Trabajo Práctico Obligatorio - Entrega 2

Actividades de Testing en un proyecto Agile (Scrum)

Este Trabajo Práctico Obligatorio (TPO) simula las actividades que realiza una persona con el rol de Tester / Ingeniero o Analista de QA / SDET dentro de una Célula Scrum, para un proyecto de una plataforma de E-commerce, realizando actividades para cada una de las iteraciones o Sprints.

Objetivos Generales del TPO

* Conocer, Entender y Ejecutar algunas de las actividades que los Testers, Ingenieros de QA o SDET realizan en un equipo de ingeniería
* Adquirir conocimientos de cómo se lleva adelante un proyecto Agile
* Utilizar herramientas de desarrollo que suelen utilizarse en proyectos de Software
* Conocer cómo se lleva adelante la gestión de las tareas, requerimientos y defectos encontrados durante el desarrollo de una aplicación de Software

# Entrega 2 – Ejecución de Casos de Prueba y Automatización

## Objetivos de la Entrega

* Ejecución de los Casos de Prueba creados para la Entrega 1. Reporte de Defectos
* Actualización de los Casos de Prueba creados para la Entrega 1
* Testing Exploratorio
* Automatización de Casos de Prueba
* Análisis del proceso de Testing realizado

En la entrega 1 creamos Casos de Prueba en base a los Criterios de Aceptación especificados. Sin embargo, vimos que éstos quizás no eran lo suficientemente claros o tenían información que se contradecía con los mocks de la aplicación.

Una de las tareas más importantes de la Calidad del Software es tratar de refinar los requerimientos, ya que éstos son utilizados por todo el equipo de ingeniería (para desarrollar la aplicación).

Ahora, para esta entrega nos vamos a asumir que el proceso desarrolló avanzó hasta obtener una versión MVP (mínimum viable producto) y, como miembros del equipo de Calidad, vamos a ejecutar los casos de prueba creados, vamos a crear nuevos casos, y también automatizar y reportar defectos.

Con las tareas realizadas en esta entrega tendremos un panorama más amplio de qué actividades realizan Testers, Analistas Funcionales y Automatizadores.

### Descripción de la Entrega y de la Aplicación

**La versión de la aplicación “Plataforma Ecommerce” se encuentra disponible en:** [**http://automationpractice.com/index.php**](http://automationpractice.com/index.php)

* Esta versión de la aplicación está instalada en un entorno de pruebas similar al de producción
  + Tiene instalada una base de datos con productos de prueba
  + El servidor es más pequeño que el que usará producción, y por eso *se espera que tenga problemas de latencia o conectividad*
* Tiene una versión estable de la aplicación (versión MVP, con las funcionalidades mínimas necesarias para que salga a Producción)

## Consignas

La entrega consta de 4 módulos. Se espera que puedan presentar avances de forma semanal.

Consideraciones generales para todos los módulos:

* Todo lo que produzcan (casos de prueba, defectos, documentación, scripts de código) debe ser incluido al repositorio
* Consideren esta entrega como una actividad integral, que les puede servir como ejemplo de qué actividades realizan los equipos de ingeniería, y más específicamente quienes tienen roles relacionados al Testing
* Ante cualquier duda sobre las entregas o cómo presentar algo, consulten con el profesor durante la clase

### Módulo 1 - Análisis de los Casos de Prueba creados para las Historias de Usuario de la Entrega 1

Haremos un análisis de los Casos de Prueba realizados para la Entrega 1.

#### **Qué se debe presentar**

Como equipo, y como primera actividad de esta entrega, respondan lo siguiente:

1. Para las Historias de Usuario de la entrega 1:
   1. ¿Consideran que los Casos de Prueba creados, cuando los ejecuten, les darán la suficiente confianza de que la aplicación cumple con los Criterios de Aceptación?
   2. ¿Han tenido que asumir cómo se iba a comportar la aplicación para crear los Casos de Prueba?
   3. ¿Pueden identificar algo que no les haya permitido crear Casos de Prueba (por ejemplo, requerimientos poco claros, falta de visibilidad de cómo se va a realizar el proyecto, no conocer cómo se lleva adelante el desarrollo, etc) y por qué?
2. Algunas de las Historias de Usuario eran más claras que otras. ¿Cuál es la Historia de Usuario con la que más problema han tenido para crear Casos de Prueba y por qué? ¿Cuál ha sido la más clara de entender y por qué?

