UNIVERSIDAD INTERNACIONAL DE LAS AMÉRICAS
ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO DE GRADUACIÓN

Para optar por el grado de Bachillerato en Ingeniería Informática

PROPUESTA PARA LA AUTOMATIZACIÓN DE PRUEBAS DE LA APLICACIÓN EDOC EN LA EMPRESA GURUSOFT, UBICADA EN LA SABANA

GÉNESIS IVONNE CUBILLO SALAZAR
AUTORA

MBD. OLMAN NÚÑEZ PERALTA
TUTOR

DANIEL ÁLVAREZ GARRO
LECTOR

San José, Costa Rica
DICIEMBRE, 2021

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN EJECUTIVO

El desarrollo web ha crecido en los últimos años, por lo que las empresas deben realizar constantes cambios a sus sistemas o sitios web; mientras el equipo de desarrollo realiza los cambios, el equipo de pruebas y calidad hace el testeo de la aplicación. Estas pruebas son parte del ciclo de desarrollo del *software*, pueden ser manuales, y consisten en verificar la funcionalidad de la aplicación, sean características nuevas, o bien, características antiguas que siguen siendo parte del sistema.

Usualmente, el testeo manual es una gran parte del proceso de pruebas, sin embargo, existen pruebas automatizadas que pueden reemplazar ciertas tareas manuales. Estas son desarrolladas por el equipo de calidad, o bien, por el equipo de desarrollo y reducen el tiempo que se consume ejecutando pruebas de tipo manual, por su habilidad de ejecutarse más rápidamente, en diferentes navegadores, entre otras características.

Por lo tanto, la automatización resulta una solución eficiente para ejecutar pruebas, donde el desarrollador, el profesional de calidad u otro miembro del equipo no tengan que gastar su tiempo ejecutando pruebas manuales. De esta manera, se reduce el tiempo en tipos de pruebas como la regresión y se confirma que las principales funcionalidades estén trabajando de forma correcta.

Las pruebas de automatización consisten en manejar el computador para realizar ciertas tareas escritas por los desarrolladores o *testers* en un lenguaje de programación. Para esto requiere de un *framework*, tal es el caso de Selenium, que se utiliza para la presente propuesta, y es una herramienta para automatizar diferentes navegadores con el fin de comprobar funcionalidades en sitios web e imitar el comportamiento que tienen los usuarios cuando interactúan con el sistema.

Dicho esto, las pruebas se desarrollan para una aplicación llamada eDoc, que pertenece a la empresa Gurusoft. Dicha aplicación no tiene pruebas automatizadas, por lo que, si se realizan pruebas, deben ser de forma manual. Las pruebas automatizadas serían un mejoramiento tanto para el proceso de desarrollo como el proceso de pruebas de la empresa.

CAPÍTULO I. INTRODUCCIÓN

La presente propuesta de proyecto va dirigida a la fase de pruebas en el ciclo de vida del *software* de una aplicación llamada eDoc, que pertenece a la empresa Gurusoft, enfocada en la facturación electrónica.

Las pruebas de *software* (*Software Testing*) son el conjunto de actividades dentro del ciclo de vida del *software*, que se realizan para identificar posibles fallos en el funcionamiento de un programa. Estas se pueden realizar de forma manual, creando casos de pruebas (conjunto de variables, pasos de entrada y resultados esperados que se crean para que el analista pueda determinar si la funcionalidad o requerimiento de un sistema es satisfactorio) y ejecutándolos sobre el sistema que está bajo construcción.

Una vez que un sistema ya se encuentra en producción, será sometido a varios cambios, requerimientos y mejoras. Lo que significa que el desarrollo y su ciclo de vida no finalizan, al contrario, por cada nueva versión, se debe comprobar la función del requerimiento nuevo, así como el resto del sistema. Lo que significa que, cada vez que se integran nuevos cambios al sistema, el proceso de testeo se hace más grande y repetitivo, ya que se incluyen las pruebas para el nuevo funcionamiento y las pruebas para comprobar que el resto del sistema continúe funcionando de manera correcta. A este último tipo se le llaman *pruebas de regresión*. Por lo general, las pruebas de regresión manuales suelen ser repetitivas y dan espacio a la necesidad de la creación de pruebas automatizadas.

La automatización de pruebas es una técnica de pruebas de *software* que se realizan utilizando herramientas de *software* especiales para ejecutar un conjunto de casos de prueba y es la mejor manera de aumentar la eficacia, la cobertura de las pruebas y la velocidad de ejecución en las pruebas de *software*.

En el presente proyecto, se pretende crear un plan de pruebas (documento que describe el alcance, el enfoque, los recursos y los casos de prueba para la automatización) y desarrollar un *framework* para la automatización de pruebas en la aplicación eDoc, previamente mencionada. De esta forma, se facilitará el proceso de pruebas y de desarrollo para el aplicativo y se ahorrará el trabajo repetitivo para el equipo de desarrollo.

Planteamiento del problema

En el proceso de desarrollo de *software* existe una etapa de pruebas, la cual es ejecutada principalmente por personal del aseguramiento de calidad y *testers*. Quienes realizan manualmente pruebas sobre el sistema al que se le aplican cambios, ya sean nuevos cambios o bien revisar módulos existentes. En Gurusoft, la mayoría de estas pruebas no son llevadas a cabo por *testers*, sino por los mismos desarrolladores, quienes toman su tiempo de desarrollo para realizar la fase de *testing* de forma manual. Esto consume tiempo y esfuerzo. Pero este tiempo podría ser reducido con las pruebas automáticas, lo que se pretende desarrollar en el presente proyecto.

Problemas

Atrasos en la ejecución de procesos de desarrollo.

En la aplicación eDoc se realizan cambios y modificaciones en el código de la aplicación y, posteriormente, se necesita verificar que sigue funcionando de manera apropiada. En la empresa, esto se hace de forma manual, y los tiempos de respuesta o el tiempo que se consume haciéndolo suelen atrasar el proceso de desarrollo (desarrollar nuevas características, arreglos de bugs existentes, diseños), ya que, antes de seguir desarrollando sobre las aplicaciones, lo ideal es verificar su estado después de cada adición, eliminación o modificación de código.

Errores en la funcionalidad en el sistema debido al testeo manual.

Debido a que las pruebas de regresión realizadas en el sistema eDoc se toman su tiempo y son hechas por personas, este proceso no se hace de forma completa. Y, como consecuencia, existen errores en el sistema, se reintroducen defectos antiguos que volvieron a aparecer por falta de un constante monitoreo y la posibilidad de perder un defecto en el sistema por error humano. El test de regresión no es confiable cuando solamente es manual.

Inexistencia de bitácoras o reportes luego de las pruebas.

No existen reportes que presenten resultados luego de una regresión manual. Lo cual evita que se lleve una bitácora o un *tracking* de los fallos. Esto suele llevar a ignorar los defectos que está presentando la aplicación y, finalmente, que no sean ser arreglados, lo cual afecta la salud de la aplicación.

Tareas repetitivas.

Por su propia naturaleza, las pruebas de regresión requieren mucho tiempo y son repetitivas. Al realizarlas de forma manual, causan desinterés y desmotivación en el equipo, lo que desencadena en no llevar a cabo las pruebas de forma correcta. Por esta razón, es sumamente importante automatizarlas.

Objetivo general

Eficientizar el ciclo de vida del desarrollo de *software* y el ciclo de vida del testeo por medio de una propuesta de automatización de pruebas para la empresa Gurusoft.

Objetivos específicos

1. Identificar la situación actual del proceso de *testing* en la empresa.
2. Analizar los módulos y funcionalidades del sistema eDoc, para la creación de casos de prueba automatizados.
3. Crear el documento de plan de pruebas, donde se registrarán los casos de pruebas, el alcance de la automatización de pruebas, tipos de pruebas, entre otros.
4. Elaborar, según los requerimientos y la información recolectada del plan de pruebas, el prototipo de automatización.
5. Programar la propuesta de automatización de las pruebas que se ejecutarán sobre el sistema eDoc.
6. Realizar las pruebas pertinentes garantizando que todos los requerimientos están incluidos.

Justificación del proyecto

Con el objetivo de buscar mejoras al proceso de desarrollo de *software* realizado para el aplicativo eDoc y teniendo en cuenta que actualmente en Gurusoft no se tiene un proceso de pruebas definido ni mucho menos casos de prueba automatizados, se busca desarrollar el prototipo de la automatización; para que, de esta manera, el aseguramiento, control de calidad y procesos de prueba se puedan mejorar, con el fin de reducir tiempos en el testeo de la aplicación eDoc, así como reducir los tiempos de espera de desarrollos siguientes.

Viabilidad técnica

Técnicamente, el proyecto es posible, ya que implica una mejora en el proceso de desarrollo y en el proceso de pruebas de la empresa. Además, en la actualidad, en la organización se utiliza Scrum y se está implementando poco a poco prácticas de principios Agile. En las prácticas y desarrollo Agile, las pruebas deben realizarse con frecuencia. Por lo tanto, en lugar de esperar a que finalice el desarrollo, antes de comenzar las pruebas, estas se realizan continuamente, a medida que se agregan funciones; si esto se hace solamente de forma manual, no es posible que el desarrollo ni las pruebas sean de acuerdo con los principios ágiles.

Consideraciones de *software*:

- C# .Net Core 4.7.2
- Visual Studio Community 2019
- Specflow v3.8.7
- Nunit v3.17.0
- Selenium Web Driver v3.14.0
- AutoIT 3.3.14.5
- GitHub 2.25.0

Consideraciones de *hardware*:

- Computadora de escritorio con Windows 10:
  - Procesador: AMD Ryzen 5 3600.
  - Memoria RAM: 16.0 GB.
  - Disco duro 2 Terabytes.

Viabilidad operativa

La propuesta de automatización para el sistema es operacionalmente viable, ya que se hará uso después de su desarrollo. Cada vez que se integren nuevos cambios y versiones en el aplicativo eDoc, se podrá hacer uso del *framework*.

Los usuarios que utilizarán la automatización son, principalmente, los miembros del equipo de desarrollo, quienes están familiarizados con el lenguaje de programación por utilizar para la programación del prototipo. Sin embargo, necesitarán una breve explicación o introducción para utilizar el proyecto y para realizar mantenimientos en este. No obstante, el entrenamiento queda fuera del alcance del presente proyecto.

En el entrenamiento, se incluyen los siguientes temas: explicación de tecnologías y del código, ejecutar la automatización, estructura y mantenimiento del *framework* en cuanto a la inserción de nuevo código o modificaciones en el actual, y lectura e interpretación de reportes.

El tiempo estimado de la capacitación es de un día o dos como máximo. Debido a que el equipo cuenta con conocimientos previos de programación. Sin embargo, es importante recalcar que la capacitación ni ningún tipo de entrenamiento están incluidos. La estudiante se compromete solamente a la presentación del prototipo.

Viabilidad económica

Aunque la mayoría de las tecnologías por utilizar son de uso libre y no requieren de compra de licencia, existe un gasto económico, que es el de sistema de control de versiones, el cual debe ser pagado, ya que el plan sin costo no provee todas las funcionalidades necesarias para el repositorio del código del proyecto. Los costos de GitHub de detallan a continuación:

- Plan Pro Team: $4 mensuales.
- Plan Enterprise: $21 mensuales.

