfied: colorama in c:\python39\lib\site-
    ↪ packages (from crayons->webdriver-manager) (0.4.4)
Requirement already satisfied: idna<3,>=2.5 in c:\python39\lib\
    ↪ site-packages (from requests->webdriver-manager) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\
    ↪ python39\lib\site-packages (from requests->webdriver-
    ↪ manager) (1.26.5)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\python39\
    ↪ lib\site-packages (from requests->webdriver-manager)
    ↪ (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\python39\
    ↪ \lib\site-packages (from requests->webdriver-manager)
    ↪ (2020.12.5)
Installing collected packages: webdriver-manager
Successfully installed webdriver-manager-3.4.2

C:\git\calidad_software_proyecto>
    ↪ Investigacion_1_Pruebas_de_Regresion\Ejemplo_Practico>
    ↪ python -c "import HtmlTestRunner"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'HtmlTestRunner'

C:\git\calidad_software_proyecto>
    ↪ Investigacion_1_Pruebas_de_Regresion\Ejemplo_Practico>if
    ↪ ERRORLEVEL 1 (pip check html-testRunner & pip install
    ↪ html-testRunner )
No broken requirements found.
Collecting html-testRunner
  Using cached html_testRunner-1.2.1-py2.py3-none-any.whl (11 kB
    ↪ )
Requirement already satisfied: Jinja2>=2.10.1 in c:\python39\lib\
    ↪ \site-packages (from html-testRunner) (3.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\python39\
    ↪ lib\site-packages (from Jinja2>=2.10.1->html-testRunner)
    ↪ (2.0.1)
Installing collected packages: html-testRunner
Successfully installed html-testRunner-1.2.1

C:\git\calidad_software_proyecto>
    ↪ Investigacion_1_Pruebas_de_Regresion\Ejemplo_Practico>if
    ↪ ERRORLEVEL 0 (echo Done ) else (echo Some errors, please
    ↪ check )
Done
C:\git\calidad_software_proyecto>
    ↪ Investigacion_1_Pruebas_de_Regresion\Ejemplo_Practico>
\end{lstlisting}

Estas dependencias son para utilizar las siguientes librerías:

\begin{lstlisting}[language=Python]

from unittest import TestSuite, makeSuite
from unit_tests.GoogleSearch import GoogleSearch
from HtmlTestRunner import HTMLTestRunner

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import WebDriverWait
from unittest import TestCase
from time import sleep

```

En la prueba de regresión se cuenta con un caso a testear, este abrirá Google, buscara Netflix en la barra de búsqueda y enviará un ENTER para comenzar la búsqueda. Posteriormente con el mouse dará click a la opción NETFLIX COSTA RICA, esta opción deberá concordar con la establecida en el script.

```

def setUp(self):
    self.driver = webdriver.Chrome(
        ↪ ChromeDriverManager().install()
        ↪ )
    self.driver.implicitly_wait(10)
    self.driver.maximize_window()
    self.wait = WebDriverWait(self.driver
        ↪ , 10)

```

```

def test_search_netflix(self):
    self.driver.get('https://google.com/'
        ↪ )
    self.driver.find_element_by_name('q')
        ↪ .send_keys('Netflix' + Keys.
        ↪ RETURN)
    self.driver.find_element_by_xpath("//
        ↪ h3[contains(., 'Netflix - Costa
        ↪ Rica')]").click()
    sleep(1)
    self.assertEqual(self.driver.title ,
        ↪ 'Netflix - Costa Rica - Ve series
        ↪ online, ve películas online')

```