### Módulo 2 - Ejecución de los Casos de Prueba creados

En base a los Casos de Prueba (Test Case, TC) creados, deberán seleccionar un subconjunto y deberán ejecutarlos en el entorno de pruebas: <http://automationpractice.com/index.php>

En el archivo subido en Teams, “TPO – TC Template” tienen una sección de Ejecuciones. Pueden utilizarla en cada uno de los Casos de Prueba para especificar el resultado de la Ejecución.

#### Guía y Recomendaciones para realizar la ejecución de los Casos de Prueba

* Seleccionen un Subconjunto de Test Cases que ejecutarán: no necesitan correrlos todos, pero sí que identifiquen cuáles serán los primeros. En esto pueden basarse en las prioridades que les asignaron a cada uno de los TCs.
  + Indiquen en una lista cuáles son estos casos, junto a su prioridad
  + Además de la prioridad, tengan siempre en cuenta: los TCs que ejecuten les tienen que dar la suficiente confianza de que la aplicación funciona de acuerdo a los requerimientos
* Idealmente, la persona que escribió el TC no debería de ejecutarlo
* Indiquen, para cada TC, su resultado de ejecución, PASS o FAIL
  + Un TC PASS es aquel en el que todos los pasos la aplicación se comportó de acuerdo con lo esperado
  + Un TC FAIL es aquel en el que el sistema no se comportó de acuerdo con lo esperado

##### En caso de que un TC sea FAIL

Si un TC falla, significa que existen 2 situaciones posibles. Deberán tomar una decisión sobre qué hacer:

* La aplicación tiene un defecto
  + En este caso, deberán crear el reporte de un defecto. En el archivo subido en Teams, “TPO – TC Template” encontrarán el ejemplo de un reporte de defecto.
* El TC no es válido (no sigue los requerimientos planteados en las Historias de Usuario)
  + En este caso asumimos que el problema es que el TC no está definido de acuerdo a los requerimientos, y que la aplicación se comporta de acuerdo a lo esperado. Por eso, deberán actualizar el TC, y luego volver a ejecutarlo. Se espera que de esta ejecución su resultado sea PASS
  + En el archivo de “Entrega 1” anotamos los detalles adicionales de los requerimientos. Vamos a utilizar éstos como los requerimientos válidos de la aplicación: si la aplicación no se comporta de acuerdo a éstos, entonces la aplicación tiene un defecto.

#### **Qué se debe presentar**

* **Lista de TCs a ejecutar y la decisión de su selección**
* **Listado de qué TCs fueron modificados, y por qué**

### Módulo 3 - Testing Exploratorio

Suele ocurrir que, pese a que creamos casos de prueba para las historias de usuario, no cubramos todos los escenarios posibles (ya sea porque no tuvimos tiempo de ejecutar todos los casos de prueba, o porque algunos escenarios no estaban contemplados en los criterios de aceptación). Y es por eso que el Testing Exploratorio es una muy buena herramienta para encontrar defectos.

Se propone, para esta parte de la entrega, realizar Testing Exploratorio para las siguientes áreas de la aplicación:

* Comprar múltiples productos en una única operación
* Modificar una compra
* Agregar productos a la lista de deseos (Wishlist)
* Comparar productos

Seleccionen algunas de estas áreas, y organicen una actividad de Testing Exploratorio:

* Las sesiones no deben durar más de 120 minutos. La duración total dependerá de cuánta confianza les da la aplicación, y si estiman que van a encontrar defectos extendiendo la sesión
* No se recomienda que terminen una actividad de Testing Exploratorio y comiencen una nueva inmediatamente
* La organización de las actividades de Testing exploratorio corre por cuenta del equipo
* Se recomienda que, durante la sesión de Testing Exploratorio anoten los defectos que encuentran, y luego de terminada realicen el reporte de los defectos
* Una sesión de Testing Exploratorio no falla porque no se encuentran defectos: se ganó conocimiento de la aplicación

Pueden utilizar la siguiente tabla de ejemplo para organizar la actividad:

|  |  |
| --- | --- |
| Testing Exploratorio – Sesión 1 |  |
| Objetivo | Tratar de encontrar errores al crear listas de deseos |
| Posibles Escenarios / Posibles Riesgos | Agregar una nueva lista de deseos. Modificar una lista existente. Borrar una lista de deseos |
| Variaciones | Comprar los productos que existen en una lista de deseos. Agregar el mismo producto a varias listas de deseo. Borrar todas las listas de deseo creadas |
| Técnicas a utilizar | Transición de Estados |
| Notas | Se verifica primero la creación y borrado de lista de deseos. Luego se probará modificar. Luego, validar el comportamiento si un producto de la lista de deseos es comprado |

