UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Pruebas automatizadas con Selenium

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Erick Hazel Rocha García

ASESORA DE INFORME

Ing. Josefina Rosales García

Ciudad Universitaria, Cd. Mx., 2024

Agradecimientos.

La serie de acciones y decisiones que me han llevado hasta aquí son el resultado del esfuerzo de mi familia. Tuvimos momentos difíciles, también momentos que nos llenaron de alegría, siempre los llevaré conmigo. No podría haber hecho semejante logro sin su apoyo y sacrificio.

Estoy profundamente agradecido con mis padres que me orientaron a no dejar las cosas, persistir ante la adversidad y saber adaptarme, así como todo el esfuerzo de darnos lo mejor. Por cuidarnos durante esta época tan difícil como lo fue la pandemia.

A mi madre en especial por mostrarse la fuerza de voluntad que puedo tener y la inteligencia emocional que debemos tener mi hermana y yo. A ser tolerante y comprender al resto de personas que nos rodean.

A mi padre por mostrarme este mundo tecnológico y heredarme de esas habilidades para tratar de arreglar algo. Por tu carácter y los momentos de enseñanza de lo que significa ser un hermano, un padre, un líder.

A mi hermana por ayudarme con mi carga escolar y ser la persona con la puedo contar, con la que puedo divertirme jugando en el patio como niños. Por enseñarme de creatividad y paciencia.

Al hermano menor de mi mamá y a mi abuela materna por mostrarme que para hacer terminar algo no importa el tiempo ni la edad, sino las ganas que tiene uno para superarse.

A mi tío y abuelo paterno por ayudarnos situaciones donde mis padres no podían estar. Por las alegrías y aventuras que pasamos en esos trayectos.

A mis abuelos maternos por siempre confiar y alentarme a lo que podemos aspirar mi hermana y yo.

A mi esposa, por estar a mi lado estos años, por tener su cariño y ser parte de mi familia. Sin ella, sus enseñanzas y su soporte durante mi época universitaria habrían tenido un final diferente. Aprendí que hay cosas que no veo tan fácil y debo explorar más de mundos ajenos. Contigo mi visión de las cosas aumento a terrenos que por mí mismo no hubiera podido. Gracias por tu paciencia.

A la familia de mi esposa por su cariño y apoyo durante mis temporadas que estuve con ellos aun cursando la universidad.

A todas las personas que cruzaron camino conmigo, que ahora forman parte de mi ser y que sin ellas no sería el hombre que soy. A todas ellas, gracias.

A los profesores y profesoras que me guiaron a lo largo de estos años con sus conocimientos y experiencias. En particular, a la Ing. Josefina Rosales García por ayudarme encontrar mi camino en el mundo laboral y el desarrollo de este trabajo.

También a Omnitracs/Solera y las personas que colaboran dentro de esta, por darme la oportunidad laboral y de desarrollar este trabajo con base a la experiencia que me brindaron.

Índice general

1. Introducción.

El presente informe de actividades profesionales describe el proceso de actualización de una aplicación de automatización perteneciente a la empresa Solera.

Este proyecto surge como respuesta a la innovación de la marca para poder estar a la vanguardia en el aseguramiento de calidad de software y con el fin de mejorar los procesos de verificación y validación del sistema web a través de las pruebas automatizadas.

El proyecto envuelve conocimientos de codificación, programación orientada a objetos, conocimientos básicos de páginas web, un paradigma de programación llamado *Page Object Model* y uso de *git,* así como las bases del marco de trabajo *Selenium WebDriver*.

Cabe destacar que el proyecto no se desarrolló desde cero, sino que con anterioridad otras personas ya habían analizado los requerimientos del sistema. Hicieron una exploración y establecieron como resolver la situación de automatización decidiendo utilizar Selenium WebDriver como la herramienta de interacción con el navegador y Java como lenguaje de programación. Sin embargo, debido al poco personal, la fase de pruebas fue establecida y desarrollada en su totalidad por mí, acatando los requerimientos ya acordados por los analistas y aprovechando mis habilidades con mi primer desarrollo dentro de la compañía.

De inicio, el proyecto se presenta con Maven como gestor de proyectos haciendo uso de Eclipse IDE como herramienta de escritura de código, por lo que en este documento se expresan los cambios realizados y el proceso por el cual se pasó para culminar con el proyecto que se tiene hoy en día.

2. Objetivo.

Analizar e implementar mejoras de manera integral a los procesos, tareas y actividades sobre el software de automatización para la página web de la empresa a través del *framework* Selenium WebDriver con la creación de códigos en el lenguaje de programación Java, así como del software de gestión Gradle.

2.1. Alcances.

- Actualizar el software referente para las pruebas automatizadas.
- Otorgar una mayor versatilidad al automatizador al agregar subproyectos y nuevas funcionalidades.
- Impulsar una buena gestión del automatizador para los futuros casos de pruebas a partir de una documentación más clara y actualizada.
- Registrar los cambios hechos en el proyecto de software para que pueda usarse como base de conocimiento para el resto de los integrantes del equipo *QA*.

3. Descripción de la empresa y labor desempeñada.

La empresa comenzó como Audatex, conocida como el primer sistema automatizado de reparación de vehículos que se expandió por Europa Y América del norte, ofreciendo soluciones para necesidades de seguros y reparación de automóviles, pero ésta entró a formar parte de Solera en 2006.

Actualmente, la corporación se encarga de proveer servicios de gestión del ciclo de vida de los vehículos ofreciendo diferentes soluciones para vehículos particulares como transportes de carga.

La empresa ha crecido orgánicamente a través de más adquisiciones para expandir sus plataformas de software y datos en puntos clave del ciclo de vida de propiedad de un vehículo a través de sus cuatro líneas de negocio:

- reclamación,
- reparación y
- soluciones para vehículos, así como
- soluciones para flotillas.

Particularmente, el ámbito de desarrollo en el que se desenvuelve el proyecto es para hacer pruebas de un servicio web que se encarga de la logística de los transportes de carga.

El cargo que desempeño es como *QA Tester* enfocado, principalmente, en las pruebas automatizadas. Mis principales responsabilidades como QA son:

- Diseñar, desarrollar y mantener scripts de pruebas automatizadas.
- Ejecutar casos de prueba automatizados y analizar los resultados para identificar defectos de software.
- Realizar pruebas de regresión para garantizar que las funciones existentes no se vean afectadas por nuevos cambios.
- Revisar y mejorar los procesos, metodologías y herramientas de calidad de los QA existentes.

4. Contexto laboral y definición del problema.

4.1. Contexto laboral.

Mi participación dentro de la empresa comenzó siendo becario dentro de un periodo de 6 meses. Esto con la finalidad de obtener una oferta laboral al finalizar este tiempo debido a que recién había llegado la compañía al país, durante la temporada de cuarentena, y quería comenzar a preparar nuevo talento en México.

De inicio, estuve en el área de desarrollo de software donde, en conjunto con un grupo de 3 personas, se creó un automatizador para la instalación de las herramientas que usarían los desarrolladores de software, culminando con un puesto dentro de la empresa como *QA tester* al finalizar esa pieza de software y mi estadía como becario debido a que personas dentro del área estaban por cambiar de empresa.

4.2. Participación profesional.

Las actividades que realizo en la empresa son las siguientes:

- Superviso y mantengo en funcionamiento un programa que monta el servidor local en cada máquina de los usuarios de manera automática.
- Me encargo en la creación de prueba automatizadas para la plataforma web de la empresa los cuales garantizan el correcto funcionamiento de las nuevas o actualizadas características que llegarán al cliente.
- Participo en la búsqueda, recopilación y creación de los datos para las pruebas automatizadas en cada uno de los distintos servidores.
- Verifico que las pruebas codificadas por otros compañeros sean funcionales en los distintos servidores de ensayo que se tienen.
- Apoyo con las cuestiones técnicas que surgen en la instalación del servidor local durante los lanzamientos de producto.
- Pruebas manuales a la API y servicio web de la empresa.

- Realizo la planificación para la automatización de las pruebas automatizadas de las API, así como los correspondientes para el servicio web.
- Le doy mantenimiento al software de automatización de QA en las áreas de web. móvil y API.

4.3. Definición del problema.

Una parte importante del desarrollo de software es la construcción de los proyectos y los sistemas completos. Desde la clásica herramienta *make* en los 70's, ha habido varios mecanismos y sistemas distintos para construcción de proyectos, algunos enfocados a ciertos aspectos de la construcción, otros tratando de abarcar todo el espectro: compilación, manejo de dependencias, integración continua, automatización del proceso de construcción, etc (Zamudio, 2024).

Es durante este proceso de desarrollo de software que el papel de *QA tester* entra en acción y, para tener buenos resultados en esta área de pruebas, se necesita de las mejores herramientas posibles, sin éstas o sin la debida actualización hacia las nuevas tecnologías que el ámbito provee, se podría ralentizar el trabajo de los aseguradores de calidad llevando a la compañía a necesitar de más personal incrementando los costos o aumentando el tiempo del ciclo de software.

Actualmente existen distintas propuestas para la gestión y construcción de proyectos Java, tales como Maven, Gradle o Ant. Sin embargo, tanto Ant como Maven hacen uso de los archivos XML para la gestión de dependencias y herramientas de construcción. Aunque este tipo de archivo no es difícil de comprender, sí se dificulta al momento de que este se va volviendo más extenso debido a la utilización de etiquetas, tal como lo hace HTML, y sin el apropiado procesador de texto que enriquezca la interacción y lectura de este, puede resultar en un problema, sobre todo para los recién iniciados en este tipo de proyectos.

Asimismo, para hacer uso de esta herramienta y llevar a cabo la escritura de los casos de prueba se hace uso del IDE Eclipse que, a pesar de no ser una mala herramienta, sí presenta una interfaz un poco antigua, aparte de que su gestor de extensiones no es la más rápida ni

la más exenta de errores al momento de querer instalar los *plug-in* lo que impacta al *QA tester* para iniciar con su labor. En adición, al ser una herramienta de codificación más completa, conlleva a considerar un mayor espacio en disco y uso de recursos de la computadora.

Por otro lado, al iniciar este proyecto los principales contribuidores del software dejaron la empresa pasando el conocimiento a varios miembros del equipo que, de igual forma, cambiaron de puesto fuera de la compañía, dejando poca documentación al respecto, sin persona alguna dentro de la empresa a la cual acudir por soporte y en medio de una expansión hacia otro tipo áreas de oportunidad del producto, se presentó, por parte de los líderes, una tarea difícil de lograr: preparar la aplicación para más pruebas automatizadas y con esto añadirla a un flujo de integración y distribución continua.

Con esto en mente, se identifica la necesidad de revitalizar al proyecto de software dotándolo de mejoras que permitan la integración y distribución continua, así como una actualización de las herramientas de trabajo que el *tester* usará día con día para la creación de sus casos de prueba. En consecuencia, se tendrá un proyecto más comprensible para los nuevos usuarios que necesitan aprender de este ambiente, de modo que su mantenimiento a largo plazo sea más accesible respetando Selenium WebDriver como la herramienta de interacción con el navegador y Java como lenguaje de programación.

5. Marco teórico.

5.1. ¿Qué es software.?

“En computación, el software en sentido estricto es todo programa o aplicación programada para realizar tareas específicas. Es el soporte lógico e inmaterial que permite que la computadora pueda desempeñar tareas inteligentes, conduciendo a los componentes físicos o hardware con instrucciones y datos a través de diferentes tipos de programas” (Olvera, 2014). Y no se trata sólo de hacer que la máquina funcione; va más allá, logrando ayudarnos con éxito en el desempeño de un trabajo específico.

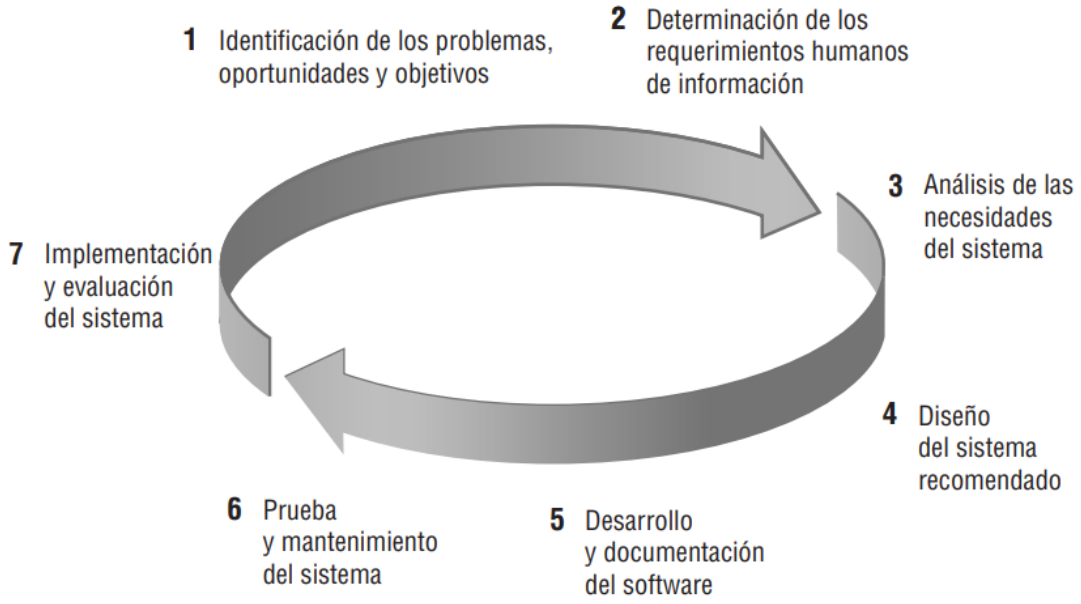
5.2. Metodologías de desarrollo de software.