El *hardware* utilizado y la computadora de escritorio para desarrollar serán propiciados por la estudiante, así como otros recursos, tales como el acceso a internet. El *software* por utilizar se lista a continuación y no tiene costo alguno, ya que es de uso libre:

- Visual Studio Community 2019
- Google Documents
- Git & GitHub Desktop 2.6.0

- Selenium Web Driver 3.14.0
- AutoIT 3.3.14.5
- Nunit 3.17.0
- Specflow 3.8.14

Viabilidad legal

No existe una razón legal que imposibilite el cumplimiento de los objetivos del proyecto. Sin embargo, se debe tomar en cuenta que, para el desarrollo de este, se utilizarán datos sensibles de la empresa, lo que conlleva a un acuerdo de confidencialidad escrito entre la estudiante y la empresa para la no divulgación de datos de la compañía y sobre los derechos de autor.

Con respecto al *software* utilizado, no se incumplirá ninguna ley, ya que toda herramienta tecnológica será de uso libre y para la herramienta de pago, la estudiante pagará la licencia por los meses donde esté desarrollando el prototipo. Una vez entregado el prototipo a la empresa, este plan se dejará de pagar. Además, el presente proyecto no deberá incumplir las siguientes leyes:

- Ley 9048, sobre delitos informáticos en temas de divulgación de información, ni en uso ilegal de *software*.
- Ley 8148, adición de los artículos 196 BIS, 217 BIS y 229 BIS al Código Penal; Ley 4573 para reprimir y sancionar los delitos informáticos de la Asamblea Legislativa de la República de Costa Rica (2001): la cual explica en el artículo 217, que será castigada por diez años aquella persona que influya en el resultado de datos de un sistema. Así mismo, el artículo 229 menciona que se castigará a una persona por cuatro años si esta borra, modifica o accede a información sin autorización.
- Ley de derechos de autor (Ley 6683) por parte de la Asamblea Legislativa de la República de Costa Rica (1982): según el acuerdo firmado entre la estudiante y la empresa, todo lo que se desarrolle será propiedad de la compañía, una vez que se entregue el prototipo.
- Ley 8968 sobre la protección de la persona frente al tratamiento de sus datos personales: su objetivo garantiza que cualquier persona merece respeto a sus datos personales, se encuentren estos en un proceso automatizado o en algún tratamiento manual.

Proyecciones

Con el prototipo se pretende reducir la forma manual en la que se hace el testeo de regresión en la aplicación eDoc, agilizando el proceso de ciclo de vida del desarrollo del *software*, así como el proceso de vida de *testing*. La organización Gurusoft se verá beneficiada reduciendo sus tiempos de pruebas y de desarrollo. Además, la automatización evitará que la regresión de pruebas sea un cuello de botella para el proceso de actuales y futuros desarrollos.

Alcance funcional

Como parte del proyecto, se creará y diseñará el *framework* de automatización. Se automatizan con la herramienta y tecnología seleccionada, todos los casos de prueba que fueron documentados en el *test plan*. Esta fase de desarrollo es la más amplia y la que requiere más tiempo, debido a que se lleva a cabo una programación del *framework* o *test suite* (conjunto de pruebas) de automatización.

Para empezar con el desarrollo, es importante recalcar que se utilizarán los siguientes modelos y prácticas en el código: Orientación Programada a Objetos, bases de Behavior Driven Development (ver apéndice C) y Modular Based Test Framework (ver apéndice D)

Alcance metodológico.

Para la creación del documento de pruebas, se utilizará como base Agile(ver apéndice E) , la cual menciona que un plan de pruebas debe contener los siguientes aspectos:

1. Alcance de pruebas: detalle sobre el alcance que tendrán las pruebas automáticas del prototipo en el aplicativo eDoc.
2. Funcionalidades que están bajo prueba: mencionar aquellos módulos del aplicativo eDoc que estarán siendo testeados.
3. Tipos de *testing*: describir y mencionar el tipo de *testing* para cada uno de los escenarios.
4. Casos de pruebas: desarrollar *test cases* y seleccionar los que serán automatizados.
5. Ambientes: ambiente donde serán ejecutados los casos de prueba automáticos.

Con respecto al desarrollo del prototipo, entre las metodologías por utilizar, se encuentran:

1. Modular Based Test Framework: modelo a seguir para la estructura del proyecto.

2. Behavior Driven Development: tipo de desarrollo que facilita la comunicación entre el negocio, el *product manager* y el equipo técnico como los desarrolladores.
3. OOP: práctica a seguir para la codificación del proyecto.
4. SDLC, STLC: ciclo de vida de *software* y *testing,* serán incluidos y se seguirán sus etapas.

Alcance tecnológico.

Para el desarrollo y creación del prototipo, las tecnologías por utilizar serán las siguientes:

Software:

- C# .Net 4.7.2
- Visual Studio Community 2019
- Specflow v3.8.7
- Nunit v3.17.0
- Selenium Web Driver v3.14.0
- AutoIT 3.3.14.5

Hardware:

- Computadora de escritorio con Windows 10:
  - Procesador: AMD Ryzen 5 3600.
  - Memoria RAM: 16.0 GB.
  - Disco duro 2 Terabytes.

CAPÍTULO II. MARCO REFERENCIAL

Ingeniería de software

La ingeniería de *software* o *Software Engineering* (SE) implica la construcción de programas informáticos. Es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de *software*, así como el estudio de estos enfoques (Wiebe y Chan, 2012).

Esta no es solamente el desarrollo o la programación de código, también requiere que los ingenieros declaren de forma precisa los requerimientos que el sistema debe satisfacer y crear los diseños que coincidan con esos requerimientos (O'Regan, 2019). Es importante mencionarlo, ya que, cuando se habla de ingeniería de *software*, se tiene el concepto erróneo de que solo se trata del desarrollo, pero esto no es así; ya que, antes de la programación y luego de cada programación de código, existen varias tareas que caben dentro del proceso o concepto de esta ingeniería.

El *software* está presente en casi todos los aspectos de la vida en la actualidad. Se puede encontrar desde electrodomésticos, juguetes, teléfonos celulares, automóviles, entre otros. Todos estos productos fueron creados por compañías donde se desarrolla o se desarrolló *software*.

En toda empresa donde se crea *software*, se debe seguir un proceso o plan específico de trabajo para desarrollar, mantener o mejorar los aplicativos que realiza la empresa. A este plan se le denomina ciclo de vida del desarrollo de programas (*Software Development Life Cycle*). Este es un flujo de trabajo o una secuencia de pasos que abarca el diseño, la documentación, la programación, las pruebas y el mantenimiento continuo de un *software* entregable (Atlassian, 2021a). Boyde (2014) enumera las siguientes fases: Requisitos y Especificaciones, Diseño, Desarrollo, Testeo y, por último, Despliegue y Mantenimiento.

- Requisitos y Especificaciones: se define el alcance, objetivos, requisitos y requerimientos del producto. Para esta primera fase, los analistas de negocio recopilarán los requisitos detallados completos del cliente. *Project Managers* y *Business Analyst* son los participantes de esta etapa, quienes recolectan toda la información de los clientes para conocer detalladamente el sistema que ellos necesitan.
- Diseño (*Design*): se plantea cómo se transformarán las especificaciones a la realidad. Esta fase puede implicar la creación de documentos de diseño, las pautas de codificación y la discusión de las herramientas, prácticas y tiempos de ejecución

- Desarrollo (*Implementation/Development*): llevando a cabo los diseños, por medio de la codificación. El equipo de desarrollo traslada el diseño del *software*, al código fuente y todos los componentes del sistema son llevados a cabo en esta fase.
- Testeo (*Verification/Testing*): se confirma que el producto desarrollado cumple con las especificaciones. Además, el usuario final confirma la usabilidad del producto y su conformidad con las expectativas. Las pruebas se empiezan una vez que se completa la codificación y cuando los módulos del sistema se lanzan para las pruebas. En esta fase, el *software* desarrollado se pone a prueba y los defectos encontrados se asignan a los desarrolladores para que los solucionen. Durante la fase de testeo, existe otro ciclo de vida, llamado ciclo de vida de pruebas o *Software Testing Life Cycle* (STLC), del que se explicará en la siguiente sección.
- Despliegue y Mantenimiento (*Release/Maintenance*): se conecta el producto final al ambiente donde los usuarios finales lo utilizarán. El equipo de desarrollo se encargará del mantenimiento del producto, si surgen problemas o si se deben realizar mejoras, la cual es una situación que siempre sucede en las empresas de *software* y así mismo en la empresa donde se desarrollará el presente proyecto.

Similar al desarrollo del *software*, el *software testing* tiene su propio ciclo de vida denominado ciclo de vida de pruebas. El STLC consiste usualmente en seis fases de testeo. Las etapas tienen el siguiente orden, tal como los enumera Mahfuz (2016): Análisis de Requerimientos, Planificación del *Testing*, Desarrollo de Casos de Prueba, Ejecución del Test y Casos de Prueba, y Resultados.

- *Requirement Analysis* (Análisis de requerimientos). Esta fase es parte de todo el *Software Life Cycle*. Sin embargo, abarca gran parte en el *Testing Life Cycle* (Mahfuz, 2016). Durante esta, se analizan y estudian los requerimientos para entender el alcance que tendrá el *testing*, lo que significa que, si existe algo que no se pueda probar, se debe documentar. Esta fase va de la mano con la primera etapa del SDLC, ya que, mientras los requerimientos del sistema están siendo recolectados, al mismo tiempo se puede ir realizando o recopilando los datos para el futuro testeo que será llevado a cabo durante la etapa o al finalizar la etapa del desarrollo.

- *Test Planning* (Planificación del *testing*). En esta etapa, se prepara la documentación de la estrategia del *testing* y el plan de pruebas. En los documentos se detallan los recursos, ambientes, tipos de pruebas, tecnologías o herramientas, entre otros conceptos esenciales para el *testing*.
- *Test Case Development* (Desarrollo de casos de prueba). Se definen las pruebas de caso o *test cases*, que serán incluidas en el *Test Plan*. El equipo de calidad de *software* desarrolla los escenarios y casos de prueba.
- *Test Execution* (Ejecución del testeo). Ejecución de los *test cases* que se definieron en la etapa anterior. El proceso consiste en ejecutar los casos de prueba y llevar a cabo el *testing* en general.
- *Result Analysis* (Análisis de resultados). Se muestran los resultados y reportes de los defectos encontrados en el sistema para que el equipo de desarrollo realice las correcciones correspondientes.