#### Qué se debe presentar

* Resumen de las actividades: sesiones realizadas
* Defectos o mejoras encontradas
* Si no sabemos si el comportamiento de la aplicación es el esperado, confirmar con el profesor

### Módulo 4 - Automatización

Este módulo propone que comiencen a explorar las herramientas que existen para realizar la automatización de Casos de Prueba.

Como tenemos una aplicación web, podemos utilizar herramientas como Selenium, que nos permitirán automatizar acciones a realizar sobre un explorador web. Luego, podremos construir Casos de Prueba automatizados para disminuir el esfuerzo de ejecución manual.

#### Parte A - Webdriver

Para poder ejecutar scripts que utilicen Selenium en sus PC necesitamos:

* Un webdriver para el explorador web que vamos a utilizar
* Instalar las librerías de Selenium, para el lenguaje de programación que vayamos a utilizar (que ya lo hicimos para la Pre-Entrega)

##### Selenium Webdriver

Se mencionan a continuación los pasos para instalar el webdriver de Google Chrome:

* Verificar qué versión de Google Chrome tenemos en nuestra PC (la versión 107 es la última al momento que esta guía fue escrita)
* Ir a <https://chromedriver.chromium.org/downloads> y hacerle click al link de la versión de nuestro explorador
* Descargar la versión correspondiente a nuestro Sistema Operativo
  + Windows: chromedriver\_win32
  + Mac OS (Intel): chromedriver\_mac64
  + Mac OS (Apple Silicon): chromedriver\_mac\_arm64

Luego, descomprimir el archivo y configurarlo como ejecutable por nuestro sistema operativo.

**Tutorial de cómo configurarlo:** [**https://zwbetz.com/download-chromedriver-binary-and-add-to-your-path-for-automated-functional-testing/**](https://zwbetz.com/download-chromedriver-binary-and-add-to-your-path-for-automated-functional-testing/)

* **En caso de problemas, consultar con su equipo o con el profesor**

Para verificar que el webdriver es reconocido como ejecutable, pueden abrir una terminal y ejecutar:

Windows > chromedriver.exe -v

Mac OS > chromedriver -v

#### Parte B - Selenium IDE

* Instalar la extension “Selenium IDE” para Google Chrome: <https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd?hl=en>
* Luego, les aparecerá un ícono de “extensiones” a la derecha de la barra de búsqueda del explorador. Al hacerle click se abrirá la ventana de Selenium IDE

Graphical user interface, text, application

Description automatically generated

* Seleccionar “Record a new test in a new project”. Escriban un nombre para el proyecto
* Ingresen la URL de la aplicación de Prueba: <http://automationpractice.com/index.php>

Graphical user interface, application

Description automatically generated

* Luego de presionar OK, se abrirá una nueva ventana de Chrome. Y todos los pasos que realicen se grabarán

##### Exportar un script de prueba

Cuando Selenium IDE esté grabando, realicen los siguientes pasos en la página:

* Hacer click al cuadro de búsqueda
* Escribir “dress”
* Hacer click en el botón de búsqueda (ícono de la lupa)
* Luego de que cargue la página, ir a la ventana de Selenium IDE y detener la grabación. Guardar la grabación con algún nombre
* Luego, exportar la grabación como un script de Python:
  + Verifiquen que la opción “Tests” esté seleccionada, y luego seleccionar la opción “Export” que aparece en el menú de la grabación que acaban de hacer

Graphical user interface, application, email

Description automatically generated

* Exporten el archivo

Graphical user interface, application

Description automatically generated

El script exportado será similar al siguiente:

# Generated by Selenium IDE  
import pytest  
import time  
import json  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.common.action\_chains import ActionChains  
from selenium.webdriver.support import expected\_conditions  
from selenium.webdriver.support.wait import WebDriverWait  
from selenium.webdriver.common.keys import Keys  
from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities  
  
class Test001():  
 def setup\_method(self, method):  
 self.driver = webdriver.Chrome()  
 self.vars = {}  
   
 def teardown\_method(self, method):  
 self.driver.quit()  
   
 def test\_001(self):  
 self.driver.get("http://automationpractice.com/index.php")  
 self.driver.set\_window\_size(1539, 1018)  
 self.driver.find\_element(By.ID, "search\_query\_top").click()  
 self.driver.find\_element(By.ID, "search\_query\_top").send\_keys("dress")  
 self.driver.find\_element(By.NAME, "submit\_search").click()