Para llevar a cabo estos proyectos de software se debe contemplar el llamado ciclo de vida del desarrollo de software (SDLC por sus siglas en inglés). El SDLC es un conjunto de actividades y resultados asociados que producen un producto de software. Según Kendall & Kendall se pueden tener 7 etapas y pueden apreciarse en la Figura 1:

1. Identificación de los problemas, oportunidades y objetivos.
2. Determinación de los requerimientos humanos de información.
3. Análisis de las necesidades del sistema.
4. Diseño del sistema recomendado.
5. Desarrollo y documentación del software.
6. Prueba y mantenimiento del sistema.
7. Implementación y evaluación del sistema.

Figura 1

Las siete fases del ciclo de desarrollo de sistemas (SDLC)



Nota. Kendall, K. A., & Kendall, J. E. (2005). *Análisis y diseño de sistemas* [Diagrama]. Pearson Educación.

"La metodología de desarrollo de software es el conjunto de técnicas y métodos que se utilizan para diseñar una solución de software informático. Es importante señalar que existen varias, de manera que es una decisión de cada equipo" (Colaborador de UCMA, 2023).

Algunos requisitos que deben tener las metodologías de desarrollo según Gómez, López, & Bacalla(2014):

- Visión del producto.
- Vinculación con el cliente.
- Establecer un modelo de ciclo de vida.
- Gestión de los requisitos.
- Plan de desarrollo.
- Integración del proyecto.

- Medidas de progreso del proyecto.
- Métricas para evaluar la calidad.
- Maneras de medir el riesgo.
- Como gestionar los cambios.
- Establecer una línea de meta

Estas metodologías pueden dividirse en 2 categorías; las tradicionales y las ágiles.

5.2.1. Metodologías tradicionales.

En metodologías tradicionales, "un proyecto pasa por una serie de ciclos de vida que incluyen las fases de inicio, planificación, ejecución, seguimiento y cierre. La planificación inicial requiere una documentación completa, ya que la desviación de alcance previamente establecido puede presentar un riesgo grave para el proyecto. Cabe destacar que generalmente la entrega del producto se realiza al finalizar el proyecto" (Colaborador de Smartsheet Latam, 2024).

Aquí se destacará a la metodología de prototipos.

"Pertenecen a los modelos de desarrollo evolutivo. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto" (Santander Universidades, 2024). "Se caracterizan por la forma en que permiten a los ingenieros de software desarrollar versiones cada vez más completas del sistema. Los modelos evolutivos iteran sobre las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla a partir de los requerimientos prioritarios o los que están mejor definidos. Esta primera versión se refina en una nueva iteración con las peticiones del cliente para producir un sistema que satisfaga sus necesidades" (Ojeda. & Fuentes, 2012).

Tomando en consideración las ventajas y desventajas descrita por Armenta Benitez, Rodriguez Espinoza, Medina Muñoz, & Gonzalez López (2018) se presenta la siguiente tabla para comparación:

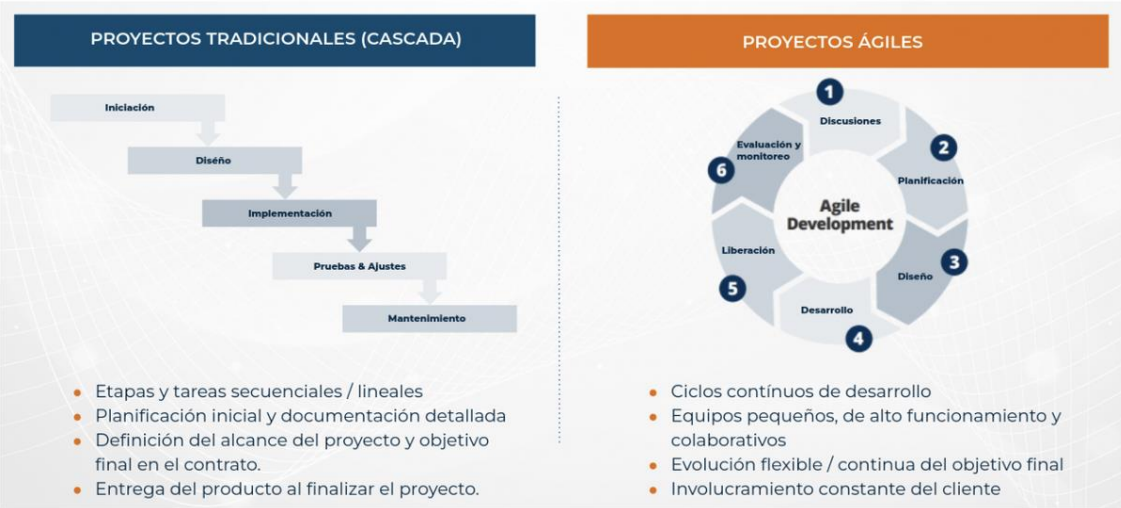
Ventajas	Desventajas
• Es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida. • Ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.	• Una vez construido el prototipo final es difícil hacer cambios de último momento. • Es posible que el prototipo sea muy lento, muy grande, no muy amigable en su uso, o incluso, que esté escrito en un lenguaje de programación inadecuado. • El desarrollador puede ampliar el prototipo para construir el sistema final sin tener en cuenta los compromisos de calidad y de mantenimiento que tiene con el cliente

5.2.2. Metodologías ágiles.

"A diferencia de la metodología anterior, las metodologías ágiles se basan en un enfoque incremental e iterativo, donde las fases de la ejecución de los proyectos son flexibles, evolucionan y a menudo se superponen entre sí. En lugar de una planificación en profundidad al comienzo del proyecto, las metodologías ágiles están abiertas a requisitos cambiantes a lo largo del tiempo y fomentan la retroalimentación constante de los usuarios finales. El objetivo de cada interacción es generar un producto que funcione, lo cual lo hace una elección más adecuada para proyectos en los que el cliente no está seguro del resultado deseado, busca un tiempo de entrega rápido y quiere participar de cerca en el proceso de diseño" (Colaborador de Smartsheet Latam, 2024). Seguidamente puede observarse la Figura 2 con la comparación de ambas metodologías.

Figura 2

Diferencias entre metodologías tradicionales y ágiles



Nota. Smartsheet LATAM. (2024, 4 marzo). *Diferencias entre metodologías ágiles y tradicionales: ventajas y desventajas* [Diagrama]. https://www.linkedin.com/pulse/diferencias-entre-metodolog%C3%ADas-%C3%A1giles-y-tradicionales-ventajas-/?originalSubdomain=es

5.3. Ambientes de trabajo para el desarrollo de software.

Tal y como se menciona en el artículo de la DGTIC (2023), para la ejecución de las actividades que comprenden las etapas antes mencionadas, se ocupan espacios de trabajo donde los equipos de desarrollo y pruebas puedan llevarlas a cabo. Al menos, estas deben contemplar 3 ambientes de trabajo:

- **Ambiente de desarrollo**: se usa para integrar el código de los diferentes componentes del equipo de programación, así como los elementos que forman el sistema de información o aplicación, como son: lenguajes, bibliotecas, *frameworks*, base de datos, entre otros.
- **Ambiente de pruebas**: tendrá las mismas características de configuración y recursos que el ambiente de trabajo anterior, pero aquí se despliegan las versiones estables del sistema o aplicación para que el equipo de pruebas realice las verificaciones de calidad pertinentes, por ejemplo: pruebas de funcionalidad, usabilidad, entre otras,

para su mejora o corrección, sin interferir con las actividades de trabajo del equipo de desarrollo.

- **Ambiente de producción**: es el espacio donde operará el sistema o aplicación, es decir, es donde se hace uso del servicio por parte de los usuarios finales. Aquí se realizan los ajustes a los elementos necesarios para mantener la disponibilidad del servicio y se programan los respaldos del ambiente.

5.4. Calidad del software.

Sí se hace uso de la definición que provee la Real Academia Española se tiene que calidad es la "propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor". Con esto en mente se presenta ese conjunto de características que define a un software como bien diseñado, según Pressman:

- **Mantenibilidad**: El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes. Este es un atributo crítico debido a que el cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocios.
- **Confiabilidad**: La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, protección y seguridad. El software confiable no debe causar daños físicos o económicos en el caso de una falla del sistema.
- **Eficiencia**: El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento, utilización de memoria, etc.
- **Usabilidad**: El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz apropiada y una documentación adecuada.

5.5. Métodos para evaluar calidad de software

Pressman (1998) señala que el aseguramiento de la calidad del software incluye un rango amplio de preocupaciones y actividades que se centran en la administración de la calidad del software.

“El aspecto de la calidad vinculado al software consiste en llevar adelante este proceso de la mejor forma que permita al proyecto asociado terminar en el tiempo planificado y dentro del presupuesto asignado” (Pantaleo & Rinaudo, 2015).

Tiene ciertas actividades que se llevan a cabo. Éstas se expresan como sigue según Pressman (1998):

- Estándares,
- Revisiones y auditorías,
- Pruebas,
- Colección y análisis de los errores,
- Administración del cambio,
- Educación,
- Administración de los proveedores,
- Administración de la seguridad,
- Seguridad,
- Administración de riesgos (Pressman, 1998).

En esta área se destacará a los *QA testers* porque es el área donde este trabajo se desarrollará para su uso continuo dentro de la compañía.

5.6. ¿Qué es un QA?

“*QA Tester* (*Quality Assurance*) es un asegurador de la calidad. Es un perfil profesional orientado principalmente a la medición de la calidad de los procesos utilizados para crear un producto de calidad “ (Colaborador de iwantic, 2024).

“Su objetivo es evaluar cuidadosa y metódicamente productos de software con el fin de descubrir cualquier defecto, error o inconsistencia que pueda estar oculta bajo la superficie” (Colaborador de QAlified, 2023).

Dentro de las responsabilidades de un *QA Tester* se encuentran:

- Planificación de pruebas.

- Diseño de casos de prueba.
- Ejecución de pruebas.
- Reporte de defectos.
- Pruebas de Regresión.
- Pruebas de Rendimiento.
- Pruebas de Usabilidad.
- Pruebas automatizadas.

5.7. Tipos de pruebas

Somerville (2005) se refiere a que, para la mayoría de los sistemas complejos, existen dos fases distintas de pruebas del sistema:

- **Pruebas de integración**, en las que el equipo de pruebas tiene acceso al código fuente del sistema. Cuando se descubre un problema, el equipo de integración intenta encontrar la fuente del problema e identificar los componentes que tienen que ser depurados. Las pruebas de integración se ocupan principalmente para encontrar defectos en el sistema.
- **Pruebas de entregas**, en las que se prueba una versión del sistema que podría ser entregada a los usuarios. Aquí, el equipo de pruebas se ocupa de validar que el sistema satisface sus requerimientos y asegurar que el sistema es confiable. Las pruebas de entregas son normalmente pruebas de «caja negra» en las que el equipo de pruebas se ocupa simplemente de demostrar si el sistema funciona o no correctamente. Los problemas son comunicados al equipo de desarrollo cuyo trabajo es depurar el programa. Cuando los clientes se implican en las pruebas de entregas, éstas a menudo se denominan pruebas de aceptación. Si la entrega es lo suficientemente buena, el cliente puede entonces aceptarla para su uso.

Es en las pruebas de entregas donde los *testers* de la compañía se centran en resolver y que aún puede dividirse en:

- Pruebas manuales y
- Pruebas automatizadas.

5.8. Pruebas manuales.

“Las pruebas manuales son ejecutadas por *testers* que tienen interacción directa con el software, en muchos casos, empleando herramientas adecuadas para cada propósito. Los *testers* se encargan de configurar un entorno en el que se llevará a cabo la ejecución de pruebas, haciendo un recorrido exhaustivo con el software objetivo de las pruebas” (Colaborador Abstracta, 2024).

Tal y como lo mencionan en QAlified (2023) los pasos para generar pruebas manuales comienzan con la extracción de datos para crear casos de prueba. Los datos pueden recopilarse de sistemas existentes, escenarios de usuarios finales y el diseño inicial del producto.

Una vez que se recopilan los datos, el siguiente paso es diseñar los casos de prueba. Estos casos de prueba proporcionan instrucciones sobre qué probar, la salida esperada y los datos necesarios para la entrada.

La ejecución de estas pruebas es una de las partes más críticas de las pruebas de software. Cualquier cosa descubierta que no esté alineada con los requisitos se considerará un error. Se comunicará a los desarrolladores, y el caso de prueba se marcará como fallido. Este proceso continuará hasta el último caso de prueba.

5.9. Pruebas automatizadas.

“La prueba automatizada consiste en utilizar un software especial para controlar la ejecución de las pruebas y la comparación de los resultados reales con los resultados esperados “ (M., M., & O., 2019).