Las pruebas de *software* y el desarrollo del *software,* según Evertt y McLeod (2006), no son actividades ajenas. Ambos son procesos interdependientes, así como su éxito. El STLC es parte integral del SDLC y es básicamente las fases de pruebas dentro del *Software Life Cycle.* Como se mencionó anteriormente, el STLC tiene sus propias fases y algunas características en común con el ciclo de vida del *software*, pero continúan siendo diferentes entre sí, aunque sean interdependientes. Es importante recalcar los siguientes puntos para su relación:

- El STLC es parte del SDLC. No es posible que el *Software Testing Life Cycle* se ejecute de forma independiente (Mahfuz, 2016, p. 63). Siempre se va a necesitar del proceso de un *software* y del *software* en general para llevar a cabo el ciclo de vida del testeo.
- El *Software Testing Life Cycle* se limita a las pruebas. Mientras que el *Software Development Life Cycle* tiene un mayor alcance con más entradas y ejecuciones. (Mahfuz, 2016, p. 63).
- El ciclo de vida del *testing* es una parte muy importante del SDLC. “Una versión de *software* no debería suceder sin ejecutarlo a través del proceso de pruebas (Mahfuz, 2016, p. 63). Esto significa que, al crear un *software,* debe ser testeado correctamente, siguiendo las fases del ciclo de vida. De lo contrario, se estaría entorpeciendo los dos ciclos de vida, tanto del *software* como el de *testing*.

Esta relación es importante tenerla en cuenta, ya que el tipo de prototipo que se desarrollará suele llevarse a cabo por el equipo de calidad del sistema (*Software Quality Assurance*), quienes son los que usualmente hacen uso del *Software Testing Life Cycle*, sin embargo, existirá una parte programada, por lo que también se hará uso del *Software Development Life Cycle* y del *Automation Testing Life Cycle* (que se explicará más adelante). A pesar de que el proyecto sea enfocado al área y el proceso de calidad de sistemas, no quiere decir que no exista código y no se haga uso del SDLC, porque se debe recordar que, dentro del SDLC, también forma parte el equipo de calidad y de pruebas.

Dentro de los departamentos de Tecnologías de la Información, suele existir en la mayoría de las empresas de informática, el personal que se dedica al aseguramiento de la calidad del *software* (*Quality Assurance* [QA]). Según Chopra (2016), el aseguramiento de la calidad significa garantizar que un sistema cumpla con todos sus objetivos. Es una rama de la informática que tiene como principal objetivo asegurar la calidad, por lo que el *Software Quality Assurance* debe estar presente desde la definición de los requerimientos hasta su lanzamiento como producto final.

Quality Assurance (QA)

Un analista de calidad o ingeniero de la calidad del *software* (*Quality Assurance Analyst* o *Quality Assurance Engineer*) es el responsable de llevar a cabo todas las prácticas de calidad a través del *Testing Life Cycle*. Como Chopra (2016) lo menciona, las principales funciones de un analista de calidad son: crear documentos de planes de pruebas (*Test Plan*), ejecutar el *Test Plan* y gestionar todas las actividades en el plan.

El ingeniero de QA no solamente es el encargado de realizar las pruebas al sistema en desarrollo y encontrar defectos, sino que también, como parte de sus responsabilidades, debe ayudar al equipo a prevenir los defectos en el sistema, y a buscar mejoras, trabajando de forma muy cercana con los desarrolladores y otros miembros del equipo como *project managers* o *product owners*, así como en el desarrollo y creación de pruebas automatizadas. De esta forma, el ingeniero de QA se convierte en un miembro muy valioso para el equipo. No se debe confundir el término ingeniero de QA con *tester*, si bien el *tester* es un miembro del equipo del proyecto que realiza actividades de prueba, como *testeo* de sistemas o de usuarios y en sus responsabilidades está probar la aplicación, registrar y rastrear todos los defectos; la diferencia entre *tester* y el ingeniero de QA radica en que este último es responsable de la calidad, de gestionar todas las actividades y

participar en todas las actividades del ciclo de vida de *software* del equipo que desarrolla el proyecto; mientras que el *tester* realiza su actividad al finalizar el desarrollo. Sin embargo, ambos trabajan en el *software testing*.

La principal función del testeo del *software* es identificar todos los defectos existentes en un sistema. El sistema se ejecuta bajo ciertas pruebas con la intención de encontrar todos los errores del *software*. Sin embargo, el *software testing* es más que solo encontrar errores. Es básicamente el proceso para asegurar que el código del sistema cumple con las funciones esperadas y realiza lo que se supone que debe hacer basándose en los requerimientos del sistema previamente definidos.

Existen muchos tipos de *testing*. Cada uno enfocado para diferentes ocasiones y propósitos. Principalmente, se pueden dividir en dos métodos denominados: *Black Box Testing* y *White Box Testing*. El *White Box Testing* no se definirá de manera profunda, debido a que este tipo o método de testeo se basa en el código interno del aplicativo. Implica conocer el código fuente para validar su integridad y su lógica y es llevado a cabo mayormente por los desarrolladores de *software*, por ejemplo, el *unit testing* (ver apéndice F).

Por otro lado, se encuentra el *Black Box Testing*, el cual verifica la funcionalidad de la aplicación. El *testing* en este método se suele basar en requerimientos y funcionalidades del sistema. Este proceso suele ser llevado a cabo por el equipo de QA, por lo tanto, no requiere conocimiento específico del código interno del *software*, y se suele dividir en Pruebas Funcionales (*Functional Testing*) y No Funcionales (*Non Functional Testing*). Para el presente proyecto, se hará mayor uso del *Functional Testing*, y según Mahfuz (2016), este tipo de *testing* se enfoca en los requerimientos del sistema o la aplicación. Basa sus casos de prueba en las especificaciones del componente de *software* bajo testeo, y está orientado a validar los requisitos funcionales de una aplicación. Entre algunos tipos de *Functional Testing* se encuentran:

- *Sanity testing:* determina si la versión de un *software* se está comportando bien como para realizar un esfuerzo mayor de testeo. Mahfuz (2016) describe el siguiente ejemplo: "Si el nuevo *software* falla cada 5 minutos, o destruye las bases de datos, es posible que no esté en una condición lo suficientemente «sana» como para justificar más pruebas hasta que se haya «ajustado» un poco más" (p. 67). Este proceso de prueba lo realizan los equipos de *Quality Assurance*. El *sanity testing* se enfoca en un testeo rápido, y no muy detallado, después de cambios en el código; si durante este tipo de testeo las

pruebas son fallidas, se rechazan los cambios y los desarrolladores deberían arreglarlo. También este testeo puede ser ejecutado cuando se realiza una corrección crítica y urgente en el sistema, por lo que requiere una prueba inmediata.

- *Regression testing*: las pruebas de regresión buscan descubrir errores de *software* después de introducir cambios en el programa, y es uno de los tipos de pruebas más comunes. También es uno de los métodos de prueba más costosos, porque se vuelve a probar después de que se han realizado arreglos o modificaciones en el *software* o su entorno. Debido a las implicaciones de costos y al hecho de que ocurre después de que se ha identificado un problema, puede ser difícil determinar cuántas repeticiones de pruebas se requerirán. Este proceso de prueba lo realizan los equipos de prueba (equipo de *Quality Assurance*) y las herramientas de prueba automatizadas pueden resultar útiles para este tipo (Mahfuz, 2016). La necesidad del *regression testing* se da por varias situaciones: comúnmente se da cuando se necesita cambiar algo en el código y se debe probar si este afecta otra parte de la aplicación o no; cuando se agrega una nueva función a la aplicación, para la corrección de defectos y para la resolución de problemas de rendimiento.

  Para la realización de testeo de regresión, es necesario crear *test cases*, los cuales deben cumplir con ciertas características, tales como casos de prueba que contengan defectos frecuentes, que verifiquen funcionalidades principales en la aplicación, funcionalidades que suelen ser más usadas por los usuarios y nuevos *test cases* que verifican funcionalidades recién integradas al sistema.
- *Smoke testing*: se ejecutan de manera rápida y brinda la seguridad de que las mayores características funcionan correctamente (Atlassian, 2021b). Este tipo es para chequear las funcionalidades básicas del sistema de *testing.* Se puede decir que un *smoke testing* es una versión pequeña y rápida del *regression testing* anteriormente explicado y asegura que las funciones críticas del sistema están funcionando.
- *User Acceptance Testing*: este es el último tipo de *testing* que se realiza contra las especificaciones del producto final. Es una prueba formal que se lleva a cabo para determinar si un sistema satisface o no sus criterios de aceptación y el cliente determinar si acepta o no el sistema. Por lo general, se realiza por medio del uso práctico del cliente

> en la aplicación por un tiempo determinado. Normalmente, es llevado a cabo por los clientes (Mahfuz, 2016). Se ejecutan después de que se realizaron todas las pruebas necesarias en el sistema o en el módulo de este, y antes de llevarlo a cabo, se debe tener en consideración o incluso asegurar que ningún defecto crítico esté presente en la aplicación, y que el sistema siga el flujo de requerimientos y especificaciones que los usuarios y clientes solicitaron al inicio del SDLC.

El *Non Functional Testing* sigue siendo importante para el testeo de aplicaciones, pero para el presente proyecto, este tipo de testing no será el de mayor protagonista de la propuesta, principalmente se enfocara en pruebas funcionales. Sin embargo, se menciona su significado y algunos tipos.

Las pruebas de tipo no funcional se encargan de probar las partes no funcionales del sistema. Es decir, testea el desempeño y rendimiento de la aplicación. Se suele hacer con herramientas y muy pocas veces de forma manual; algunos de sus tipos son*: Performance Testing*, Security Testing, *Load Testing* y *Stress Testing*.

Además de las pruebas anteriormente mencionadas, existen muchos más tipos, por ejemplo: el *end to end testing*, *integration testing*, pruebas automatizadas, entre otros. Al desarrollo de pruebas se les denomina pruebas automatizadas, que son técnicas de prueba que utilizan herramientas de automatización para controlar la configuración del entorno, la ejecución de pruebas y el informe de los resultados. Esto requiere un proceso de prueba manual formalizado que existe actualmente. Por lo que el analista de QA primero debe establecer un proceso de prueba efectivo. El propósito de las herramientas de prueba automatizadas es automatizar, principalmente, las pruebas de regresión (Atlassian, 2021b). Esto significa que el QA debe desarrollar unos casos de prueba detallados que sean repetibles y estas pruebas deben ejecutarse cada vez que haya un cambio en la aplicación, para garantizar que el cambio no produzca consecuencias no deseadas o negativas en el sistema, como se explicó anteriormente en los tipos de test funcionales. De esta manera, se ahorra tiempo y recursos en el testeo manual.

La principal diferencia entre el testeo manual y automático es bastante simple. El testeo manual es llevado a cabo por el ser humano, quien interactúa directamente con la aplicación, haciendo clic a través de diferentes módulos en el sistema, interactuando con Apis y con herramientas. Atlassian (2021b) menciona que el testeo manual: “Es muy costoso, ya que requiere

que alguien configure un entorno y ejecute las pruebas por sí mismo, y puede ser propenso a errores humanos, ya que el evaluador podría cometer errores tipográficos u omitir pasos en el script de prueba" (párr. 2).