```

def tearDown(self):
    self.driver.close()
    self.driver.quit()
    print('Test Completed')

```

Al correr el script este hará todo de manera automática, a continuación se mostrara cada pantalla que este abre para realizar el test.

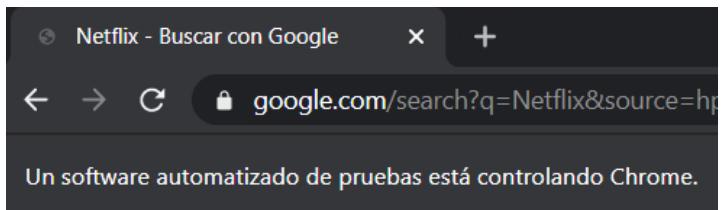


Figura 1. Test abriendo Google y corriendo automáticamente



Figura 2. Abre Google

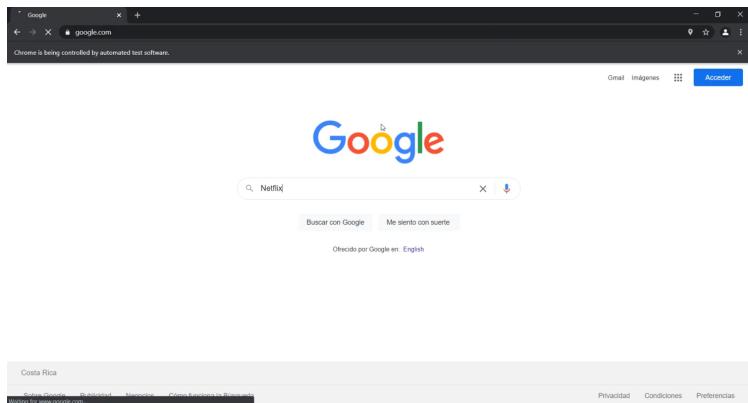


Figura 3. Escribe en el buscador lo deseado