Para realizar la automatización de este tipo de pruebas se siguen los siguientes pasos:

1. Se revisa el sitio web o aplicación a probar
2. Se diseña el caso de prueba con los pasos a seguir
3. Se localizan los elementos web en el HTML para generar el código
4. Se implementan las pruebas
5. Se registran los resultados

6. Finaliza la prueba

En muchos casos, las pruebas manuales se realizan primero para una posterior automatización del proceso, si este lo requiere.

La automatización de pruebas se vuelve un punto importante cuando un proceso es repetitivo y se quiere reducir el tiempo, así como recursos, para analizar el producto. De igual manera, cuando el software es lo suficientemente estable y solo se hacen cambios mínimos y no se agregan nuevos requerimientos.

La industria busca que la automatización ayude a la rápida verificación del sistema.

Refiriéndose a la creación de software, no hay mucho que cambiar. La elaboración de código fuente, para su implementación en software, tiene que ser tan veloz como las pruebas que se le tengan que realizar para ofrecer una opción con los menos errores posibles. Es en este punto donde entra en juego la realización de pruebas automatizadas.

5.10. Pruebas automatizadas vs pruebas manuales.

A continuación, se muestra una tabla comparativa de las diferencias entre estos tipos de pruebas de QAlified (2023):

	Pruebas Manuales	***Pruebas Automatizadas***
Tiempo de Ejecución	Las pruebas manuales consumen mucho tiempo, ya que los casos de prueba se ejecutan uno tras otro, y el uso de recursos también es alto.	Dado que las pruebas se crean y realizan utilizando herramientas especializadas, se requiere menos tiempo y recursos para llevarlas a cabo, lo que resulta en una entrega de producto más rápida.
Configuración Inicial	Las pruebas se ejecutan manualmente, por lo que se necesita menos esfuerzo en la configuración inicial.	Se requiere mucho esfuerzo en la configuración inicial debido a la creación y mantenimiento de los scripts/herramientas.
Confiabilidad	La confiabilidad de las pruebas manuales es menor, ya que las pruebas no se pueden realizar con	Las pruebas automatizadas son mucho más confiables en comparación con las pruebas manuales. Esto se debe a que estas pruebas realizan la misma

	alta precisión cada vez debido a errores humanos.	operación cada vez que se ejecutan, lo que las hace más confiables para fines de prueba.
Integración con CI/CD Pipelines	Requiere la ejecución manual de pruebas, por lo que es difícil de integrar en tuberías CI/CD.	Las pruebas automatizadas se pueden integrar fácilmente con las tuberías CI/CD debido a la ejecución constante de pruebas y retroalimentación.
Programación	Las pruebas manuales no son programables. No se puede aplicar programación para crear pruebas complejas que recuperen defectos.	En las pruebas automatizadas, los testers pueden crear pruebas complejas para descubrir defectos ocultos. Así que, con programación, esto puede lograrse.
Reutilización	En las pruebas manuales, los testers deben definir nuevos casos de prueba para cada función.	Los scripts de prueba son utilizables en varios ciclos del software, lo que lo hace más atractivo para realizar pruebas.
Informe	La salida de las pruebas manuales se informa como implementada, por lo que puede haber variaciones en el informe.	Garantiza consistencia en el seguimiento e informes, generando así informes de prueba estandarizados.
Flexibilidad	Ofrece flexibilidad para adaptarse a escenarios de prueba y cambios en los requisitos.	Dado que se desarrollan programas para realizar estas pruebas, es menos flexible ante cambios no anticipados.
Cobertura de Resultados	Cuando un tester realiza pruebas manuales, los demás miembros del equipo de desarrollo no verán los resultados.	En el caso de las pruebas automatizadas, los miembros del equipo pueden iniciar sesión en el sistema para ver los resultados en tiempo real. Esto mejora la colaboración y contribuye a obtener productos de alta calidad.

5.11. Herramientas y técnicas de desarrollo para automatización de pruebas.

En ZAPTEST (2022) consideran que la elección de las herramientas de automatización de pruebas adecuadas es esencial. Las herramientas de pruebas automatizadas funcionan mejor cuando son:

- Fácil de usar
- Capaz de probar una variedad de sistemas operativos, navegadores y dispositivos
- Equipado con las herramientas necesarias (*full stack*) para probar lo que necesita
- Que sea compatible con su lenguaje de scripting y que sea fácil de usar incluso para personas que no conocen el lenguaje de scripting o no tienen conocimientos de codificación
- Reutilizable para múltiples pruebas y cambios
- Capacidad para aprovechar grandes conjuntos de datos de múltiples fuentes para proporcionar validaciones basadas en datos.

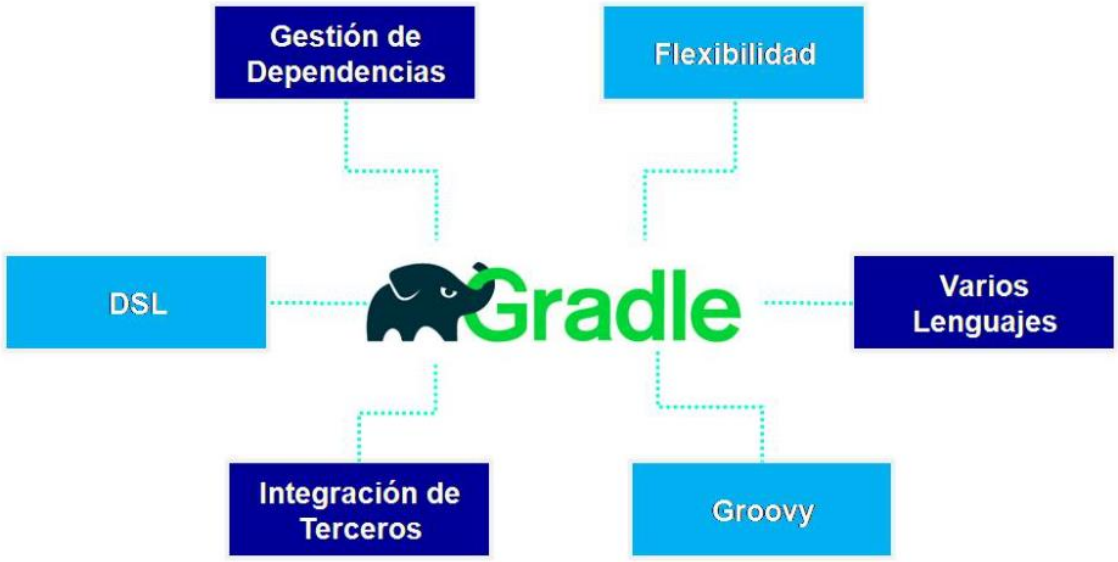
Cabe destacar que no existe la herramienta ni la técnica perfecta, al igual que puede haber más de una opción que pueda funcionar para resolver la situación planteada.

5.11.1. Gradle

“Gradle, es una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando Groovy o Kotlin DSL (Domain Specific Language)” (Muradas, 2023).

Figura 3

Características de Gradle.



Nota. Muradas, Y. (2023, 14 abril). *Qué es Gradle: La herramienta para ser más productivo desarrollando* [Imagen] OpenWebinars.net. https://openwebinars.net/blog/que-es-gradle/

Sus características principales se pueden apreciar en la Figura 3 y se enlistan a continuación:

- Depuración colaborativa.
- Construcción incremental.
- Diseño de repositorio personalizado.
- Dependencias transitivas.
- Soporte a Groovy y Scala incorporado.
- Compilación incremental para Java.
- Embalaje y distribución de JAR, WAR y EAR.
- Integración con Android Studio.
- Soporte de MS Visual C ++ y GoogleTest.
- Publicar en repositorios Ivy y Maven.
- TestKit para pruebas funcionales.
- Distribuciones personalizadas.

- Lee el formato POM.
- Compara *builds*.
- Compilador Daemon, es decir, compilador que se ejecuta en segundo plano.
- Personalizar y extender escaneos.
- Caché de dependencia de terceros.

5.11.2. Maven

"Apache Maven es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, como por ejemplo en todas las fases de compilación y empaquetado y la instalación de mecanismos de distribución de bibliotecas, para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo" (Yagüe, 2023).

Dentro de sus características se encuentran:

- Sistema de gestión dependencias.
- Mecanismo distribuido de distribución de bibliotecas.
- Plugins customizables.
- Multiplataforma.
- Es software libre.
- Fomenta la reutilización de código y de bibliotecas.
- Es compatible con múltiples IDEs.

5.11.3. Selenium WebDriver

"Un *framework* es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software. Son usados por programadores porque permiten acelerar el trabajo y favorecer que este sea colaborativo, reducir errores y obtener un resultado de más calidad" (Edix, 2021).

Sanchéz (2017) establece que Selenium es un conjunto de herramientas de código abierto que nos ayuda a automatizar acciones que un usuario puede realizar sobre aplicaciones

web. Cada herramienta dentro de este conjunto tiene un enfoque diferente para apoyar el proceso de automatización de pruebas.

“Permite a los evaluadores ejecutar pruebas en diferentes navegadores, plataformas y lenguajes de programación” (Dayal, 2021).

Se divide en 4 principales componentes:

- Selenium IDE.
- Selenium RC.
- Selenium WebDriver.
- Selenium Grid.

Del cual se aborda en este documento Selenium WebDriver como la herramienta de interacción con el navegador, establecido previamente por mis predecesores en el área QA.

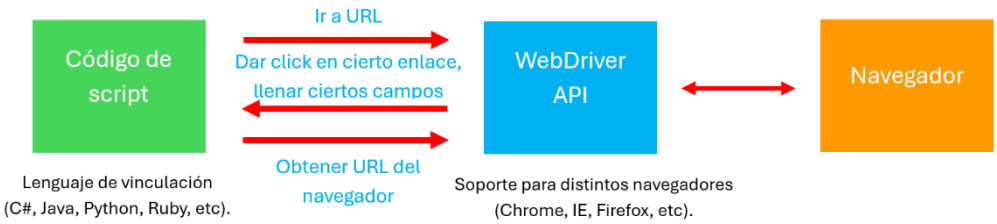
Tal y como lo menciona Unadkat (2024) Selenium WebDriver es un *framework* que le permite ejecutar pruebas en varios navegadores. Esta herramienta se utiliza para automatizar las pruebas de aplicaciones basadas en web para verificar que funcionen de manera esperada.

Selenium proporciona controladores específicos para cada navegador y, sin revelar la lógica interna de la funcionalidad del navegador, el controlador del navegador interactúa con el navegador respectivo estableciendo una conexión segura. Estos controladores de navegador también son específicos del lenguaje que se utiliza para la automatización de casos de prueba como C#, Python, Java, etc.

“A primera vista puede parecer que Selenium está manejando el navegador directamente desde nuestro código, sin embargo, este proceso es un poco más complejo de lo que pareciera. La arquitectura de WebDriver está dividida en tres partes principales: lenguaje de vinculación, WebDriver API y drivers” (Sánchez, 2017). Tal y como se observa en la Figura 4 de manera muy simple.

Figura 4

Arquitectura WebDriver



Nota Sánchez Mares, G. (2017, mayo). *Selenium WebDriver en un Ambiente de Pruebas Continuas* [Imagen]. SG, 54, 14-17. https://sg.com.mx/revista/54/selenium-webdriver-un-ambiente-pruebas-continuas

Básicamente lo que hacen los programas codificados, en nuestro caso en Java, es enviar peticiones a la API de WebDriver y este traduce los comandos interpretándolos y dando la sensación de que el navegador se maneja por sí solo.

Para cada script se tienen variables que sirven de identificadores para localizar los elementos distinguidos de la página web. Estas pueden ser reconocidas por 8 estrategias que proporciona el propio Selenium:

- Nombre de la clase.
- Selector CSS.
- Id.
- Nombre.
- Vínculo del texto.
- Vínculo del texto parcial.
- Etiquetas.
- Xpath.

Tomando de ejemplo el código HTML (Figura 5) presente en la documentación de Selenium WebDriver (Colaborador de Selenium, 2023):

Figura 5

Código ejemplo HTML

```html
<html>
<body>
<style>
.information {
  background-color: white;
  color: black;
  padding: 10px;
}
</style>
<h2>Contact Selenium</h2>

<form action="/action_page.php">
  <input type="radio" name="gender" value="m" />Male  
  <input type="radio" name="gender" value="f" />Female <br>
  <br>
  <label for="fname">First name:</label><br>
  <input class="information" type="text" id="fname" name="fname" value="Jane"><br><br>
  <label for="lname">Last name:</label><br>
  <input class="information" type="text" id="lname" name="lname" value="Doe"><br><br>
  <label for="newsletter">Newsletter:</label>
  <input type="checkbox" name="newsletter" value="1" /><br><br>
  <input type="submit" value="Submit">
</form>

<p>To know more about Selenium, visit the official page
<a href ="www.selenium.dev">Selenium Official Page</a>
</p>

</body>
</html>
```

Nota Colaborador Selenium. (2023). *HTML* [Imagen]. S https://www.selenium.dev/documentation/webdriver/elements/locators/

Se pueden obtener las siguientes localizaciones:

- Por nombre de la clase:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.className("information"));
```

- Por selector CSS:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.cssSelector("#fname"));
```

- Por id:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.id("lname"));
```

- Por nombre:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.name("newsletter"));
```

- Por vinculo de texto:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.linktext("Selenium Official Page"));
```

- Por vinculo de texto parcial:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.partialLinktext("Official Page"));
```

- Por etiqueta:

```
WebDriver driver = new ChromeDriver();

driver.findElement(By.tagName("a"));
```

- Por xpath:

```
WebDriver driver = new ChromeDriver();
```

```
driver.findElement(By.xpath("//input[@value='f']"));
```

5.11.4. Page Object Model

El POM es un patrón de diseño que consiste en separar las distintas clases JAVA que se automatizan para describir diferentes páginas de la aplicación Web bajo una misma prueba. Como ejemplo básico puede observarse la Figura 6.