Por otro lado, las pruebas automatizadas son realizadas por un robot que ejecuta código o un *script* previamente creado. Así que, según SmartBear (2021a): "se puede dedicar más tiempo a otras tareas, como las pruebas exploratorias, mientras se automatizan las pruebas que consumen mucho tiempo, como las pruebas de regresión" (párr. 5). Además de que estas pruebas automáticas ahorran tiempo y facilitan la realización de otras actividades, por ende, agilizan el ciclo de vida del *testing*, así como todo el proceso y ciclo de vida del desarrollo. Porque, al terminar un desarrollo, el equipo de QA debe probarlo y esto, por supuesto, se toma su tiempo. Hasta que no se finalice el proceso de *testing*, no se termina ni el *Software Development Life Cycle* ni el *Software Testing Life Cycle.*

Entre otros beneficios del *automated testing,* se encuentran: hay más velocidad en el desarrollo y en la entrega del producto, los lanzamientos de características nuevas, actualizaciones en la aplicación o *releases* se hacen más frecuentemente. En síntesis, se reduce el tiempo y se aumenta el desarrollo.

Para la creación de un *Test Suite* (conjunto de casos de prueba) de pruebas automatizadas, existen varias tecnologías, herramientas y *frameworks*. Algunas de ellas dependientes de un lenguaje de programación y otras que son totalmente independientes de cualquier lenguaje. Se debe recordar que los sistemas, si son sitios webs, pueden ser accedidos desde cualquier dispositivo, desde una computadora de escritorio hasta un celular, por lo que estas herramientas muchas veces están enfocadas para diferentes ambientes.

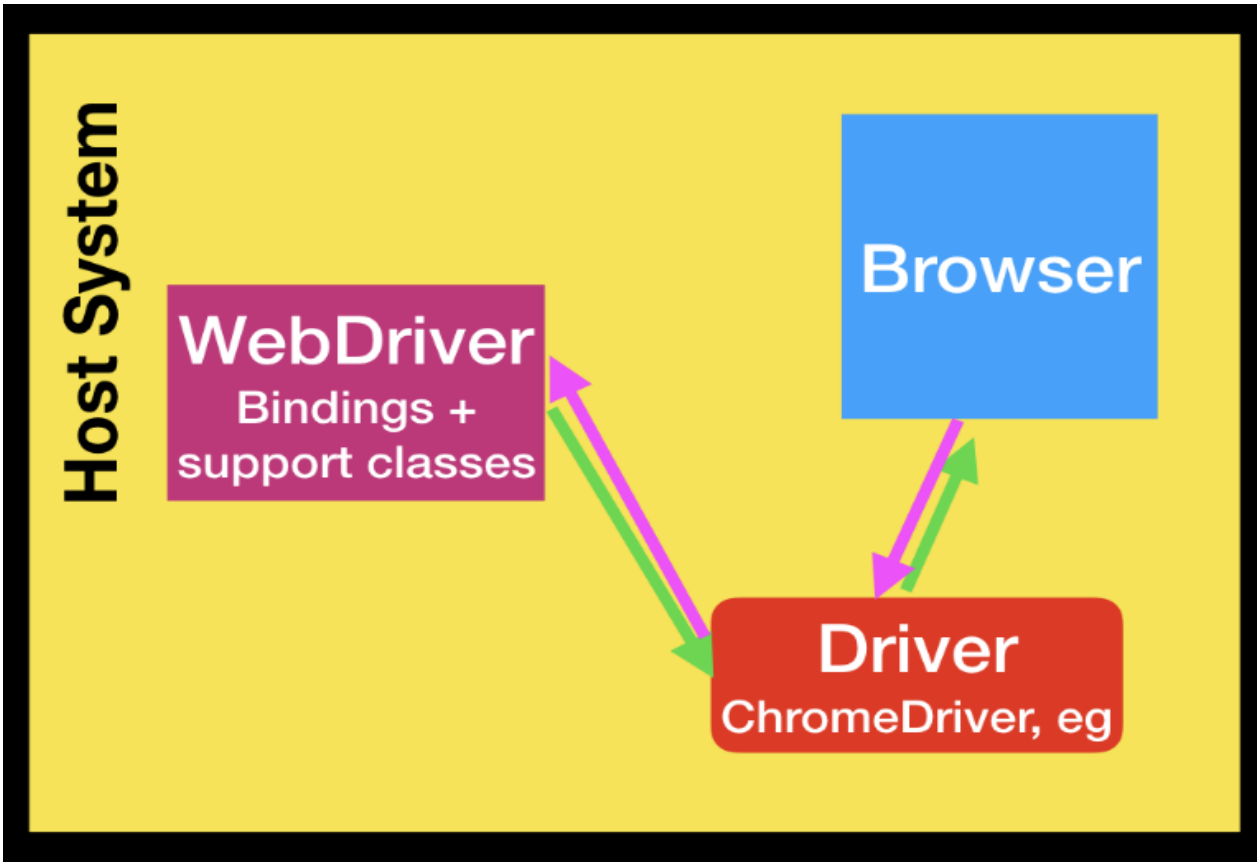
Entre las herramientas para crear pruebas de automatización se encuentra Selenium Web Driver, que es una librería para controlar el navegador y hacer una simulación de un usuario utilizando o interactuando con el explorador de internet y la página web. La primera versión de Selenium fue lanzada por primera vez en 2004 y desde entonces se ha vuelto muy popular, grandes empresas lo están utilizando actualmente, además de que tiene patrocinio por empresas como Micro Focus, SauceLabs, SmartBear, entre otras. Esta herramienta es capaz de realizar actividades como si una persona estuviera utilizando el sitio, por lo que puede hacer clics, introducir texto en cajas

de texto, seleccionar listas desplegables, hacer clic en hipervínculos, movimientos del *mouse,* entre otras acciones.

Selenium soporta automatizar la mayoría de los navegadores como Chrome, Mozilla y Firefox. El elemento que se utiliza para controlar el explorador se le llama *Driver*, es el encargado de levantar o abrir el navegador. Este *driver* es específico para cada *browser,* por ejemplo, al *driver* que se utiliza para correr pruebas en Chrome se llama ChromeDriver

En la figura 1, se explica cómo funciona Selenium. Primeramente, el WebDriver contiene las clases e instrucciones para la interacción con el navegador y este WebDriver se comunica con el Driver del navegador (en este caso ChromeDriver), para enviarle las instrucciones al explorador de internet.

Figura 1
WebDriver

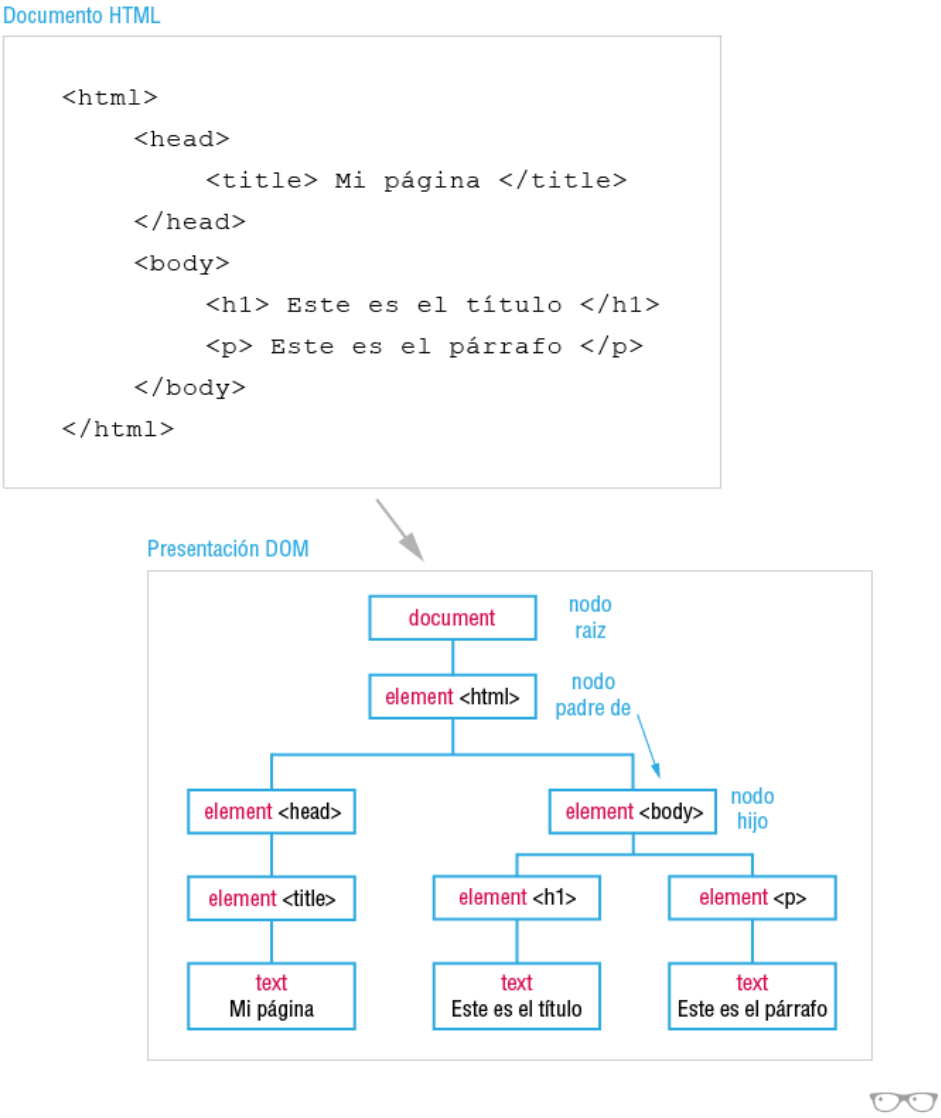


Fuente: Selenium (2021a).

Selenium (2021a) indica que: “Una técnica fundamental para utilizar Selenium, es conocer cómo se deben encontrar los elementos en el sistema” (párr. 1). En una página o sistema web, siempre se tienen un conjunto de elementos como botones, cajas de texto, entre otros. Este conjunto de elementos debe tener una estructura, un estándar y una forma de mostrar los elementos. A ese estándar se le denomina HTML.

Selenium debe utilizar el *Document Object Model* (DOM) de la página, es la forma en que HTML mantiene todos los elementos de forma estructurada (Ver figura 2). Por lo tanto, Selenium utiliza el DOM para acceder a los elementos de la página por medio del nombre del elemento o identificador único.

Figura 2
Diagrama HTML y DOM



Fuente: GeeksforGeeks (2021).

Selenium, además del WebDriver, tiene otras herramientas para la automatización de pruebas, sin embargo, para la creación del presente prototipo, se estará utilizando solo la librería del WebDriver. Es importante agregar y explicar que esta depende de un lenguaje de programación para ser utilizada, lo que quiere decir que necesita la creación de métodos y clases para comunicarse y darle instrucciones al navegador. La librería es compatible con varios lenguajes muy utilizados,

por ejemplo: C# (el cual se estará utilizando para la creación del prototipo), Java, Python, JavaScript y Rugby.

En la figura 3, se muestra un ejemplo de un código en Java, utilizando la librería Selenium. Donde, básicamente, lo que este código hace es: inicializar el *driver* para comunicarse con el navegador Firefox e ir a la página principal de Google; seguidamente, inserta un texto en la caja búsqueda de Google, presiona la tecla *enter* y, por último, selecciona el primer resultado y copia el nombre del texto del título para, finalmente, imprimir el contenido del título en consola.