```

<!DOCTYPE html>
<html>
<head>
    <title>Unittest Results</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" integrity="...>
</head>
<body>
    <div class="container">
        <div>Test failed: TestResults_unit_tests.GoogleSearch.googleSearch</div>
        <div>Run 1 test in 12.529s</div>
        <div>Run 1 test in 12.529s</div>
        <div>GoogleSearch</div>
        <div>GoogleSearch</div>
        <div>net_search.netflix</div>
        <div>net_search.netflix</div>
    </div>
</body>

```

Figura 6. Test cuando falla

Cuando el test pasa, este genera un reporte con la información de la prueba. Dentro de la información esta la duración del test, si este pasa y a la hora en la que inicia.

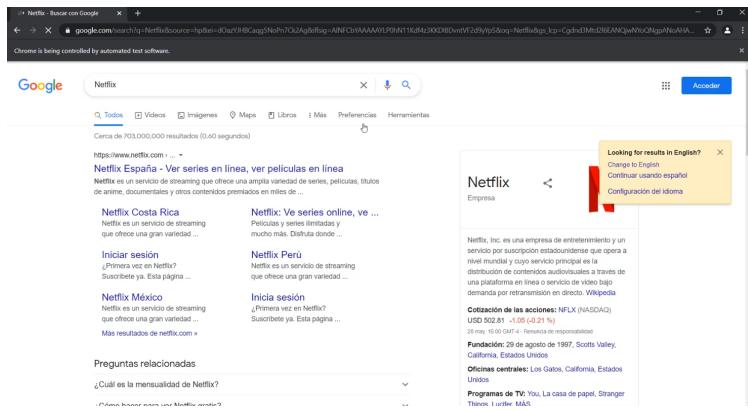


Figura 4. Da click en la pagina deseada

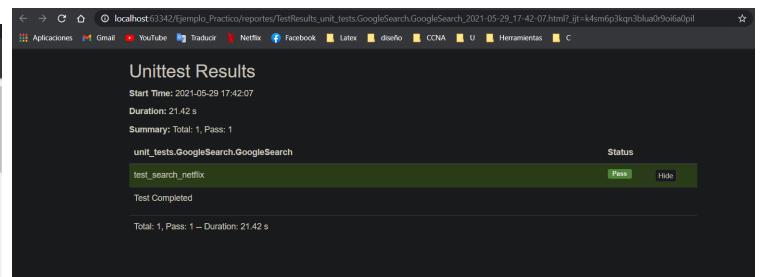


Figura 7. Test cuando pasa

## 1.7. Prueba de regresión utilizando Selenium IDE

Para poder utilizar esta opción, lo primero que se debe realizar es la descarga del IDE de Selenium, esta es una extensión de Google Chrome.

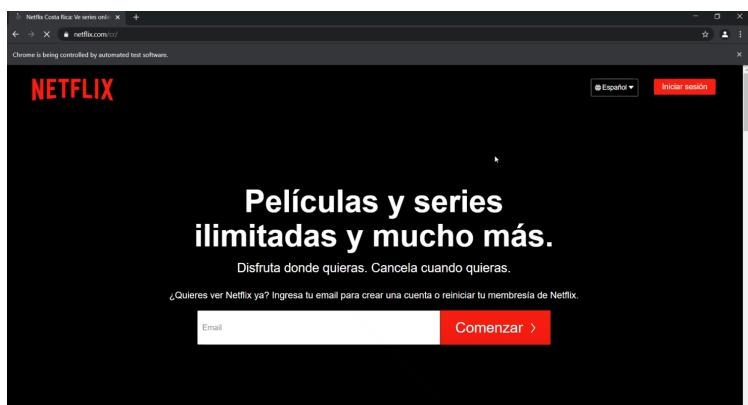


Figura 5. Abre la pagina y comprueba que el título concuerde

Si la prueba falla, se muestra como un error normal y se indica el porque del fallo, en esta ocasión el titulo de la pagina no coincide con el que se estipula en el script.

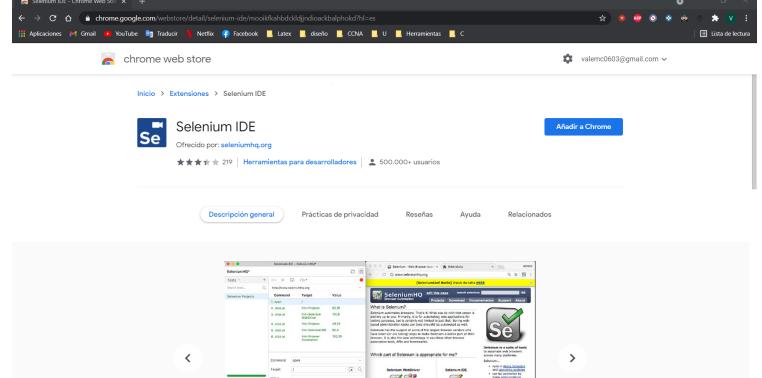


Figura 8. Selenium IDE

Esta herramienta permite realizar pruebas de regresión a páginas web sin tener que escribir código. En este ejemplo se probará que se pueda ingresar a la página de Facebook de Netflix.

Este IDE registra las interacciones con sitios web para ayudar a generar y mantener la automatización del sitio, realizar pruebas y eliminar la necesidad de realizar tomas repetitivas manualmente.

La siguiente imagen es de la línea de comandos que debe realizar el test, cuando alguno de estos falla se pone en color rojo la linea que fallo.

Figura 9. Línea de comandos

Figura 10. Error en la línea de comandos

El IDE muestra cuales comandos logra completar y cuales no, los muestra conforme lo va realizando. En este ejemplo el tiempo de espera fue muy largo y no se pudo encontrar la pagina de Netflix - Home — Facebook.

**Running 'netflix'**

1. open on / OK
2. type on name=q with value netflix OK
3. sendKeys on name=q with value \${KEY\_ENTER} OK
4. runScript on window.scrollTo(0,1287.199951171875) OK
5. Trying to find xpath=//h3[contains(.,'Netflix - Home | Facebook')]]... Failed: Implicit Wait timed out after 30000ms

**'netflix' ended with 1 error(s)**

Figura 11. Corrida en curso

Ejemplo de cuando el test logra completar todos los comandos establecidos.