Figura 6

Ejemplo POM con requerimientos mínimos

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Nota. Van Zyl, J., See, F. A. V., & Porter, B. (2009, 4 febrero). [Extracto código]. Maven – Introduction to the POM. https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

Estas clases contienen diferentes localizadores (encargados de identificar cada elemento de una página) y métodos (representan operaciones en la UI) que luego serán usados en una misma prueba.

Este patrón de diseño permite:

- Separar las operaciones y verificaciones. Para una mejor claridad al momento de escribir código.
- Generar código reusable. Un método se escribe una vez y se usa varias veces sin duplicar código.
- Identificar operaciones por el nombre de los métodos.

- Casos de prueba cortos y optimizados.
- Fácil mantenimiento. Al cambiar el comportamiento de la página, el método se adapta para cumplir con la operación sin que la prueba cambie.

5.11.5. IDE

Tal y como lo define Red Hat (2023), un entorno de desarrollo integrado (*integrated development environment* IDE por sus siglas en inglés) es un sistema de software para el diseño de aplicaciones que combina herramientas del desarrollador comunes en una sola interfaz gráfica de usuario (GUI). Generalmente, un IDE cuenta con las siguientes características:

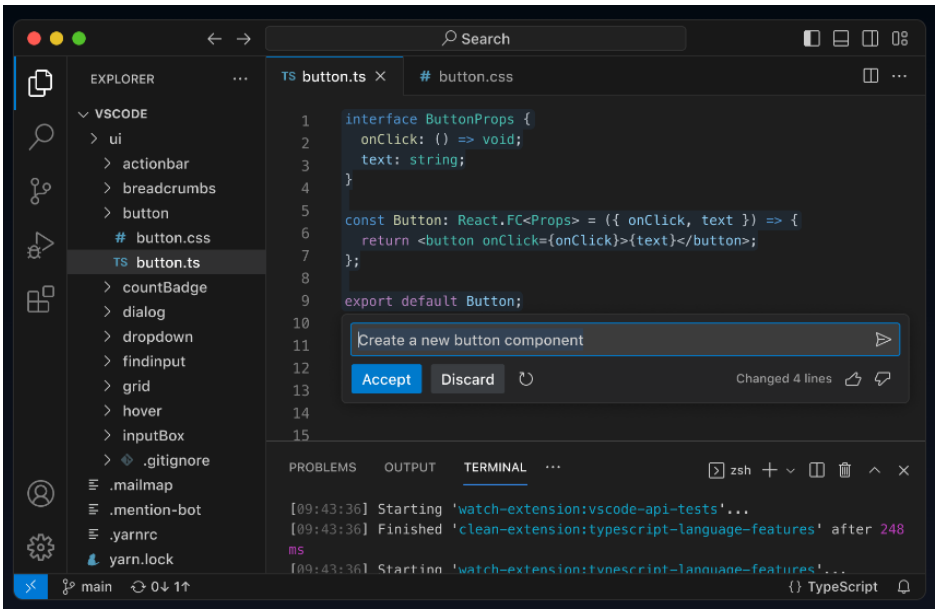
- **Editor de código fuente**: editor de texto que ayuda a escribir el código de software con funciones como el resaltado de la sintaxis con indicaciones visuales, el relleno automático específico para el lenguaje y la comprobación de errores a medida que se escribe el código.
- **Automatización de las compilaciones locales**: herramientas que automatizan las tareas sencillas y repetitivas como parte de la creación de una compilación local del software para que use el desarrollador, como la compilación del código fuente de la computadora en código binario, el empaquetado de ese código y la ejecución de pruebas automatizadas.
- **Depurador**: programa que sirve para probar otros programas y mostrar la ubicación de un error en el código original de forma gráfica

5.11.6. VSCode

Tal y como Flores (2023) menciona, Visual Studio Code (VS Code) es un editor de código fuente, véase Figura 7, desarrollado por Microsoft. Es software libre y multiplataforma, está disponible para Windows, GNU/Linux y macOS. VS Code tiene una buena integración con Git, cuenta con soporte para depuración de código, y dispone de un sinnúmero de extensiones, que básicamente te da la posibilidad de escribir y ejecutar código en cualquier lenguaje de programación.

Figura 7

Interfaz Visual Studio Code



Nota. Visual Studio Code - Code editing. Redefined. (2021, 3 noviembre). [Imagen]. https://code.visualstudio.com/

Dentro de sus funcionalidades se encuentran:

- Extensiones.
- Soporte de varios lenguajes de programación.
- Integración de IntelliSense y GitHub Copilot para una mejor codificación.
- Uso del control de versiones.
- Personalización de interfaz gráfica.
- Terminal integrada.
- Uso de herramientas de análisis de resultados integrado.

5.11.7. CI/CD

Red Hat (2022) explica que CI/CD es la sigla para la integración y la distribución o implementación continuas, cuyo objetivo es mejorar y agilizar el ciclo de vida de desarrollo del software.

La integración continua (CI) es una práctica que consiste en incorporar los cambios de código a un repositorio compartido de código fuente de forma automática y periódica. La distribución continua, o implementación continua, (CD) es un proceso de dos partes en el que se integran, prueban y distribuyen los cambios de código. Mientras que en la distribución los cambios no se llegan a implementar en la producción de forma automatizada, en la implementación sí se lanzan las actualizaciones en este entorno automáticamente. Sirva de referencia la Figura 8 de como es este flujo.

Figura 8

Flujo CI/CD

Nota: Red Hat. (s. f.). La integración y la distribución continuas (CI/CD) [Imagen]. https://www.redhat.com/es/topics/devops/what-is-ci-cd

En la práctica, los cambios que implementan los desarrolladores en la aplicación en la nube podrían ponerse en marcha unos cuantos minutos después de su creación (siempre que hayan pasado las pruebas automatizadas).

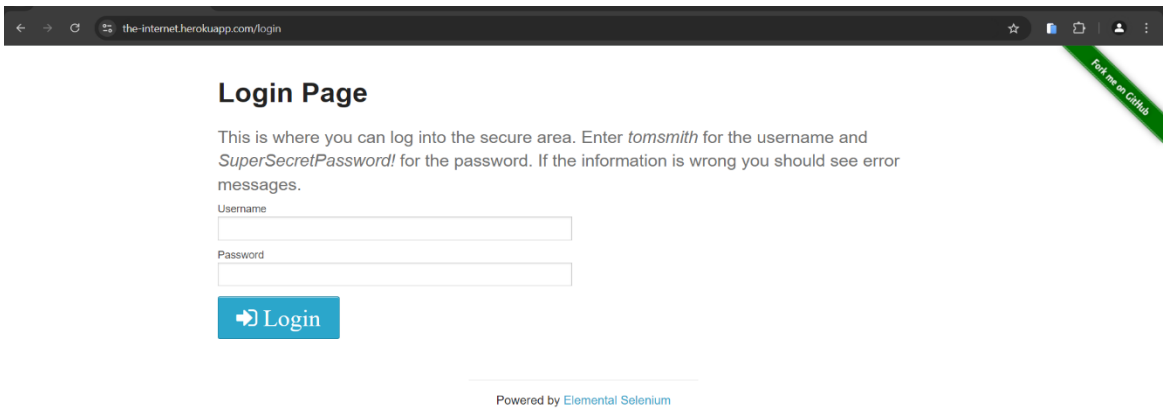
5.12. Ejemplo de prueba automatizada.

Como referencia se hace uso de la siguiente página web **https://the-internet.herokuapp.com/login** (Figura 9) para generar un código de interacción con Selenium WebDriver.

Tal y como se mencionó en la sección 5.8 Pruebas automatizadas el primer paso es visitar la página web a probar.

Figura 9

Página Demo



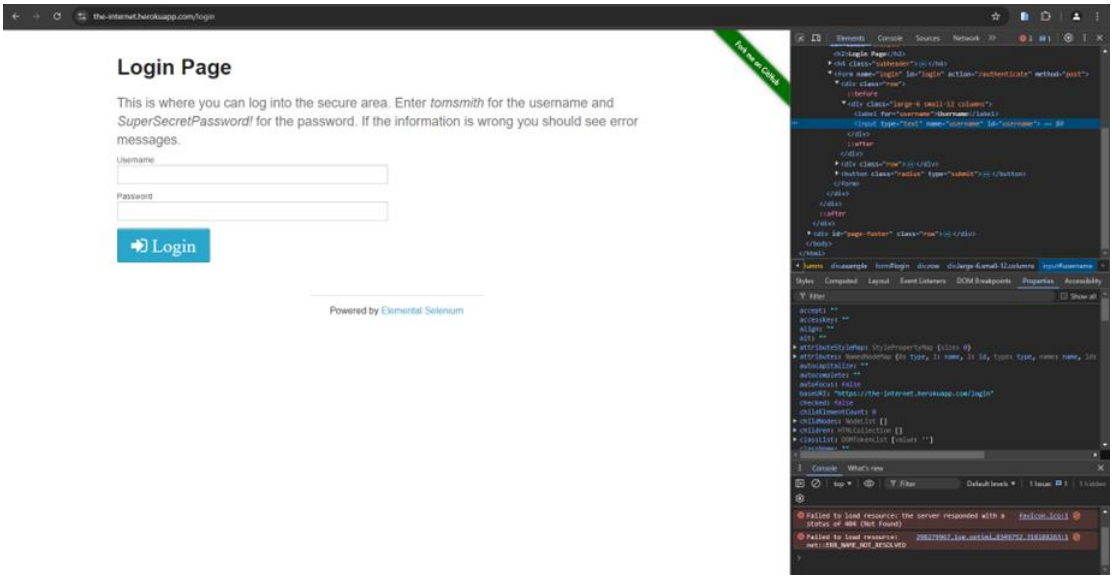
A continuación, se diseñan los pasos a seguir para probar la funcionalidad de ingresar al sitio escribiendo el usuario y contraseña correcto.

1. Abrir el navegador
2. Ingresar el sitio web deseado
3. Localizar y escribir el usuario
4. Localizar y escribir la contraseña
5. Oprimir el botón Login

Teniendo en cuenta los pasos anteriores, se procede a localizar los elementos HTML con ayuda del Inspector (Figura 10) de elementos presente en el navegador, aplicando la técnica mostrada en la sección 5.9.3 Selenium WebDriver.

Figura 10

Inspector de elementos



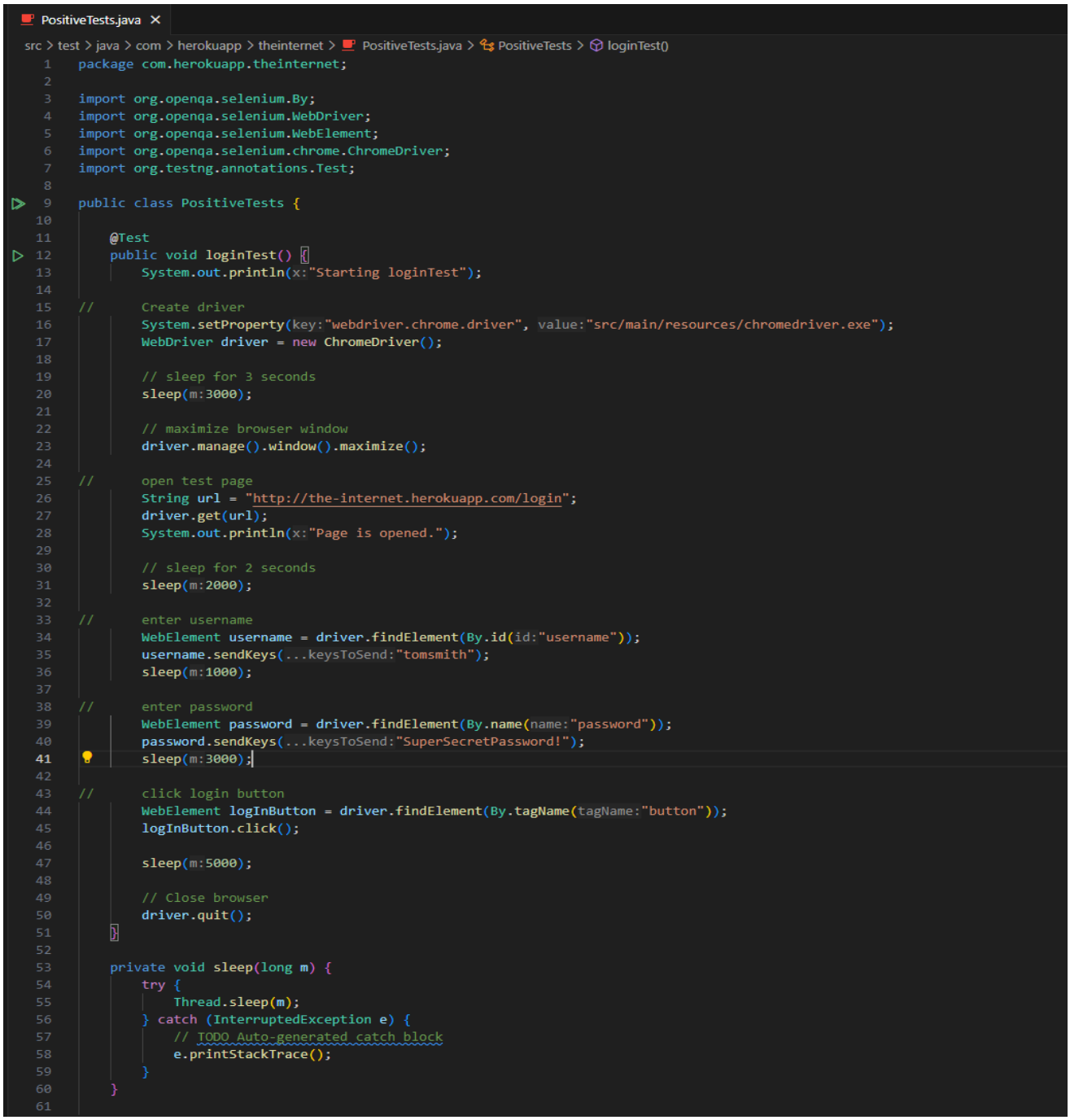
Teniendo los elementos ya localizados, se procede a generar el script con los pasos que seguirá Selenium WebDriver, véase Figura 11.