Figura 3
Código de ejemplo en Java con Selenium

```java
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.WebDriverWait;
import static org.openqa.selenium.support.ui.ExpectedConditions.presenceOfElementLocated;
import java.time.Duration;

public class HelloSelenium {

    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        try {
            driver.get("https://google.com/ncr");
            driver.findElement(By.name("q")).sendKeys("cheese" + Keys.ENTER);
            WebElement firstResult = wait.until(presenceOfElementLocated(By.cssSelector("h3")));
            System.out.println(firstResult.getAttribute("textContent"));
        } finally {
            driver.quit();
        }
    }
}
```

Fuente: Selenium (2021b).

Existen otros tipos de *frameworks* que ayudan en la creación de las pruebas automatizadas, estos son guías y buenas prácticas por seguir para un mejor funcionamiento, mantenimiento y orden en la automatización. Algunas de ellas son: Linear Automation Framework, Data Driven Framework, Modular Based Framework, entre otros. El *framework* basado en módulos (*Modular Based Framework)* requerirá que el código de las pruebas sea dividido en varios módulos lógicos y aislados. Para cada módulo, se crea una secuencia de comandos de prueba separada e independiente. Este tipo de *framework* hace posible la realización del código con el paradigma de

Programación Orientada a Objetos, que se basa en el concepto de clases y objetos. Una clase es una plantilla para objetos y un objeto es una instancia de una clase. Cuando se crean los objetos individuales, heredan todas las variables y funciones de la clase (W3schools, 2021). Este paradigma se utiliza para estructurar el código en piezas simples y reutilizables.

Uno de los beneficios del *Modular Based Framework* es que el mantenimiento suele ser más sencillo, debido a que, si se tienen varios módulos, en caso de cambios (que siempre existen en el desarrollo), no se tendrá que cambiar todo el código, sino solo la parte que se necesite, sin afectar de manera crítica el resto del código.

Automation Testing Life Cycle (Ciclo de vida de la automatización de pruebas)

Así como la programación y el testeo tienen sus fases y ciclo de vida, también lo tiene el *testing automation*. Este ciclo de vida será utilizado para la realización de la propuesta de automatización. Consta de las siguientes fases: decidir el alcance de la automatización, selección de la herramienta correcta, diseño, configuración del ambiente, ejecución de pruebas y, por último, mantenimiento (QA Craft, 2021).

- Alcance: similar al STLC, se deciden los módulos de la aplicación que se van a testear.
- Herramienta: seleccionar la herramienta correcta es muy importante, ya que implica entrenamiento y costos para la empresa.
- Diseño y planificación: se caracteriza en cómo abordar y lograr el objetivo de la automatización de pruebas. Durante esta etapa, se realiza un procedimiento y plan de automatización.
- Configuración de ambientes: configuración de la máquina donde serán ejecutados los *scripts.*
- Ejecución de pruebas: es fundamental garantizar que todos los casos de prueba se ejecuten de manera correcta.
- Mantenimiento: esta etapa es cuando se agregan nuevos *scripts* y requieren inspección y mantenimiento, para mejorar el poder de los *scripts* de automatización.

Todos estos pasos o fases serán incluidos en el documento del *Test Plan*, el cual se explica a continuación.

Plan de pruebas

Para llevar a cabo un testeo, sea manual, automático o bien una combinación de ambos, se requieren varios componentes que resultan esenciales para lograr un testeo exitoso. Estos son desde documentos hasta artefactos creados antes o durante la fase del *testing*. Estos elementos ayudan en la planeación y estimación del testeo. Algunos de ellos incluyen:

Plan de pruebas (*Test Plan*): es un documento que describe el alcance y las actividades que se llevarán a cabo en el *testing*. Este documento es lo primero que se necesita crear en el *Software Testing Life Cycle* y se debe realizar de forma concisa y específica. Las secciones que contemplan el *Test Plan* deben estar muy bien detalladas. A continuación, se mencionan las diferentes partes que debe tener este documento:

- Alcance: definir el alcance que tendrán las pruebas, lo que se pretende testear y lo que no y que, por ende, queda fuera del alcance de pruebas.
- Herramientas y tecnología: se escogen las herramientas que se van a utilizar para la automatización o cualquier otro tipo de testeo. Se debe tener en cuenta si las tecnologías son de código abierto y si el equipo tiene conocimiento de este, de lo contrario, se debe realizar y describir la necesidad de un entrenamiento, así como explicar la razón por la que se escoge la herramienta.
- Tipo del testeo: como se mencionó anteriormente, existen muchos tipos de *testing*, se debe mencionar aquellos que se van a llevar a cabo y explicar por qué se decidió utilizar este *testing.*
- Casos de prueba (*test cases*): un *test case* son acciones que se ejecutan para verificar una función o característica del sistema. Estos pueden ser manuales o automáticos, pero la mayoría de las veces serán manuales primeramente, para luego ser automatizados. Los casos de prueba deben ser documentados con las siguientes secciones:
    - Identificador (ID) del *test case*: debe ser un ID único que identifique el caso de prueba.
    - *Test scripts*: el paso a paso de cómo ejecutar el *test case.* Estos pasos deben ser sumamente detallados con el fin de que sea entendible para cualquier otra persona que no sea quien los realizó.
    - Prerrequisitos: cualquier requisito que se necesite antes de ejecutar el *test case*.

    - Resultado esperado.
- Ambiente: las aplicaciones suelen tener varios ambientes donde se ejecutan. Esto depende de cada empresa, sin embargo, suelen existir tres ambientes en la mayoría de las compañías, los cuales son: ambiente de desarrollo (donde los programadores realizan y envían su código nuevo), ambiente de pruebas (es el ambiente donde los desarrolladores envían su código y características nuevas, y estos son testeados por el equipo de QA) y, por último, el ambiente de producción (donde los usuarios finales utilizan la aplicación). Pueden existir más de los anteriormente mencionados, pero en esta sección del *test plan* se debe describir donde serán ejecutadas las pruebas automáticas.
- Riesgos: todos aquellos riesgos que se identifiquen durante el desarrollo del testeo y las pruebas automáticas se deben documentar en esta sección.

Una vez este documento esté realizado, se procede a ejecutar el *testing* manual o bien a desarrollar las pruebas automatizadas. La empresa, al ser una organización de desarrollo de *software*, cumple con el ciclo de vida de desarrollo del *software* y, en cierta parte, con el ciclo de vida de *testing*, que anteriormente se explicaron; sin embargo, aunque se cumplan estos dos ciclos de vida, los dos se ven afectados por la forma en que se realizan las actividades del testeo. La forma manual en que se realiza actualmente y la omisión de documentos como el plan de pruebas y sus componentes entorpecen y retrasan actividades dentro de la empresa.

CAPÍTULO III. MARCO METODOLÓGICO

Enfoques de investigación

La investigación es un conjunto de procesos que se aplican para estudiar algún fenómeno o situación, con el fin de intentar entenderlo o medirlo. Es una actividad muy diversa que puede llevarse a cabo de diferentes maneras, perspectivas y se aplica a cualquier campo profesional. Por lo tanto, existen varios enfoques de investigación para adaptase de la mejor manera al tipo de investigación en el que se desea trabajar.

Enfoque cualitativo

La investigación de enfoque cualitativo implica la recolección de datos con el fin de responder al planteamiento del problema y entender conceptos. Está enfocado en profundizar el caso de estudio, por lo que se busca un entendimiento de la situación bajo estudio. No se enfoca en medir, sino que busca describir la percepción dentro de una situación por estudiar.

De acuerdo con Gómez (2006), el enfoque cualitativo se basa en la recolección de datos sin medición numérica y sin conteo. Principalmente, se utilizan las descripciones y observaciones. Los estudios de enfoque cualitativo involucran la recolección de datos utilizando técnicas que no se pretenden cuantificar, como la observación, entrevistas, revisión de documentos, experiencias personales e interacción con grupos. Se llevan a cabo en ambientes cotidianos, donde los participantes se comportan como lo harían en su rutina.

Enfoque de investigación seleccionado

Para la presente investigación, se utilizará el enfoque cualitativo, ya que su objetivo es observar y analizar el proceso de testeo en el desarrollo de *software* de la empresa, que actualmente se realiza de forma manual y que, por último, se plantee una propuesta de pruebas automáticas; de esta manera, se podrá optimizar el ciclo de vida del *software* y el ciclo de vida de *testing*. Las razones por las cuales se escoge el enfoque de investigación cualitativo se listan a continuación:

- Este enfoque se dirige a la resolución de un problema específico.
- Tal como lo menciona el investigador o la investigadora, empieza examinando la realidad, recolecta datos y desarrolla su teoría con lo que observa que ocurre.

- Se depende mucho de la opinión, perspectivas de los colaboradores de la empresa, la observación y entrevistas abiertas para la recolección de datos.
- La recolección de datos o el tipo de datos pueden ser orales, verbales, visuales o análisis y revisión de documentos.

Tipos de investigación

Los tipos de investigación son diferentes formas o modalidades que se aplican para desarrollar una investigación. Los tipos dependen del problema por resolver, el tipo de pregunta en la investigación, método o incluso la fuente de sus datos.

Investigación descriptiva

La investigación descriptiva, según Bernal (2010), es guiada por las preguntas de investigación que formula el investigador. Además de que utiliza, principalmente, las técnicas como la encuesta, la observación, entrevistas y revisiones de documentos.

Tipo de investigación seleccionado

En la presente investigación, se busca describir la situación actual de la empresa, con respecto al proceso de la realización de pruebas manuales por medio del documento de plan de pruebas, así como brindar y describir las mejoras para este proceso, las cuales serán reflejadas por medio de la propuesta de automatización.

Fuentes de información

En las fuentes de información se identifican las fuentes donde se obtendrá información para desarrollar el proyecto. Estas pueden presentarse de forma digital o física y, a continuación, se clasifican:

Fuentes primarias

Según Grande y Abascal (2014), las fuentes de tipo primarias son aquellas que el investigador crea en concreto para su estudio. La información no existe en el momento en que se necesita, por lo tanto, surge la necesidad de crearla por medio de técnicas como encuestas, entrevistas, la observación o reuniones de grupo.

Para la presente propuesta, las fuentes de información primaria utilizadas serán la entrevista y observación. Estas serán la fuente de información para recolectar datos de la empresa, sus actividades y técnicas, ya que esta información solo existe dentro de la compañía, por lo tanto, hay que crearla por medio de las técnicas anteriormente mencionadas.

Fuentes secundarias

La información que ha sido previamente creada es existente y está disponible para acceder a ella, es de tipo secundaria. Esta se obtiene de forma más rápida, económica y con menor esfuerzo que la secundaria. Según Grande y Abascal (2014), puede ser utilizada para complementar a la primaria. Las secundarias pueden presentarse en forma de datos, informes, páginas web o metodologías y libros. Para el caso concreto de la presente investigación, se utilizan las fuentes secundarias, tales como páginas web, documentos y libros extraídos de internet.