* Copien este archivo al repositorio (recomendación: crear una carpeta “automated” dentro de “test\_cases”)
* Ejecuten este archivo desde su IDE o una terminal
  + *Pueden ejecutarlo desde el IDE o desde una terminal, con el comando “pytest <nombre\_del\_archivo.py>”*
  + Al ejecutarlo, debería abrirse una ventana de explorador y se deberían ejecutar los mismos pasos que realizó Selenium IDE

Nota: dependiendo de qué pasos haya grabado Selenium IDE, el script puede fallarles o no.

#### Parte C - Assertions

El script que guardamos sólo realiza acciones sobre un explorador, pero no realiza ningún tipo de validación. En el ejemplo presentado, sólo se dirige a una página luego de hacer una búsqueda y el test termina.

Es por eso que necesitamos darle “inteligencia” a nuestros casos de prueba automatizados, para que sepan qué validaciones tienen que hacer para verificar que un caso de prueba es PASS o FAIL.

##### Agregamos una aserción para validar que al realizar una búsqueda que trae resultados no se muestre un mensaje de error

Primero tenemos que buscar qué elemento de la página podemos utilizar para validar si la búsqueda devuelve resultados o no

* Por ejemplo, si buscamos la palabra “invalid” no se traerá ningún resultado

Graphical user interface, text, application

Description automatically generated

Luego, abrimos la consola de desarrollador del explorador y buscamos el mensaje de error que se muestra en el HTML de la página:

A screenshot of a computer

Description automatically generated with medium confidence

Vemos que el elemento está identificado con la clase “alert-warning”. Vamos a utilizar esto para validar cuándo aparece este mensaje en la página y verificar que una búsqueda válida trae resultados.

Para hacer esto, agregamos las siguientes líneas al final del método de test:

time.sleep(1)  
with pytest.raises(NoSuchElementException) as exif:  
 self.driver.find\_element(By.CLASS\_NAME, "alert-warning")  
assert 'Unable to locate element' in exif.value.msg

time.sleep(1)

* Una breve espera por si la página tarda en cargar. No es necesaria

with pytest.raises(NoSuchElementException) as exif:  
 self.driver.find\_element(By.CLASS\_NAME, "alert-warning")

* Esperamos que para una búsqueda válida Selenium no encuentre el elemento. Por eso, utilizamos “with” y la excepción que se va a lanzar al buscar el elemento en la página

assert 'Unable to locate element' in exif.value.msg

* El mensaje de error incluye el string “Unable to locate element”. Podemos utilizar esto para verificar si es la excepción esperada

Entonces, el método del paso quedaría de la siguiente forma:

def test\_001(self):  
 self.driver.get("http://automationpractice.com/index.php")  
 self.driver.set\_window\_size(1539, 1018)  
 self.driver.find\_element(By.ID, "search\_query\_top").click()  
 self.driver.find\_element(By.ID, "search\_query\_top").send\_keys("dress")  
 self.driver.find\_element(By.NAME, "submit\_search").click()  
 time.sleep(1)  
 with pytest.raises(NoSuchElementException) as exif:  
 self.driver.find\_element(By.CLASS\_NAME, "alert-warning")  
 assert 'Unable to locate element' in exif.value.msg

Si ejecutamos este test debería de pasar. Y si cambiamos el texto de la búsqueda (cambiamos “dress” por “invalid”) el test fallará.

#### Qué se debe presentar

* Al menos 2 casos de prueba automatizados.
  + Seleccionen alguno de los casos de prueba que hayan creado, graben los pasos con Selenium IDE y luego exporten el archivo pytest.
  + Agreguen las aserciones que les permitan verificar que el TC automatizado hace las mismas (o más) verificaciones que el TC manual
  + Aquellos elementos dinámicos, o pop-ups en la página (por ejemplo, las vistas previas o Quick-view de los productos) son un poco más complejos de automatizar. Por eso, se recomienda que comiencen con TCs simples (Test Cases de Login, por ejemplo).