Primero se crea el objeto para controlar el *driver* del navegador, después se indica la URL a la que se quiere ingresar. Seguidamente, se generan las interacciones con la página escribiendo los WebElements localizados previamente y los valores que se les pasaran.

Se termina después de oprimir el botón de Login y se cierra el navegador.

Figura 11

Script demostrativo



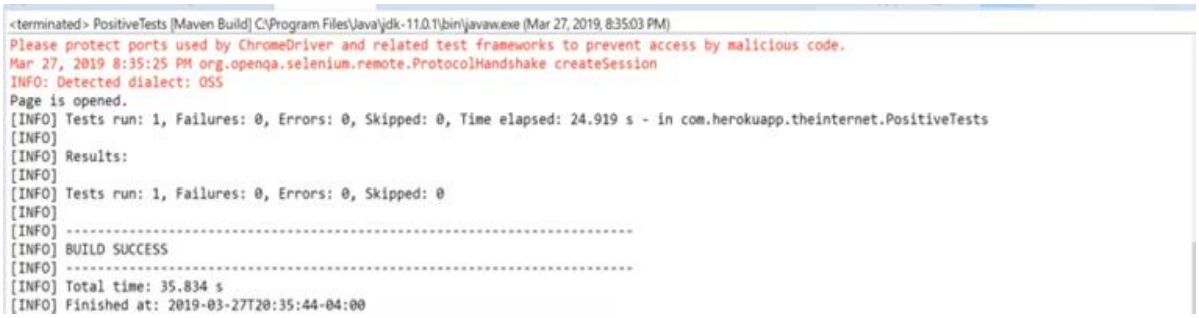
```java
package com.herokuapp.theinternet;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class PositiveTests {

    @Test
    public void loginTest() {
        System.out.println("Starting loginTest");

//      Create driver
        System.setProperty("webdriver.chrome.driver", "src/main/resources/chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        // sleep for 3 seconds
        sleep(3000);

        // maximize browser window
        driver.manage().window().maximize();

//      open test page
        String url = "http://the-internet.herokuapp.com/login";
        driver.get(url);
        System.out.println("Page is opened.");

        // sleep for 2 seconds
        sleep(2000);

//      enter username
        WebElement username = driver.findElement(By.id("username"));
        username.sendKeys("tomsmith");
        sleep(1000);

//      enter password
        WebElement password = driver.findElement(By.name("password"));
        password.sendKeys("SuperSecretPassword!");
        sleep(3000);

//      click login button
        WebElement logInButton = driver.findElement(By.tagName("button"));
        logInButton.click();

        sleep(5000);

        // Close browser
        driver.quit();
    }

    private void sleep(long m) {
        try {
            Thread.sleep(m);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

Nota: Dmitry Shyshkin. (s. f.). Selenium-WebDriver-with-Java-for-beginners [Extracto código]. https://github.com/dimashyshkin/Selenium-WebDriver-with-Java-for-beginners/tree/master/Section%203/Lecture%2017%20WebDriver%20commands/selenium-for-beginners

Para su ejecución se hace uso de la terminal y del comando **gradlle test** de donde, al finalizar, se tiene la salida de como resultó la prueba, como puede apreciarse en la Figura 12.

Figura 12

Terminal



Nota: Dmitry Shyshkin. (s. f.). Selenium-WebDriver-with-Java-for-beginners [Imagen]. https://www.udemy.com/course/selenium-for-beginners/

6. Desarrollo.

6.1. Requerimientos.

Con el propósito de solucionar los problemas planteados en la sección Definición del problema y atendiendo a la necesidad de parte de la empresa de mantenerse a la vanguardia en el área de QA, se realizará la actualización del proyecto de automatización.

Debe cumplir con los siguientes puntos:

- Actualizar las dependencias usadas en el *software*.
- Debe ser fácil de entender tanto para los nuevos integrantes que se incorporan a la fuerza de trabajo de QA, como los más experimentados en el área de automatización ajenos a este proyecto en específico.
- Reducir el uso de 2 programas (uno para codificar y el otro para ejecutar Selenium WebDriver) a uno solo, manteniendo la misma funcionalidad.
- Actualizar y mantener a futuro la documentación relacionada con el proyecto.
- Tener un software capaz de ajustarse al flujo CI/CD que en un futuro se implementará.
- Mantener Selenium WebDriver como la herramienta de interacción con el navegador y Java como lenguaje de programación.
- Modificar la programación de los códigos de programación que sean necesarios para que se adapten a la nueva configuración.
- Diseñar, desarrollar e implementar nuevos *scripts* para el correcto funcionamiento de la actualización.

6.2. Análisis.

El presente trabajo abarca la etapa de pruebas dentro del ciclo de vida del software del producto de la empresa, en el cual este tendrá su propio ciclo de software para su debida actualización.

Con el tiempo, habiendo realizado algunos cursos afines con la automatización de Selenium WebDriver y tras un periodo de investigación sobre varias herramientas que se ocupan con

este, se observó que las utilidades usadas no eran la mejor opción, como se explicará más adelante.

Esto llevo a discutir con los líderes la posibilidad de encontrar las herramientas necesarias que hicieran esta labor de manera eficaz, ya que ellos le dan relevancia para entregar los productos si ya están verificados de manera correcta. Con esto en mente, y la debida aprobación en el proyecto, se inició con la tarea de identificar los puntos clave que impedían hacer su trabajo al *QA tester* más rápido, todo con la ventaja de tener libertad creativa para hacer la toma de decisiones por mí mismo orientando los resultados a lo que mejor le podría convenir al área de QA.

Inicialmente, el proyecto estaba construido con *Maven* como su gestor de proyectos, el cual tenía la dificultad de entender el archivo POM. Y es que mientras el proyecto continuara creciendo, la dificultad para leer rápidamente este archivo se volvía una tarea más complicada. Sirva de ejemplo del archivo POM la Figura 13.

Tal y como se menciona en *Gradle vs. Maven: Performance, Compatibility, Builds, & More* aquí se tiene una definición del archivo POM para compilar, realizar análisis estático, ejecutar pruebas unitarias y generar los archivos JAR.

Figura 13

Ejemplo de código POM.xml

```
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.12.1</version>
    <executions>
        <execution>
            <configuration>
                <configLocation>config/checkstyle/checkstyle.xml</configLocation>
                <consoleOutput>true</consoleOutput>
                <failsOnError>true</failsOnError>
            </configuration>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>findbugs-maven-plugin</artifactId>
    <version>2.5.4</version>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-pmd-plugin</artifactId>
    <version>3.1</version>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Nota. Altvater, A. (2024b, marzo 4). *Gradle vs. Maven: Performance, Compatibility, Builds, & More* [Extracto código]. Stackify. https://stackify.com/gradle-vs-maven/

Por otro lado, el código para tener la misma respuesta en Gradle (Figura 14) es el siguiente:

Figura 14

Ejemplo de código build.gradle

```
apply plugin:'java'
apply plugin:'checkstyle'
apply plugin:'findbugs'
apply plugin:'pmd'
version ='1.0'
repositories {
    mavenCentral()
}
dependencies {
    testCompile group:'junit', name:'junit', version:'4.11'
}
```

Nota. Altvater, A. (2024b, marzo 4). *Gradle vs. Maven: Performance, Compatibility, Builds, & More* [Extracto código]. Stackify. https://stackify.com/gradle-vs-maven/

Al apreciar la diferencia en tamaño de líneas de código requeridas para expresar los mismo en ambas Figuras (13 y 14), es notorio que Gradle resulta ser el vencedor. Otro rasgo por destacar es la legibilidad que proporciona el script de compilación de Gradle en comparación al de Maven que hace uso de XML y esto permite que los nuevos integrantes puedan leer el archivo sin mayores complicaciones, asimismo, visualmente no genera un agobio al usuario por lo nuevo que tiene que aprender.

Teniendo en cuenta estos factores se decidió en cambiar la herramienta de compilación Maven por Gradle. Al mismo tiempo, y ya tomando la decisión de cambiarlo, se notó que Gradle podía hacer uso de subproyectos para tener bajo una misma instancia varios proyectos que podrían compartir utilidades, algo que resultaría ventajoso para el futuro ya que teníamos varios automatizadores que tenían el mismo funcionamiento. Lo cual llevo a realizar una actualización en los 3 códigos más importantes.

Ahora bien, a causa del cambio en la herramienta de compilación se observó una mejora en la respuesta de velocidad al compilar el proyecto. A continuación, se muestra una tabla de medición tomada a partir de la comparación de ambos *frameworks*.

Tiempo de compilación

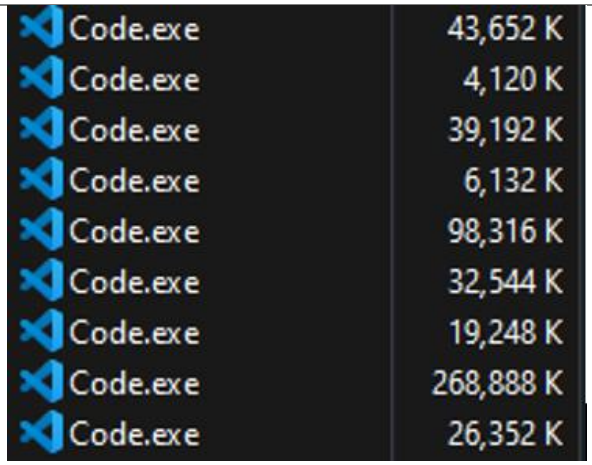
Intento	Maven	Gradle
1° compilación	25 segundos	23 segundos
2° compilación	24 segundos	9 segundos
3° compilación	26 segundos	13 segundos
4° compilación	25 segundos	12 segundos
5° compilación	27 segundos	9 segundos

En definitiva, se notó una mejora en el uso de Gradle y su *deamon* tras la primera compilación. Una reducción de tiempo que puede ayudar a revisar los casos de prueba codificados mucho más rápido.

En cuanto al uso del IDE Eclipse y tomando en consideración las características que se deben valorar para tener las mejores herramientas para el desarrollo de pruebas automatizadas se notaron puntos en los que se podría mejorar. Son los siguientes:

- No era tan fácil de utilizar debido a su complejidad.
- Personas inexpertas con el IDE podrían tener problemas de adaptación al principio.
- Mayor uso de recursos de la computadora.
- Solo se ocupaba para correr el automatizador. Las pruebas se codificaban en un editor de textos, lo que llevaba a tener 2 programas para hacer uso del proyecto de software.
- Interfaz de usuario (GUI) poco amigable.