Fuentes terciarias

Entre las fuentes de información de tipo terciarias, se encuentran resúmenes, bibliografías, manuales e incluso guías físicas o virtuales que contienen información sobre las secundarias. Se hará uso de las terciarias para la actual propuesta, con las siguientes fuentes: bibliografías físicas y digitales, tutoriales y manuales obtenidos de internet.

Variables

Las variables, según Hernández (2017), son propiedades o características de fenómenos, hechos o entidades físicas que pueden oscilar y su variación es susceptible al medirse u observarse. Pueden adquirir diferentes valores y es relevante incluirlos porque estos factores pueden cambiar durante el curso de la investigación.

Tabla 1
Variables

Objetivo específico	Variable	Definición conceptual	Definición operacional	Definición instrumental
Identificar la situación actual del proceso de *testing* en la empresa.	Proceso de *testing*	"Proceso de evaluación y verificación de que un producto o aplicación de *software* hace lo que se supone que debe hacer" (International Business Machines Corporation [IBM], s.f., párr. 1).	Entrevista	Formulario de entrevista.
Analizar los módulos y funcionalidades del sistema eDoc, para la creación de casos de prueba automatizados.	Módulos y funcionalidades.	Módulos y funcionalidades: "Un módulo es un componente de *software* o parte de un programa que contiene una o más rutinas. Uno o más módulos desarrollados de forma independiente componen un programa" (Techopedia, 2021, párr. 1).	Entrevista Observación	Formulario de entrevista. Observación del sistema eDoc.
	Casos de prueba.	Casos de prueba: "Es un escenario de prueba que mide la funcionalidad en un conjunto de acciones o condiciones para verificar el resultado esperado" (Parasoft, 2021, párr. 3).		

Objetivo específico	Variable	Definición conceptual	Definición operacional	Definición instrumental
Crear el documento de plan de pruebas, donde se registrarán los casos de pruebas, el alcance de la automatización de pruebas, tipos de pruebas, entre otros.	Plan de pruebas.	"Un plan de prueba a menudo describe un documento que identifica el cronograma de proyectos del equipo de control de calidad, así como las diversas tareas que asumirán" (SmartBear, 2021b, párr. 2).	Crear plan de pruebas	Principios de metodología Agile. Plantillas de plan de pruebas.
	Tipos de pruebas.	"Un tipo de prueba es un grupo de actividades de prueba destinadas a probar características específicas de un sistema de *software*" (International Software Testing Qualifications Board [ISTQB], 2019, p. 40).		
Elaborar según los requerimientos y la información recolectada del plan de pruebas, el prototipo de automatización.	Prototipo.	"Un prototipo es una versión inicial compacta de la solución o parte de la solución de un sistema construido en un breve periodo de tiempo" (IBM, 2006, párr. 1).	Requerimientos	Lenguajes de programación y librerías: C#, Selenium.

Objetivo específico	Variable	Definición conceptual	Definición operacional	Definición instrumental
Programar la propuesta de automatización de las pruebas, que se ejecutarán sobre el sistema eDoc.	Programar.	“Diseñar la manera en cómo la computadora realizara un proceso sobre datos para obtener resultados, por medio de la codificación” (Román y Ramírez, 2012, p. 8).	Programar automatización	Lenguaje de programación#, y librería Selenium.
Realizar las pruebas pertinentes para garantizar que todos los requerimientos están incluidos.	Requerimientos.	“Los requerimientos corresponden a las necesidades o requisitos del cliente o patrocinador y que se espera sean solventadas en el resultado del proyecto” (Rojas et al., 2020, p. 50).	Revisión del prototipo creado	Casos de prueba.

Población

La población es el conjunto de personas u objetos que participan del problema que fue planteado en una investigación. Para la presente propuesta, son todas las personas colaboradoras entre 20 y 45 años que laboran en la compañía Gurusoft en Costa Rica.

Muestra

La muestra es el subconjunto de la población en el que se llevará a cabo el proyecto y la que mayormente se utilizará para extraer la información y recolectar los datos. La muestra en este caso será el personal perteneciente al Departamento de TI como el equipo de desarrollo y el *project manager*. A continuación, se muestra el cálculo del tamaño de la muestra (Oncins de Frutos, s.f.):

N = 40

Z = 1.960

P = 50.00%

Q = 50.00%

E = 3.00%

$$n= \frac{N* Za^2*p*q}{e^2*(N-1)+Za^2*p*q}$$

$$n = \frac{40 \ * \ 1.960^2 * 0.50 * 0.50}{0 - 03^2 * (40 - 1) + 19.60^2 * 0.50 * 0.50}$$

$$n = \ 38.59$$

Donde:

n = Tamaño de muestra

N = Tamaño de población

Z = Nivel de confianza

E = Error de estimación estimado

p = Probabilidad de que ocurra el evento

q = (1-p) = Probabilidad de que no ocurra el evento

Instrumentos de recolección de datos

Observación. Según Barrantes (2013), debido a que el enfoque es cualitativo, uno de los principales instrumentos en este es la observación. La observación significa utilizar los sentidos, con el fin de obtener información y datos para el análisis.

Para el caso concreto de esta investigación, se utiliza la observación, con el objetivo de que la estudiante tenga una mejor percepción e interpretación de las actividades de la empresa, así como del sistema que estará bajo las pruebas automatizadas.

Entrevista. La entrevista básicamente es una conversación, por lo general de manera oral entre dos personas, se puede dar de forma telefónica, presencial y también con el avance de la tecnología se puede realizar por video llamada o escrita a través de un chat. Entre los tipos de entrevista, según Barrantes (2013), que se utilizan para la presente investigación se encuentran:

- Controlada: la conversación recae sobre el entrevistador. Quien realiza una guía o un cuestionario que se ha preparado previamente y lo utilizará durante la entrevista.
- No estructurada: la conversación recae esta vez en el entrevistado, quien narra experiencias y situaciones; el entrevistador puede realizar preguntas para guiar y mantener el hilo de la conversación.

Se hará uso de la entrevista para la recolección de datos, aprendizaje de actividades y acontecimientos que no se ven a simple vista o no quedan claramente explicados por medio de la observación. Así mismo, proporcionará información de escenarios, situaciones y actividades de la empresa.

Recolección y análisis de datos

Uno de los instrumentos utilizados para la recolección de datos fue la entrevista, se realizaron cuatro entrevistas a diferentes personas con distintos roles en la empresa.

La primera entrevista se realizó a la Directora de Operaciones (COO) de la empresa, con quien se obtuvo un conocimiento general de procesos de la empresa y del funcionamiento de la aplicación.

Asimismo, se entrevistó a uno de los *project managers*, quien tiene el rol de planear y monitorear los proyectos que se llevan a cabo dentro de la empresa. Es quien debe ejecutar las fases del ciclo de vida de gestión de proyectos, identifica la necesidad de negocio para construir una

solución, y quien tiene un mayor contacto con los clientes de un proyecto de una empresa. Se documentaron dos entrevistas con el Project Manager, en donde se explicó de forma más detallada el funcionamiento de la aplicación y algunas características técnica.

Por último se realizó una entrevista a uno de los colaboradores en el área de soporte y desarrollo, donde se obtuvieron características técnicas de la aplicación. El personal de desarrollo, son quienes desarrollan todos los requerimientos o historias de usuario, por lo tanto, son encargados de insertar código o realizar modificaciones a la aplicación. Además, el personal de soporte, son quienes brindan ayuda a los usuarios de la aplicación, en caso de que requieran asistencia técnica.

Cada colaborador proporcionó información valiosa, para entender cómo desarrollar las pruebas para el presente proyecto. A continuación se detallan los resultados de las entrevistas.

Tabla 2 Entrevistas

Entrevista 1	
Tipo de entrevista	Estructurada
Rol del entrevistado	Gerente
Medio	Skype
¿Existen diferentes ambientes para las aplicaciones?	Actualmente existen tres ambientes diferentes para la aplicación Edoc. El de desarrollo, de producción y el de calidad.
¿Quiénes realizan las pruebas en el desarrollo y cómo son llevadas a cabo?	Cuando se trata de desarrollo nuevo, las pruebas en el sistema son realizadas por los mismos desarrolladores, ellos testean manualmente lo que ellos mismos desarrollan en el ambiente de desarrollo. Seguidamente el personal de soporte y el Project Manager realiza las pruebas en el ambiente de calidad. Por último, se le notifica al cliente para que, por su parte también realice pruebas en el ambiente de calidad para que acepte el producto.
¿Cuáles serían las aplicaciones por automatizar y cuáles considera que requieren automatización?	Todas las aplicaciones tienen módulos que pueden ser automatizados, pero los de más uso son la aplicación de eDoc empresarial, la cuál es la aplicación que utilizan más los clientes.
Las pruebas serán de tipo funcionales, y en el UI, ¿cuáles serían los módulos	Los módulos en la aplicación pueden variar de acuerdo con las necesidades de los clientes, pero siempre se mantienen módulos generales, como administración y

que considera más importantes para ser automatizados?	configuración, los cuales están presentes para todos los clientes.
¿Cuáles considera que son funciones críticas dentro de las aplicaciones?	Las funciones críticas dentro del sistema son las aplicaciones que nunca varían y que se mantienen entre todos los clientes. Por ejemplo, la administración, configuración, almacenamiento de datos y facturas.
¿Qué metodología utilizan actualmente para desarrollar?	Actualmente nos guiamos por bases de Scrum.
¿Existen documentos creados previamente para el testeo?	No existen documentos formales en temas de pruebas.
Si existen, ¿qué rol tenía la persona que documentó o creó los casos de prueba?	Tampoco existen documentos con casos de prueba formales.
¿Cuál es el lenguaje que manejan actualmente la mayoría de los desarrolladores?	La mayoría de los desarrolladores utilizan Visual Basic, C# y JavaScript.

Entrevista 2	
Tipo de entrevista	Abierta\Libre
Rol del entrevistado	Project Manager
Medio	Microsoft Teams
En esta entrevista, se le ha dado libertad al entrevistado de expresarse y explicar la funcionalidad de la aplicación eDoc para entender su funcionamiento. La entrevistadora utilizó un espacio al final para preguntas sobre el funcionamiento y dudas que se presentaron durante la explicación del entrevistado. Los temas vistos en la entrevista realizada al Project Manager se listan a continuación: • Permisos y autenticación de usuarios: permite que los clientes puedan acceder a la aplicación, cada uno cuenta con sus credenciales, si el cliente no los tiene, no puede utilizar el eDoc, de esta forma se mantienen seguros los datos. • Configuración y administración de usuarios: dentro de este módulo del sistema, se administra y configura todo lo relacionado a los clientes que utilizan la	

aplicación. Sus funcionalidades principalmente son temas de seguridad y administración de sus datos.

- Módulo de facturación electrónica: en este módulo se encuentra todo lo relacionado a datos de facturas. Dentro de él se encuentran submódulos, en donde se lleva a cabo la administración, visualización de tiquetes o facturas de los usuarios.
- Consultas y reportes: en este módulo, se encuentran las herramientas para que los usuarios puedan acceder a los datos creados anteriormente.
- Funcionamiento general de la aplicación, y su funcionamiento en el *back end:* se realizó una explicación del sistema y cómo este está conectado a otros sistemas de la empresa, para la conectividad con el Ministerio de Hacienda.