Figura 12. Test cuando pasa

### Running 'netflix'

1. open on / OK
  2. type on name=q with value netflix OK
  3. sendKeys on name=q with value \${KEY\_ENTER} OK
  4. runScript on window.scrollTo(0,1287.199951171875) OK
  5. click on xpath=//h3[contains(.,'Netflix - Home | Facebook')]] OK
  6. assertEquals on Netflix | Facebook OK
  7. close OK
- 'netflix' completed successfully**

Figura 13. Corrida en curso cuando pasa

El IDE lo que hace es grabar lo que se realiza en la pagina web esto lo convierte en comandos que posteriormente realiza automaticamente.

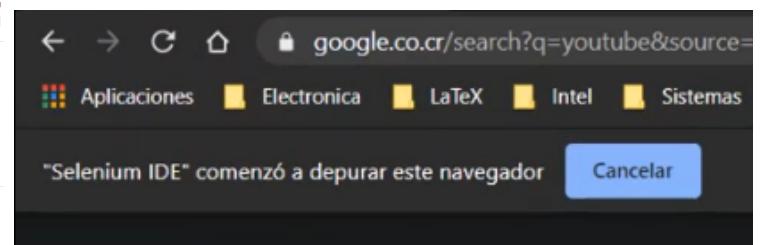


Figura 14. Selenium IDE corriendo el test automáticamente

Lo primero que se debe realizar es especificarle al IDE cual pagina es la que debe abrir para comenzar el test. El espacio marcado con el color amarillo es donde se debe introducir el URL.

Figura 15. URL

Posterior a introducir el URL se debe comenzar a grabar para esto se debe dar click en el botón REC.

Figura 16. Botón de grabación

Para finalizar la grabación se pulsa el botón que esta a la par del de grabar, pausar.

Figura 17. Botón para pausar la grabación

Y de esa manera queda listo el test para correrlo. Esta es una opción interesante para las personas que no sepan programar.

### REFERENCIAS

- [1] ALEXYNIOR (2019). *¿Qué es una prueba de regresión?*. <https://adictec.com/que-son-pruebas-de-regresion/>
- [2] THE QA TESTING CHANNEL. (2018). *Pruebas de Regresión en 1 Minuto*. <https://www.youtube.com/watch?v=OEclXnU6EQw>
- [3] INTERWARE (2018). *IMPORTANCIA DE REALIZAR REGRESION TESTING*. <https://www.interware.com.mx/blog/importancia-de-realizar-pruebas-de-regresi%C3%B3n#:~:text=Se%20dice%20que%20una%20regresi%C3%B3n,que%20todo%20sigue%20funcionando%20bien.>

- [4] ISAAC ÁLVAREZ DIZ (2018). *¿Por qué un test de Regresión?* <https://qanewsblog.com/2018/01/22/por-que-un-test-de-regresion/>
- [5] NERIO RODRIGUEZ (2017). *TestingBaires - Generar Script con Selenium IDE* [https://www.youtube.com/watch?v=\\_yoAoZuKF78&list=PL-a9\\_0KTG4ShuvHjf3EVFYERNSt915K7G&index=3](https://www.youtube.com/watch?v=_yoAoZuKF78&list=PL-a9_0KTG4ShuvHjf3EVFYERNSt915K7G&index=3)
- [6] QA MADNESS (2021). *Top-5 Tools for Regression Testing* <https://www.youtube.com/watch?v=HZvqfuADX8g>
- [7] QA MADNESS (2021). *Regression testing – What, Why, When, and How to Run It?* <https://www.youtube.com/watch?v=AWX6WvYktwk>
- [8] AUTOMATION STEP BY STEP - RAGHAV PAL (2021). *Selenium Python Small Sample Project I — Unit Test, HTML Reports* [https://www.youtube.com/watch?v=H9HUVSA\\_78U](https://www.youtube.com/watch?v=H9HUVSA_78U)