Con el propósito de verificar el uso de recursos por Eclipse y VS Code se tiene la siguiente tabla:

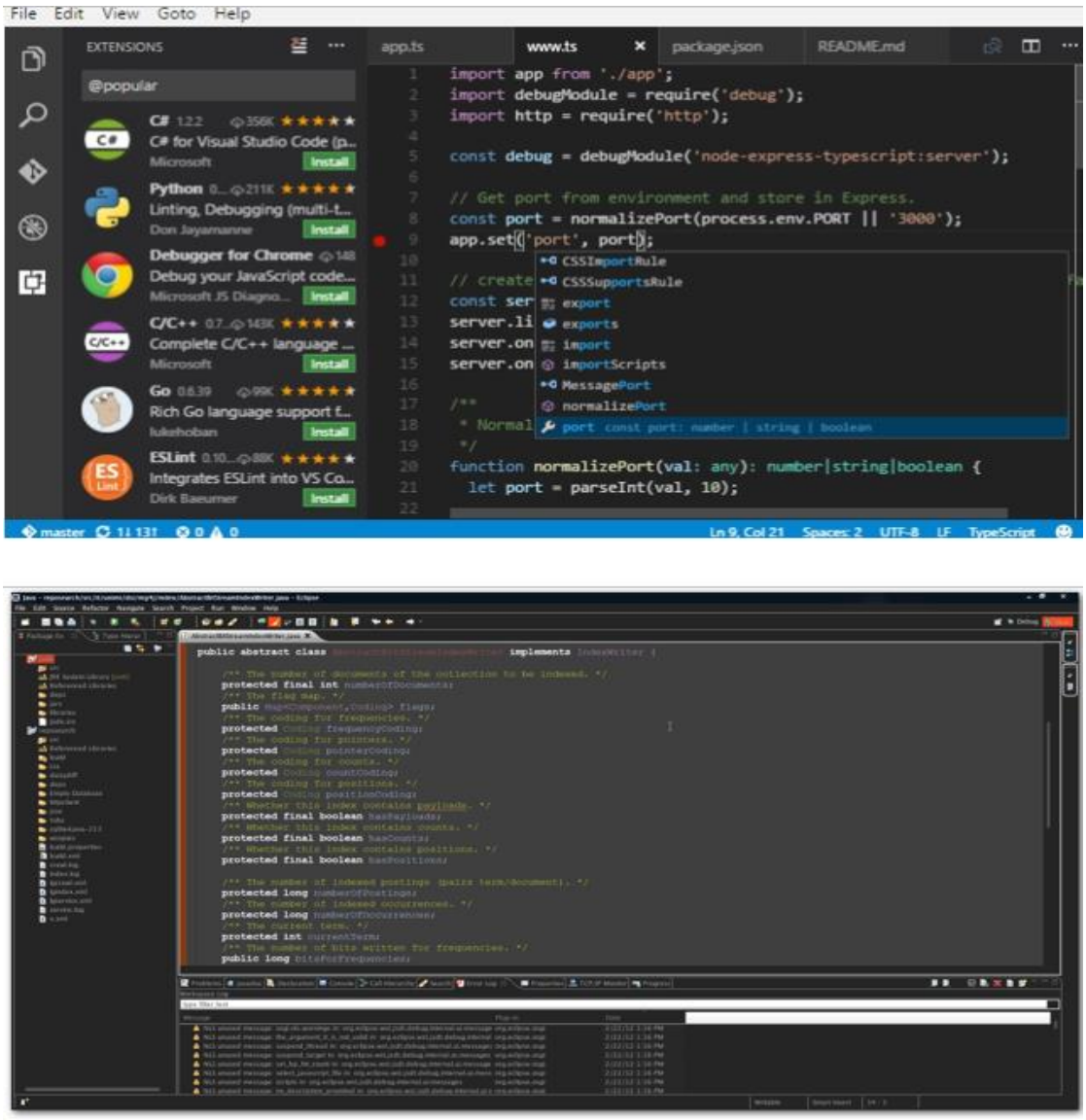
	Eclipse	VS Code
Tiempo apertura	165 segundos	67 segundos
Recursos consumidos		

De lo cual se obtuvo un mayor tiempo en la carga del programa Eclipse (tomando en cuenta la carga del proyecto a codificar) en comparación contra VS Code. Además, en la medición de recursos que le toma a la computadora hacer uso de estos softwares resulta de nuevo ganador VS Code con un total de 538.444 Kb. Indiscutiblemente, el editor de textos de Microsoft resultaba una mejor opción en cuestión de recursos de hardware. Aunque no es un tamaño en memoria alarmante, puede llegar a ser importante cuando se considera que en adición usa más herramientas y utilidades necesarias para la compilación del proyecto.

En cuanto al resto de puntos considerados anteriormente, la Interfaz de usuario poco amigable, complejo en su uso, que las personas inexpertas con el IDE podrían tener problemas de adaptación al principio y que se tenían dos programas, uno para compilar y el otro para editar el texto, se puede apreciar en la Figura 15 como se tiene una interfaz más amigable con VS Code en contraste de Eclipse que parece anticuado y saturado de iconos los cuales solo conoce el más experimentado usando esta herramienta.

Figura 15

Comparación GUI VS Code vs Eclipse IDE



Nota. Get App. (s. f.). Visual Studio Code Y Eclipse IDE [Imagen]. https://www.getapp.es/compare/2035587/2050477/visual-studio-code/vs/eclipse-ide

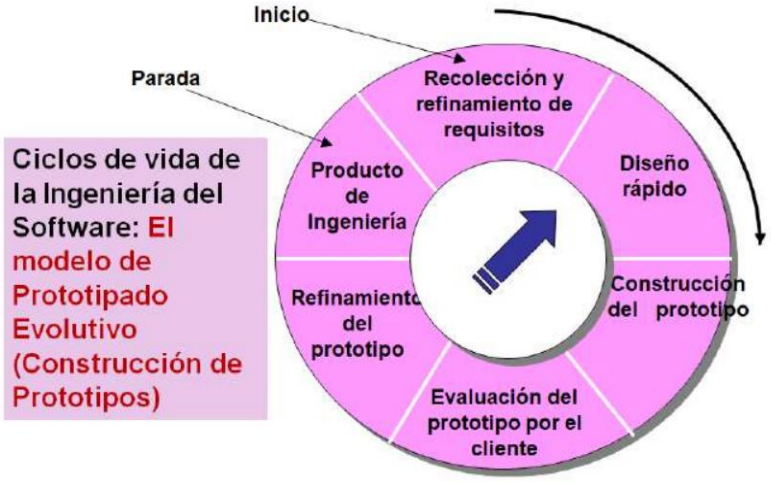
Por lo cual, se optó por hacer uso de VS Code que puede llegar cubrir las necesidades que requiere tener un IDE, tal y como se mencionó en secciones anteriores.

6.3. Metodología.

Se optó por hacer uso de la metodología de prototipos esto debido a que este modelo implica desarrollar una versión inicial del programa, presentarla como una posible solución, recibir retroalimentación de parte del usuario y perfeccionarla a través de múltiples iteraciones hasta construir el proyecto que más se alinee a los requisitos planteados anteriormente. En caso de que la respuesta sea negativa para el resultado propuesto, se debe replantear la solución y crear una nueva iteración con los mismos pasos para la creación de este prototipo. A continuación, la Figura 16 muestra de manera gráfica como es esta metodología.

Figura 16

Prototipado evolutivo



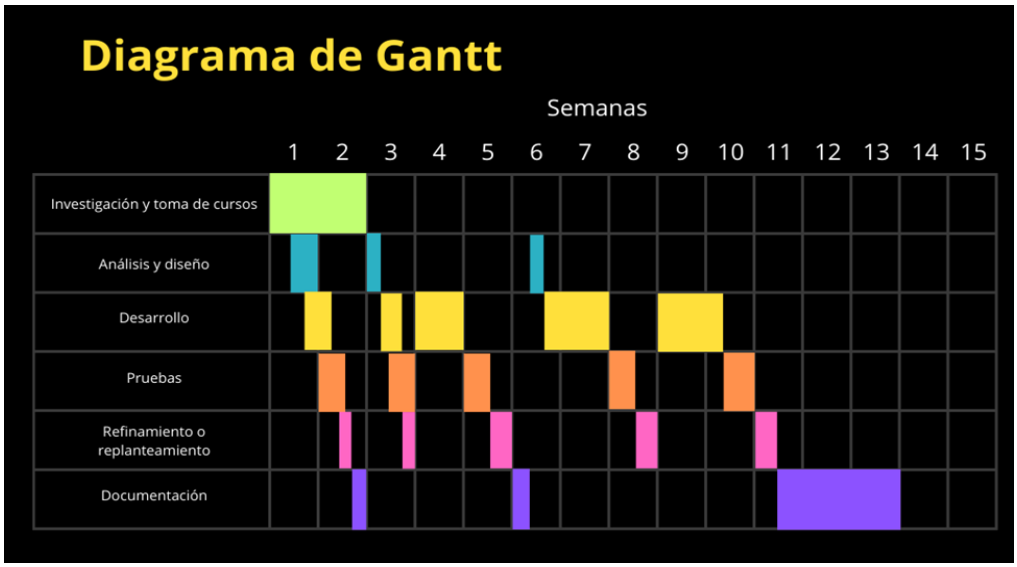
Nota. Del Carmen Gomez Fuentes, M., Ojeda, J. V., & Pérez, P. P. G. (2019). *Fundamentos de ingeniería de software* [Imagen]. En CUA UAM. Universidad Autónoma Metropolitana, Unidad Cuajimalpa. http://ilitia.cua.uam.mx:8080/jspui/handle/123456789/1000

Otro elemento por considerar en este desarrollo es el tiempo. Se ha planteado una duración de aproximadamente 13 semanas considerando los requerimientos y la nula experiencia

para realizar esta actualización. Esto puede observarse en el siguiente diagrama de Gantt de la Figura 17.

Figura 17

Diagrama de Gantt del proyecto



Tal y como puede observarse, durante la etapa de investigación y estudio de los cursos relacionados con la automatización en Selenium WebDriver, se pudo apreciar que en particular uno de los instructores hacia uso de Visual Studio Code como editor de textos y, al mismo tiempo, lo usaba para ejecutar este código, en contraste con la mayoría hacia uso del IDE Eclipse. Esto conllevo a ver la posibilidad de tener la misma configuración para este proyecto. Se desarrollo la idea a partir de las bases sentadas en el curso y con esto se tenía el primer prototipo de este proyecto.

A continuación, se inició una nueva iteración de análisis y diseño para poder actualizar Maven a partir del primer prototipo y que este ofreciera una mejor legibilidad, de nuevo, con ayuda de la investigación e información de los cursos.

Tras un periodo de prueba y error bajo esta mentalidad, se notó que la mejor solución era rediseñar el proyecto con una nueva herramienta de automatización: Gradle. Esto llevó a

hacer una nueva iteración de desarrollo, dejar el segundo prototipo como un fallo y replantear la solución para un tercer prototipo tomando como base el primero de estos.

Seguidamente, la actualización a Gradle condujo al tercer prototipo como exitoso pero que aún no era funcional ya que había que actualizar las dependencias y partes del código, como el POM, que eran exclusivas para Maven.

Finalmente, teniendo el cuarto prototipo listo y funcional, se decidió configurar el programa para futuros proyectos y con esto plantear la creación de un sistema multi proyecto, por esta razón durante esta etapa se hizo la corrección de códigos para lograr la tarea. En consecuencia, se tuvo un quinto prototipo funcional.

7. Resultados obtenidos.

Al hacer el cambio de la herramienta que permite la automatización de compilación del código de Maven a Gradle se obtuvo una mejor legibilidad en la organización del proyecto, ahora llamado *build.gradle.*

Este archivo se encuentra en la raíz del proyecto, este se encarga de administrar las dependencias para todos los subproyectos. Su contenido tiene la siguiente estructura (Figura 18):

Figura 18

Definición y estructura básica del archivo build.gradle

my-app/app/build.gradle

```
plugins {
    id 'application'
}

application {
    mainClass = 'org.sample.myapp.Main'
}

dependencies {
    implementation 'org.sample:number-utils:1.0'
    implementation 'org.sample:string-utils:1.0'
}
```

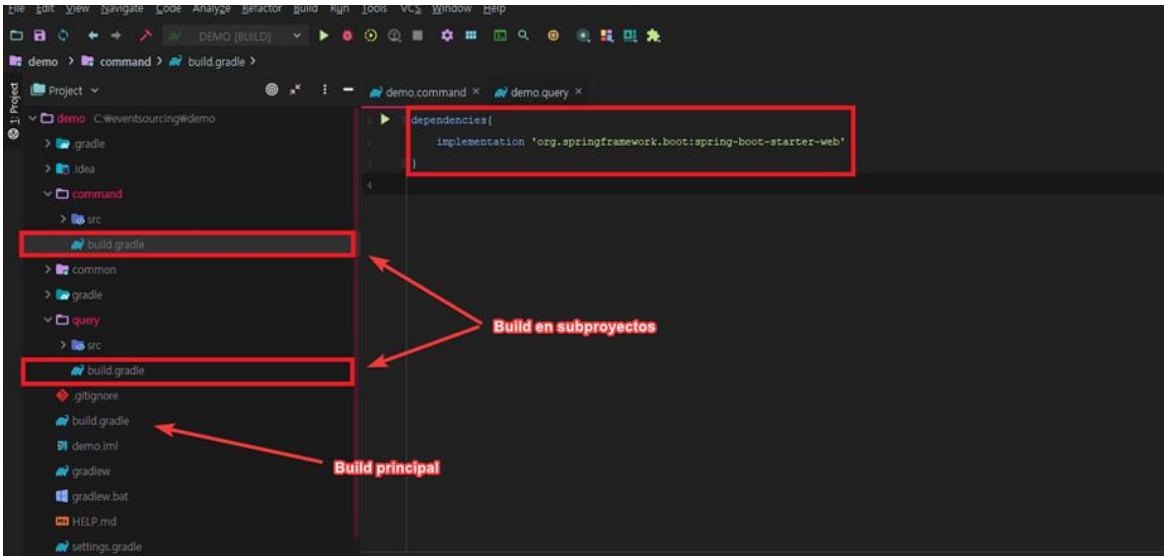
Nota. Gradle Documentation. (2023). [Extracto código]. https://docs.gradle.org/current/samples/sample_composite_builds_basics.html

De igual manera, cada subproyecto puede almacenar su propio administrador de dependencias para hacer manejo de sus propias bibliotecas y crear sus propias *task,* que permiten crear partes de código para la ejecución del programa o para tareas específicas, como eliminación de archivos temporales.