Entrevista 3	
Tipo de entrevista	Estructurada
Rol de entrevistado	Project Manager
Medio	Microsoft Teams
¿En qué tecnología están construidas las aplicaciones?	El sistema eDoc está construido bajo el framework de ASP.net, y se utiliza como lenguaje de programación Visual Basic. Además de eso AJAX, JavaScript, HTML, CSS, Bootstrap en el lado de front end.
¿Hay conocimiento de cuál es el navegador más usado por los usuarios?	El principal navegador que los usuarios utilizan para usar el eDoc, seria Google Chrome, y Microsoft Edge ya que la mayoría utiliza Windows.
Actualmente, ¿qué tan parecidos son los ambientes de desarrollo y calidad con el de producción?	Son idénticos, solamente varía información y datos de producción, ya que este ambiente es el que los clientes utilizan, por lo tanto, tiene datos reales. Pero todas las funciones y módulos se mantienen.
¿Considera que existe la oportunidad de que usted mismo sea quien utilice la automatización de las pruebas?	Sí, existe la oportunidad de que la utilice. Además de otros Project Managers para hacer pruebas e incluso desarrolladores.

Entrevista 4	
Tipo de entrevista	Estructurada\Libre
Rol de entrevistado	Soporte/Desarrollo
Medio	Microsoft Teams
¿Hay conocimiento de cuál es el navegador más usado por los usuarios?	En primer lugar, estaría Chrome como el más utilizado, y en segundo lugar sería Edge.
¿Cuáles son las funcionalidades donde suelen haber más mantenimiento o cambios?	Esto depende de los clientes, a veces hay clientes que requieren más cambios o mantenimientos comparados con otros clientes. Sin embargo, los módulos principales se suelen mantener igual para todos los clientes, y se realizan mantenimientos cada vez que se necesite hacer un cambio muy grande, lo cual no sucede muy a menudo.
¿Existen errores comunes en las aplicaciones? Ya sea que se presentaron una vez y se han vuelto a presentar.	Si ha pasado, incluso sucede que cuando se realizan pruebas de alguna funcionalidad, no se llega a tener la certeza de que funcione de manera óptima, por lo que a la hora de enviar los cambios al ambiente de producción algún código puede que deje de funcionar o el nuevo código no funciona de la mejor manera.
¿El código existente en el ambiente de pruebas y de desarrollo es exactamente el mismo?	Es el mismo, los ambientes de desarrollo y de calidad son una réplica de producción, lo que cambia es el Captcha que está deshabilitado en el ambiente de pruebas y la información que contiene el ambiente de producción.

CAPÍTULO IV. ANÁLISIS DE RESULTADOS

En este capítulo, se desarrollan los requerimientos del proyecto. Los requerimientos expresan la funcionalidad de un sistema, indican cómo debe comportarse y especifican las actividades de este. Debido a la naturaleza del presente proyecto, el cual está enfocado en pruebas, la mayor parte de los requerimientos se convierten en casos de prueba, ya que, al ser un prototipo de automatización, se requiere documentar cuáles serán las pruebas que se van a ejecutar en el sistema.

Tabla 3
Requerimientos

Requerimiento Id:	**REQ 001** **TC 001**
Módulo:	Login
Descripción:	Login: realizar el login de la aplicación con las credenciales correctas para comprobar funcionamiento de Inicio de Sesión para usuarios. Una vez iniciada la sesión, se confirma que se muestra la pantalla de bienvenida del sistema.
Datos necesarios	Nombre de usuario el cual es configurado y predefinido por la empresa: Genesis_cubillo. Contraseña: previamente creado por el usuario: testtest.

Requerimiento Id:	**REQ 002** **TC 002**
Módulo:	Login
Descripción:	Login: realizar el login de la aplicación con credenciales incorrectas para comprobar funcionamiento de validación de usuarios. Se debe verificar que, al insertar las credenciales incorrectas, se muestra un mensaje de error.
Datos necesarios	Nombre de usuario y contraseña inexistentes en el sistema. Nombre de usuario: Genesis_cubillo

Requerimiento Id:	**REQ 003** **TC 003**
Módulo:	Facturación electrónica
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, seleccionar cargar archivo de XML. Se debe subir un archivo XML, para verificar que el sistema esté cargando archivos.
Datos necesarios	Archivo XML con formato correcto según los lineamientos de Hacienda que se encuentran a continuación: https://www.hacienda.go.cr/ATV/ComprobanteElectronico/docs/esquemas/2016/v4.3/ANEXOS%20Y%20ESTRUCTURAS_V4.3.pdf

Requerimiento Id:	**REQ 004** **TC 004**
Módulo:	**Facturación electrónica**
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Nota de Crédito, seleccionar cargar archivo de XML. Se debe subir un archivo XML, para verificar que el sistema esté cargando archivos.
Datos necesarios	Archivo XML con formato correcto según los lineamientos de Hacienda que se encuentran a continuación: https://www.hacienda.go.cr/ATV/ComprobanteElectronico/docs/esquemas/2016/v4.3/ANEXOS%20Y%20ESTRUCTURAS_V4.3.pdf

Requerimiento Id:	**REQ 005** **TC 005**
Módulo:	Facturación electrónica
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Nota de Débito, seleccionar cargar archivo de XML. Se debe subir un archivo XML, para verificar que el sistema esté cargando archivos.

Datos necesarios	Archivo XML con formato correcto según los lineamientos de Hacienda que se encuentran a continuación: https://www.hacienda.go.cr/ATV/ComprobanteElectronico/docs/esquemas/2016/v4.3/ANEXOS%20Y%20ESTRUCTURAS_V4.3.pdf

Requerimiento ID:	**REQ 006** **TC 006**
Módulo:	Facturación electrónica
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, seleccionar cargar archivo de XML que contenga una identificación de compañía que no esté previamente guardada en el sistema para verificar validación.
Datos necesarios	Archivo de Factura XML con formato correcto según los lineamientos de Hacienda que se encuentran a continuación: https://www.hacienda.go.cr/ATV/ComprobanteElectronico/docs/esquemas/2016/v4.3/ANEXOS%20Y%20ESTRUCTURAS_V4.3.pdf

Requerimiento ID:	**REQ 007** **TC 007**
Módulo:	Facturación electrónica
Submódulo	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, seleccionar cargar archivo de XML. Se debe subir un archivo XML que contenga datos faltantes para verificar que el sistema no esté creando un archivo PDF, y se muestre el mensaje de error informando que el archivo posee errores.
Datos necesarios	Archivo XML sin la identificación de empresa

Requerimiento ID:	**REQ 008** **TC 008**
Módulo:	Facturación electrónica
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, se debe subir un archivo que sea diferente al tipo XML, para verificar que el sistema no esté creando un archivo PDF, y se muestre el

	mensaje de error informando que el tipo de archivo es incorrecto y no se acepta.
Datos necesarios	Archivo de tipo hoja de cálculo.

Requerimiento ID:	**REQ 009** **TC 009**
Módulo:	Facturación electrónica
Submódulo:	Factura electrónica
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, seleccionar cargar archivo de XML. Se debe subir un archivo XML que no sea de tipo factura, para verificar que el sistema no esté creando un archivo PDF, y se muestre el mensaje de error informando que el archivo no es de tipo factura.
Datos necesarios	Archivo de tipo Nota de Crédito o Débito.

Requerimiento ID:	**REQ 010** **TC 010**
Módulo:	Administración
Submódulo:	Permisos de usuarios-Mantenimiento de roles
Descripción:	En el menú de Administración, dirigirse al submenú de permisos de usuarios, seleccionar la opción de mantenimiento de roles. Agregar un rol nuevo, seguidamente guardar y verificar que el rol haya sido guardado en la lista de roles que se presenta luego de haber almacenado el rol.
Datos necesarios	Agregar el rol con el siguiente formato: Nombre: “Rol”+ Agregar Fecha Actual Descripción: “Automatizacion Test”

Requerimiento ID:	**REQ 011** **TC 011**
Módulo:	Administración
Submódulo:	Permisos de usuarios-Mantenimiento de roles
Descripción:	Buscar el rol previamente creado, en el requerimiento anterior, insertando el nombre en las cajas de texto de búsqueda.

	Una vez encontrado en la lista, seleccionarlo para desactivarlo y, por último, el sistema debe mostrar que este dato se encuentre como inactivo.
Dependiente de:	REQ 010
Datos necesarios:	Datos insertados previamente en el REQ 009 Nombre: “Rol” + Agregar Fecha Actual Descripción: “Automatizacion Test

Requerimiento ID:	**REQ 012** **TC 012**
Módulo:	Administración
Submódulo:	Clientes y proveedores.
Descripción:	En el menú de Administración, dirigirse al submenú de clientes y proveedores. Corroborar que se despliegue la lista de clientes y proveedores e insertar un nuevo dato. Posterior a esto, verificar que el dato haya sido almacenado realizando una búsqueda por medio de las cajas de texto de búsqueda.
Datos necesarios:	Para guardar un nuevo proveedor, se solicitan los siguientes datos: número de identificación, email, razón social, cliente. Estos se deben llenar con la siguiente información: Número de identificación: Número aleatorio Email: genesis_cubillo@gurusoft.co.cr Razón social: “Test Automatico”+Agregar Fecha Actual. ¿Es cliente?: Sí

Requerimiento ID:	**REQ 013** **TC 012**
Módulo:	Administración
Submódulo:	Clientes y proveedores.
Descripción:	Buscar el cliente previamente creado, en el requerimiento anterior, insertando el nombre en las cajas de texto de búsqueda. Una vez encontrado en la lista, seleccionarlo para desactivarlo y, por último, el sistema debe mostrar que este dato se encuentre como inactivo.
Dependiente de:	REQ 012

Datos necesarios:	Datos insertados previamente en el REQ 012. Nombre: “Test Automatico”+Agregar Fecha Actual Descripción: “Automatizacion Test”

Requerimiento ID:	**REQ 014** **TC 014**
Módulo:	Administración
Submódulo:	Usuarios internos
Descripción:	En el menú de Administración, dirigirse al submenú de Usuarios Internos. Corroborar que se despliegue la lista de usuarios e insertar un nuevo dato. Posterior a esto, verificar que el dato haya sido almacenado realizando una búsqueda por medio de las cajas de texto de búsqueda.
Datos necesarios:	Para guardar un nuevo usuario, el sistema solicita los siguientes datos: usuario, email, nombre, sucursal. Estos deben ser llenados con la siguiente información: Usuario: “UsuarioAutomatico” seguidamente de la fecha en formato dd/mm/aaaa Email: genesis_cubillo@gurusoft.co.cr Nombre: Usuario Automatico

Requerimiento ID:	**REQ 015** **TC 015**
Módulo:	Administración
Submódulo:	Usuarios Internos
Descripción:	En el menú de Administración, dirigirse al submenú de Usuarios Internos. Buscar el cliente previamente creado, en el requerimiento anterior, insertando el nombre en las cajas de texto de búsqueda. Una vez encontrado en la lista, seleccionarlo para desactivarlo y, por último, el sistema debe mostrar que este dato se encuentre como inactivo.
Dependiente de:	REQ 014

Datos necesarios:	Datos insertados previamente en el REQ 015. Usuario: “UsuarioAutomatico” seguidamente de la fecha en formato dd/mm/aaaa Email: genesis_cubillo@gurusoft.co.cr Nombre: Usuario Automatico

Requerimiento ID:	**REQ 016** **TC 016**
Módulo:	Administración
Submódulo:	Notificación Cliente/Proveedor
Descripción:	En el menú de Administración, dirigirse al submenú de notificar clientes y proveedores. Agregar un mensaje de notificación con un título y mensaje, seguidamente, buscar el cliente y seleccionarlo. Guardar los cambios y verificar que en la lista de notificaciones se muestre. Por último, eliminar la notificación recién guardada y esta no debe mostrarse en el sistema.
Datos necesarios:	Para guardar una nueva notificación, el sistema solicita los siguientes datos: título y mensaje. Estos deben ser llenados con la siguiente información: Título: automatización seguida de la fecha en formato dd/mm/aaaa Mensaje: “Esta es una prueba automática, por favor, omitir.”