Es en la Figura 19 donde se puede apreciar que existen 2 subproyectos bajo un mismo proyecto.

Figura 19

Estructura básica de un multiproyecto gradle



Nota. Cla. (2019, 25 diciembre). *Gradle Multi Project* [Imagen]. https://cla9.tistory.com/7

En adición a lo anterior, se logró que el proyecto sea de más fácil entendimiento, debido a la simplificación del proyecto y de la utilización de un lenguaje más natural en los build.gradle, para los nuevos *testers* que inicien en este ámbito de automatización.

Por otra parte, se mejoraron los tiempos de compilación del programa tal y como se revisó en la sección de Análisis, debido a que Gradle hace uso de un *deamon* para compilar más rápido. Esto es especialmente útil debido a los cambios constantes en el código que debe ser probado inmediatamente.

También se hizo un cambio de la herramienta de codificación del IDE Eclipse a Visual Studio Code debido a que se presentaron errores de ejecución del IDE en los que el programa no respondía a la interacción del usuario, esto debido a las limitantes de la computadora huésped. Situación que se corrigió al modificarlo por el programa de Microsoft, además de

que este editor de texto ya se ocupaba para realizar los cambios al código, solo le hacía falta configurarlo y agregar extensiones para potenciar su utilidad y hacer uso de este como un IDE.

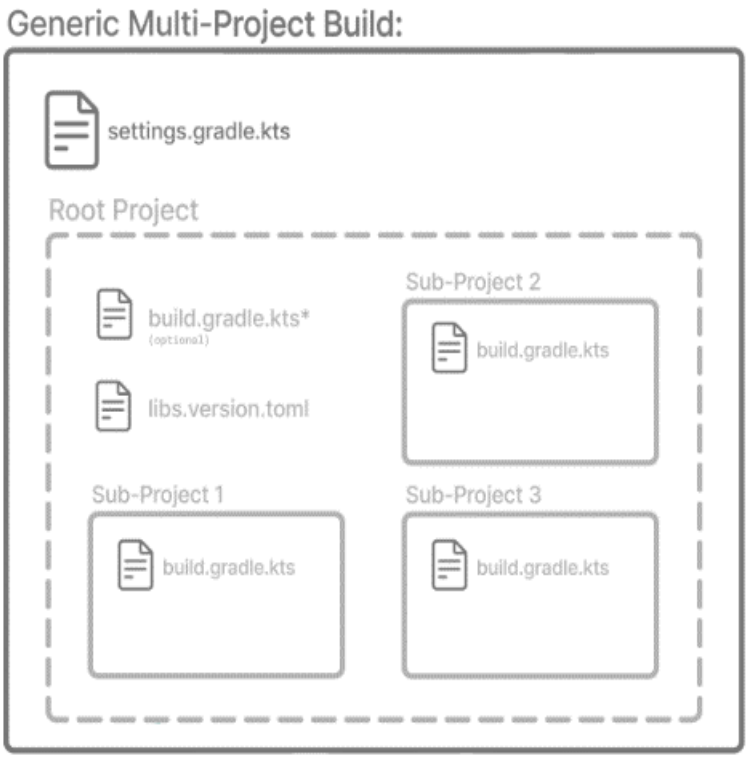
Esto resulto en que se eliminaron las dependencias a tener que usar el Eclipse para gestionar los paquetes de Maven, centrándose en la ejecución y el desarrollo del código Java sobre Visual Studio Code. Ahora se tiene una respuesta de interacción más rápida y ocupando menos recursos de la computadora. De igual forma, puede revisarse la sección de Análisis donde se muestra esta comparación.

Debido a que fue tan exitoso el proyecto, se lograron agregar 2 proyectos más y ahora se compone de 3 módulos que representan diferentes proyectos bajo una misma estructura, esto con la finalidad de que vivan y utilicen las mismas herramientas entre instancias. Esto permite que cada proyecta tenga sus propias características y que se puedan ejecutar de manera independiente.

Tal y como puede apreciarse en la Figura 19, la solución planteada presenta similitudes con el proyecto de ejemplo donde se tienen 3 subproyectos y cada uno tiene su propio archivo *build.gradle*.

Figura 20

Estructura básica de un multiproyecto gradle



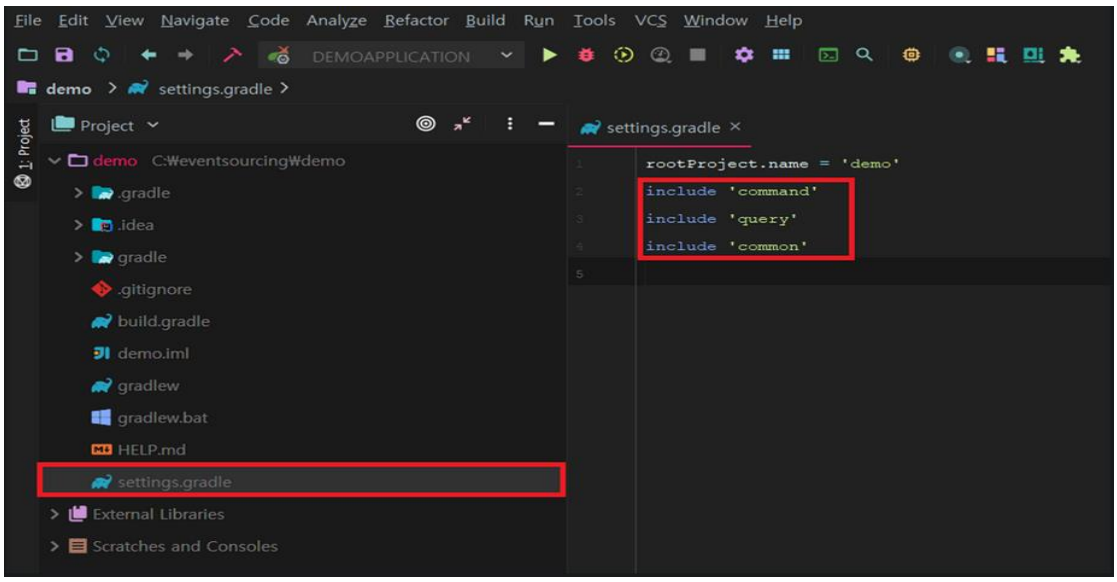
Nota. Gradle Documentation. [Imagen] (2023).
https://docs.gradle.org/current/userguide/intro_multi_project_builds.html

Existe un archivo llamado *setting.gradle* que se encarga de reunir los subproyectos en un proyecto principal y detallar como deben encontrarse estos dentro del programa. Como referencia ver Figura 20.

Cabe recalcar que el nombre de proyecto padre y de los subproyectos deben coincidir con los nombre asignados en las carpetas, com muestra se tiene la Figura 21.

Figura 21

Estructura básica de un multiproyecto gradle



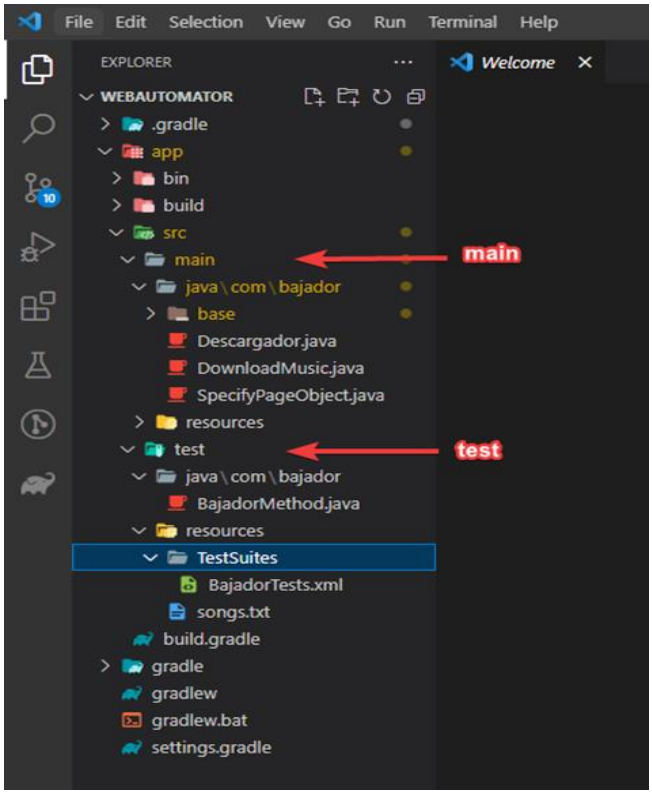
Nota. Cla. (2019, 25 diciembre). *Gradle Multi Project* [Imagen]. https://cla9.tistory.com/7

Además, se actualizaron las dependencias ocupadas en la pieza de software, teniendo una mejor organización de estas para futuros cambios y el código generado, así como los métodos, clases y variables modificadas fueron comentados haciendo uso de JavaDoc.

Para entender mejor, los proyectos tienen una estructura que divide las clases del patrón de diseño POM, en la carpeta *main* y, donde se escriben los casos de prueba, en la carpeta *test*, tal y como se ve en la Figura 22.

Figura 22

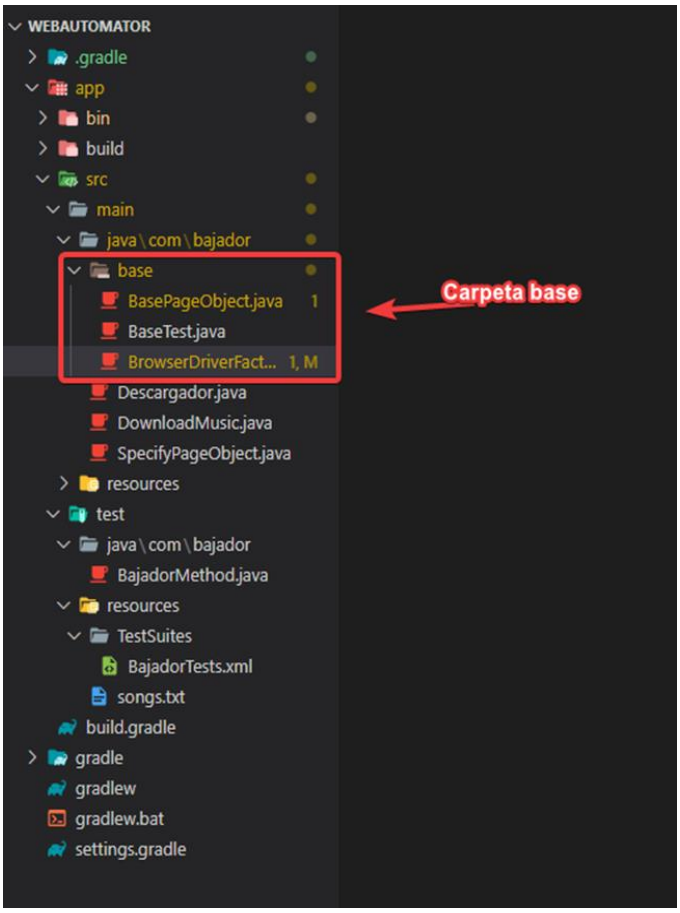
División del proyecto en carpetas



Dentro del proyecto se presenta una carpeta que se nombra *base* la cual contiene las clases y sus métodos que se repiten con frecuencia. Es en estos archivos donde se realizaron cambios para agregar los subproyectos (véase Figura 23).

Figura 23

Carpeta base

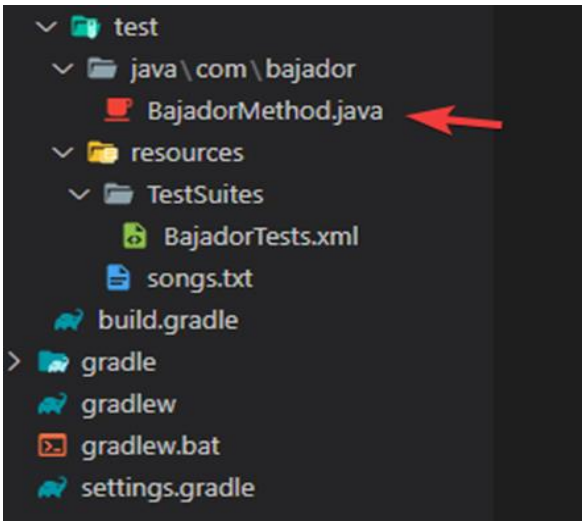


De igual importancia, se tiene la clase que contendrá todas las acciones que se realizarán para completar una tarea, como por ejemplo el realizar la exploración de un numero de orden dentro del servicio web utilizado la barra de búsqueda y filtrando los resultados con algunos parámetros, como la fecha de creación.

Esta clase se encuentra contenida dentro de la carpeta test.

Figura 24

Caso de prueba



Es en estos archivos (Figura 24) donde también se revisaron y, en caso de ser necesario, se generaron cambios, como más adelante se expresa.

Y, por último, se tiene un XML que sirve para indicarle al compilador que archivos de los casos de prueba deben ejecutarse, además de que pueden agregarse parámetros de configuración para clase de prueba que contiene todos los métodos a ocupar, vease Figura 25.

Figura 25

Test Suite

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="BajadorTest" verbose="1">
    <parameter name="browser" value="Chrome" />
    <parameter name="operatingSystem" value="WINDOWS" />

    <test name="DemoBajador">
        <classes>
            <class name="com.bajador.BajadorMethod"/>
        </classes>
    </test>

</suite>
```

Aquí se realizó una división por flujos de trabajo que se tienen dentro de la compañía para tener una mejor legibilidad y organización para usar los casos de prueba.

Teniendo en cuenta las actualizaciones realizadas anteriormente, así como los cambios realizados en los archivos en la carpeta *base*, y para garantizar el correcto funcionamiento del proyecto se llevaron a cabo actualizaciones en ciertos códigos con sus debidas pruebas. Algunos de estos *scripts* forman parte de casos de pruebas de los cuales fueron 349. A continuación, se muestra una tabla de los resultados obtenidos.

Prioridad	Numero de pruebas	Exitosos	Fallados
Alta	106	76	30
Media	83	70	13
Baja	160	99	61
Total	349	245	104

De lo anterior, cerca del 70.20% de las pruebas se conservaron sin ningun cambio, por el contrario el resto tuvo que ser rediseñado ya que al fallar se necesito de un mayor análisis para solucionar esos errores que fueron desde cambios en el servicio web provocando que los elementos buscados ya no existieran, actualización de como se obtenian los elementos o adición de pasos adicionales para encontrarlo.

Por último, se destaca el haber cumplido con dos requerimientos vitales para esta actualización que son mantener Selenium WebDriver como la herramienta de interacción con el navegador y Java como lenguaje de programación, así como conservar algunos de los scripts automatizados ya creados con anterioridad más algunos cambios en algunos otros.

8. Conclusiones y trabajo a futuro.

Resulto todo un reto lograr codificar este automatizador debido a la oportunidad de completar este desafío durante una época en la que seguía la pandemia y que implicó un mayor de esfuerzo y disciplina para poder adquirir todos los conocimientos necesarios para cambiar este proyecto.

Para llevar a cabo estas actividades se tomaron en consideración los conocimientos, habilidades y técnicas obtenidas durante el transcurso de la carrera de Ingeniería en computación.

En adición a esto, con la creación de código nuevo y la actualización del que ya está presente, se reconoció la importancia de lo que siempre se recalcaba en la universidad; como llevar a cabo buenas prácticas de programación, tales como escribir las variables y constantes con nombres legibles y relacionados con lo que hacen, crear una buena documentación para los futuros compañeros que formen parte del equipo y estos puedan reconocer fácilmente como es que funcionan los métodos escritos proporcionando una descripción de lo que realiza ese método, precondiciones que se necesitan para llevar a cabo esa tarea, así como las entradas y salidas de aquel método. Y no solo dentro del código, sino fuera de este teniendo páginas que describan el proceso por el cual se pasó, como puede utilizarse y los cambios por hacer.

El impacto de este desarrollo de software a nivel negocio de la empresa fue que durante los lanzamientos de producto donde se requiere la intervención de los *QA testers* para las pruebas hacia ambientes de producción se redujo el tiempo que estas sesiones tomaban.

Algo similar ocurre cuando se realiza pruebas en el ambiente de desarrollo.

A causa de la reducción de tiempo en la ejecución de algunas pruebas automatizadas, se puede mantener la plantilla actual del área de QA focalizada en tareas más importantes sin tener que contratar más personal, reduciendo los costos para la empresa.

También, para las nuevas incorporaciones de personal al plantel de QA de este producto se logró reducir el tiempo de asimilación de cómo hacer uso de la herramienta de automatización por la actualización hacia un *framework* más legible y fácil de comprender.

Como beneficio adicional, se evitó realizar la compra de nuevo equipo de cómputo para actualizar el hardware que proporcionaron desde el principio debido al cambio del IDE Eclipse a Visual Studio Code.

Ahora teniendo las pruebas automatizadas actualizadas el área de *QA* puede enfocarse en tareas más prioritarias dejando el trabajo repetitivo a los scripts, , que no aporta mucho al *tester*, dejando en claro que esto no significa dejar que la computadora haga todo el trabajo ya que se necesita la intervención humana para interpretar los errores y situaciones inesperadas de las pruebas automatizadas, así como las nuevas tareas que surgen día con día en las que no se tiene el desarrollo en Selenium.

De cara al futuro de este proyecto, se prevé agregar dos proyectos API y mobile, conjuntamente se pretende agregar el proyecto a un flujo *CI/CD*.

Finalmente, hemos notado que la actualización le ha sentado bien al proyecto debido a la recién adición de nuevas herramientas integradas por la empresa, que no estaban contempladas, pero que han sido fácil de integrar. Asimismo, considero que este desarrollo de software puede seguir creciendo más, aún quedan un par de proyectos dentro de la empresa que deben adicionarse a esta pieza de software, casos de pruebas que deben concluirse para tener un ambiente de pruebas automático más completo y demás proyectos que se agreguen al servicio de la empresa que necesiten una implementación de automatización. Además, esto me ha impulsado para seguir investigando y explorando más herramientas, así como técnicas que puedan contribuir a mi crecimiento como *QA tester*.

9. Referencias

Alarcón, J. M. (1 de junio de 2022). *Campus MVP*. Obtenido de Java: ¿Qué es Maven? ¿Qué es el archivo pom.xml? : https://www.campusmvp.es/recursos/post/java-que-es-maven-que-es-el-archivo-pom-xml.aspx

Altvater, A. (04 de marzo de 2024). *Gradle vs. Maven: Performance, Compatibility, Builds, & More*. Obtenido de Stackify: https://stackify.com/gradle-vs-maven/

Armenta Benitez, B., Rodriguez Espinoza, I., Medina Muñoz, L. A., & Gonzalez López, S. (2018). Aplicación del modelo de prototipos: Caso de estudio Software RedbotGamesShop. *Revista de Simulación Computacional, 2*(5), 8-13.

Colaborador Abstracta. (15 de febrero de 2024). *Testing manual vs. automatizado: ¿cuál elegir para tu proyecto?* Obtenido de Blog de Desarrollo de Software, Testing e Inteligencia Artificial: https://es.abstracta.us/blog/testing-manual-vs-testing-automatizado/

Colaborador de iwantic. (16 de marzo de 2024). *QA TESTER*. Obtenido de https://iwantic.com/especializaciones/big-data/que-es-un-qa-tester/

Colaborador de QAlified. (06 de mayo de 2023). *¿Qué hace un QA Tester? Descripción completa del trabajo*. Obtenido de https://qalified.com/es/blog/que-hace-un-qa-tester/

Colaborador de Red Hat. (11 de mayo de 2022). *La integración y la distribución continuas (CI/CD)*. Obtenido de Red Hat: https://www.redhat.com/es/topics/devops/what-is-ci-cd

Colaborador de Selenium. (3 de abril de 2023). *Selenium*. Obtenido de Locator strategies: https://www.selenium.dev/documentation/webdriver/elements/locators/

Colaborador de Smartsheet Latam. (04 de marzo de 2024). *Diferencias entre metodologías ágiles y tradicionales: ventajas y desventajas*. Obtenido de Linkedin: https://www.linkedin.com/pulse/diferencias-entre-metodolog%C3%ADas-%C3%A1giles-y-tradicionales-ventajas-/?originalSubdomain=es

Colaborador de UCMA. (20 de noviembre de 2023). *¿Cuáles son las mejores metodologías de desarrollo de software?* Obtenido de UCMA: https://www.universitatcarlemany.com/actualidad/blog/metodologias-de-desarrollo-de-software/

Colaborador de ZAPTEST. (08 de julio de 2022). *ZAPTEST*. Obtenido de ¿Qué es la automatización de pruebas? Una guía sencilla y sin jerga: https://www.zaptest.com/es/que-es-la-automatizacion-de-pruebas-una-guia-sencilla-y-sin-jerga

Colaborador QAlified. (12 de noviembre de 2023). *Aprende las diferencias entre pruebas manuales y pruebas automatizadas*. Obtenido de QAlified Building Quality: https://qalified.com/es/blog/manual-vs-automatizadas-software-pruebas/

Colaborador Red Hat. (31 de julio de 2023). *¿Qué es y para qué sirve un IDE?* Obtenido de Red Hat: https://www.redhat.com/es/topics/middleware/what-is-ide

Daly, L. (s.f.). *Atlassian*. Obtenido de Kanplan: donde el backlog y kanban unen sus fuerzas: https://www.atlassian.com/es/agile/kanban/kanplan

Dayal, R. (23 de noviembre de 2021). *Guide To Selenium WebDriver: Getting Started With Test Automation [Tutorial]*. Obtenido de Lambda Test: https://www.lambdatest.com/blog/selenium-webdriver-tutorial-with-examples/

DGTIC. (27 de febrero de 2023). *Portal TIC UNAM*. Obtenido de La importancia de separar los ambientes de trabajo en el desarrollo de los productos de software: https://www.tic.unam.mx/2020/11/16/la-importancia-de-separar-los-ambientes-de-trabajo-en-el-desarrollo-de-los-productos-de-software/

Dustin, E., Rashka, J., & Paul, J. (1999). *Automated Software Testing: Introduction, Management, and Performance.* Boston: Addison Wesley Professional.

Edix. (13 de septiembre de 2021). *Framework: qué es, para qué sirve y algunos ejemplos*. Obtenido de https://www.edix.com/es/instituto/framework/

Esteller, V., & Medina, E. (2012). Procesos de desarrollo de software y materiales educativos computarizados. *Revista Eduweb*, 85-99.

Flores, F. (13 de abril de 2023). *Qué es Visual Studio Code y qué ventajas ofrece*. Obtenido de OpenWebinars.net: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/

Gómez, Ó. T., López, P. P., & Bacalla, J. S. (2014). Criterios de selección de metodologías de desarrollo de software. *Industrial Data*, 70-74. Obtenido de Industrial Data.

Kendall, K. E., & Kendall, J. E. (2011). *Análisis y diseño de sistemas.* México: Pearson Educación.

M., E. S., M., R. M., & O., P. T. (2019). A Review to Reality of Software Test Automation. *Computación Y Sistemas*, 169-185.

Muradas, Y. (14 de abril de 2023). *Qué es Gradle: La herramienta para ser más productivo desarrollando*. Obtenido de OpenWebinars.net: https://openwebinars.net/blog/que-es-gradle/

Ojeda., j. C., & Fuentes, M. D. (2012). *Taxonomía de los modelos y metodologías de desarrollo de software más utilizados*. Obtenido de Biblat: https://biblat.unam.mx/es/revista/universidades-mexico-d-f/articulo/taxonomia-de-los-modelos-y-metodologias-de-desarrollo-de-software-mas-utilizados

Olvera, M. A. (2014). *Fundamentos de Computación para Ingenieros.* México: Grupo Editorial Patria.

Pantaleo, G., & Rinaudo, L. (2015). *Ingeniería de software.* México: Alfaomega.

Pressman, R. S. (1998). *Ingeniería del Software - Un Enfoque Práctico* (Séptima ed.). México: McGraw-Hill.

Real Academia Española. (02 de marzo de 2024). *Diccionario de la lengua española*. Obtenido de https://dle.rae.es/calidad

Rehkopf, M. (s.f.). *Pruebas de software automatizadas*. Obtenido de ¿Qué son las pruebas automatizadas?: https://www.atlassian.com/es/continuous-delivery/software-testing/automated-testing

Ríos, J. R., Ordóñez, M. P., Segarra, M. J., & Zerda, F. G. (2018). Comparación de metodologías en aplicaciones web. *3C Tecnología*, 1-19.

Sánchez, G. (2017). Selenium WebDriver en un Ambiente de Pruebas Continuas. *Publicación mensual Software Guru*, 14-17.

Santander Universidades. (11 de marzo de 2024). *Metodologías de desarrollo de software: ¿qué son?* Obtenido de Santander Open Academy: https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html

Sommerville, I. (2005). *Ingeniería del Software* (Novena ed.). México: Pearson México.

Soto, V. M. (23 de noviembre de 2020). *Pragma*. Obtenido de Tipos de pruebas y dónde aplicarlas: https://www.pragma.com.co/academia/lecciones/tipos-de-pruebas-y-donde-aplicarlas

TestProject. (31 de enero de 2021). *Axelerant*. Obtenido de TestProject: https://testproject.io/axelerant-customer-success-story/

Unadkat, J. (30 de junio de 2024). *Selenium Webdriver Tutorial in Java with Examples*. Obtenido de BrowserStack: https://www.browserstack.com/guide/selenium-webdriver-tutorial

Yagüe, C. (16 de junio de 2023). *Qué es Apache Maven*. Obtenido de OpenWebinars.net: https://openwebinars.net/blog/que-es-apache-maven/

Zamudio, E. (24 de febrero de 2024). *Construcción de Proyectos con Gradle.* Obtenido de SG Buzz: https://sg.com.mx/revista/33/construccion-proyectos-gradle

10. Anexos.

10.1. Índice de figuras.

10.2. Glosario.

Daemon: Es un programa de computadora que se ejecuta en segundo plano.

JavaDoc: es una herramienta del SDK que permite documentar, de una manera rápida y sencilla, las clases y métodos que se proveen, siendo de gran utilidad para la comprensión del desarrollo.