Requerimiento ID:	**REQ 017** **TC 017**
Módulo:	Administración
Submódulo:	Proveedores y Empresarios
Descripción:	En el menú de Administración, dirigirse al submenú de Proveedores Empresarios. Corroborar que se despliegue la lista de clientes y proveedores e insertar un nuevo dato. Posterior a esto, verificar que el dato haya sido almacenado realizando una búsqueda por medio de las cajas de texto de

	búsqueda y, por último, eliminarlo y confirmar que el sistema no esté presentando el dato recién eliminado.
Datos necesarios:	Para guardar un nuevo usuario, el sistema solicita los siguientes datos: número de identificación, email, razón social, estado, número de SAP. Los cuales deben ser llenados con la siguiente información: Número de identificación: número aleatorio. Email: genesis_cubillo@gurusoft.co.cr Razón Social: “TestAutomático” seguidamente de la fecha en formato dd/mm/aaaa Número SAP: número aleatorio.

Requerimiento ID:	**REQ 018** **TC 018**
Módulo:	Administración
Submódulo:	Usuarios Internos
Descripción:	En el menú de Administración, dirigirse al submenú de Usuarios Internos y el sistema debería desplegar una lista de usuarios internos. Se debe buscar un usuario guardado previamente, y editar el nombre del usuario.
Datos necesarios:	Nombre a buscar: TEST Nombre de Usuario: “UsuarioAutomatico” seguidamente de la fecha en formato dd/mm/aaaa hh:mm

Requerimiento ID:	**REQ 019** **TC 019**
Módulo:	Administración
Submódulo:	Cliente Proveedor
Descripción:	En el menú de Administración, dirigirse al submenú de Clientes y Proveedores. El sistema debería desplegar una lista de clientes y proveedores. Se debe buscar un cliente guardado previamente, y editar la razón social. El sistema deberá guardar los cambios.
Datos necesarios:	Nombre del cliente a editar: TEST Razón Social: “Test Automatico” seguido de la fecha en formato dd/mm/aaaa hh:mm

Requerimiento ID:	**REQ 020**

	TC 020
Módulo:	Administración
Submódulo:	Proveedor Empresario
Descripción:	En el menú de Administración, dirigirse al submenú de Proveedor y Empresario. El sistema debería desplegar una lista de proveedores. Se debe buscar un proveedor guardado previamente, y editar la razón social. El sistema deberá guardar los cambios.
Datos necesarios:	Nombre proveedor a editar: 3287442 Razón Social: “TestAutomatico” seguidamente de la fecha en formato dd/mm/aaaa hh:mm Número SAP: Número aleatorio.

Requerimiento ID:	**REQ 021** **TC 021**
Módulo:	Administración
Submódulo:	Permisos de usuarios-Mantenimiento de roles
Descripción:	En el menú de Administración, dirigirse al submenú de Roles. El sistema debería desplegar una lista de roles. Se debe buscar un rol guardado previamente, y editar la descripción del rol. El sistema deberá guardar los cambios.
Datos necesarios:	Nombre rol a editar: “AUTOMATIZACION TESTEO” Descripción: agregar fecha y hora actual en formato dd/mm/aaaa hh:mm

Requerimiento ID:	**REQ 022** **TC 022**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos
Descripción:	En el menú de Consultas y Reportes, seleccionar el submenú de Documentos Emitidos y Docs. por permisos de rol. El sistema despliega la tabla para realizar una búsqueda de los documentos emitidos por el rol. A continuación, se debe seleccionar el rango de fechas, el tipo de documento y estado de documento. Seguidamente, se verifica que el sistema despliegue la tabla con resultados.

Datos necesarios:	Criterio de Búsqueda: Tipo de documento: todos Estado de documento: todos Fechas desde: 1/4/2021 Fecha hasta: 30/9/2021

Requerimiento ID:	**REQ 023** **TC 023**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos
Descripción:	En el menú de Consultas y Reportes, seleccionar el submenú de Documentos Emitidos y Docs. por permisos de rol. El sistema despliega la tabla para realizar una búsqueda de los documentos emitidos por el rol. A continuación, se debe seleccionar el rango de fechas, el tipo de documento, y estado de documento. Seguidamente, se verifica que el sistema despliegue la tabla con resultados y se debe guardar la clave del primer resultado. Una vez guardado este valor, redirigir el sistema al submenú de Dos por criterio de búsqueda. El sistema deberá mostrar una tabla para realizar una búsqueda. En esta tabla se debe insertar el tipo de criterio de búsqueda y el valor de la clave previamente copiada. Una vez realizada la búsqueda, confirmar que el sistema está mostrando el resultado correcto en la tabla.
Dependiente de:	REQ 022
Datos necesarios:	Criterio de Búsqueda: Clave de Comprobante Descripción: Valor de la clave copiada.

Requerimiento ID:	**REQ 024** **TC 024**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos
Descripción:	En el menú de Consultas y Reportes, seleccionar el submenú de Documentos Emitidos y Reporte de Emisión Detallado.

	El sistema despliega la tabla para realizar una búsqueda de los documentos emitidos por el rol. A continuación, se debe seleccionar el rango de fechas, el tipo de documento y estado de documento. Seguidamente, se verifica que el sistema despliegue la tabla con resultados.
Datos necesarios:	Criterio de Búsqueda: Tipo de documento: todos Estado de documento: autorizado Fechas desde: 1/10/2021 Fecha hasta: 10/10/2021

Requerimiento ID:	**REQ 025** **TC 025**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos
Descripción:	En el menú de Consultas y Reportes, seleccionar el submenú de Documentos Emitidos y Reporte de Emisión Detallado en colones. El sistema despliega la tabla para realizar una búsqueda de los documentos emitidos por el rol. A continuación, se debe seleccionar el rango de fechas, el tipo de documento y estado de documento. Seguidamente, se verifica que el sistema despliegue la tabla con resultados.
Datos necesarios:	Criterio de Búsqueda: Tipo de documento: todos Estado de documento: autorizado Fechas desde: 1/10/2021 Fecha hasta: 10/10/2021

Requerimiento ID:	**REQ 026** **TC 026**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos

Descripción:	En el menú de Consultas y Reportes, seleccionar el submenú de Documentos Emitidos y Reporte de Emisión Detallado. El sistema despliega la tabla para realizar una búsqueda de los documentos emitidos por el rol. A continuación, se debe seleccionar la opción de búsqueda de RUC. Seguidamente, buscar el nombre de la razón social y seleccionarla. Una vez seleccionada la razón social, se debe confirmar que el campo de Número RUC ha sido rellenado. Seguidamente, rellenar los campos de tipo de documento, estado de documento y fechas, y realizar la búsqueda para confirmar que el sistema esté rellenando la tabla con resultados.
Datos necesarios:	RUC: Gurusoft. Tipo de documento: Nota de Crédito Estado de documento: Autorizado Fechas desde: 1/10/2021 Fecha hasta: 10/10/2021

Requerimiento ID:	**REQ 027** **TC 027**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos
Descripción:	Realizar la inserción de agregar Notificaciones Cliente y Proveedores del requerimiento con identificación: REQ 016. Posterior a la inserción de la notificación, dirigirse al menú de Consultas y Reportes, y al submenú de Notificaciones Cliente Proveedor. Seguidamente, buscar la notificación recién creada.
Dependiente de:	REQ 016
Datos necesarios:	Título de notificación: “NotificacionAutomatica” seguido de fecha actual en formato: mm/dd/aaa hh:ss:mm Estado: Todos

Requerimiento ID:	**REQ 028** **TC 028**
Módulo:	Consultas y Reportes
Submódulo:	Documentos Emitidos

Descripción:	Dirigirse al menú de Consultas y Reportes, y al submenú de Envío de Mail. Seguidamente, realizar una búsqueda para confirmar que el sistema muestre los resultados correctos según la búsqueda realizada.
Datos necesarios:	Estado: Envío de Mail: Enviado Estado de documento: Autorizado Destinatario: ivog28@gmail.com Asunto: Portal Interno Usuario Creación: Genesis Cubillo

Requerimiento Id:	**REQ 029** **TC 029**
Módulo:	Seguridad
Descripción:	Las pruebas de seguridad incluirán dos escenarios, los cuales van a realizar la comprobación de validaciones en la aplicación. 1. Comprobación de SQL Injection, para obtener el tipo de error que muestra la aplicación cuando se inserta un apóstrofe en algún campo de texto, de esta manera, se comprueba que esté conectado directamente a la base de datos. 2. Introducir un texto de más de 1000 caracteres en un campo de texto para comprobar si la aplicación se desborda fácilmente.
Datos necesarios:	Introducir apóstrofe en campos de texto. Introducir un texto de más de 1000 caracteres.

Requerimiento Id:	**REQ 030** **TC 030**
Modulo	Facturas
Submódulo	Consulta detallada

Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, y a continuación se debe seleccionar fechas que superen los tres meses de periodo entre ellas. Se debe confirmar, que el sistema muestra el pop up de notificación de que no acepta fechas mayores a los tres meses.
Datos necesarios:	Fechas: Desde: 01/04/2021 Hasta: 30/09/2021

Requerimiento Id:	**REQ 031** **TC 031**
Modulo	Facturas
Submódulo	Documentos recibidos
Descripción:	En el menú de Recepción Electrónica, dirigirse al menú de Facturas, y Aprobar documentos recibidos. A continuación, se debe verificar que, si no hay resultados en la tabla de facturas, se esté mostrando el mensaje de información de que no existen documentos pendientes para gestionar.

Requerimiento Id:	**REQ 032**
Módulo:	Reportes de resultado de automatización
Descripción:	Luego de ejecutar las pruebas, se crearán los reportes con los resultados de cada prueba ejecutada. Estos reportes se almacenan dentro de una carpeta llamada “Resultados” con la fecha y hora correspondiente.
Datos necesarios:	Ejecutar las pruebas automáticas para obtener resultados y, de esta manera, se generen los reportes.

Requerimiento Id:	**REQ 033**
Descripción:	Creación de reglas para repositorio de proyecto.

Datos necesarios:	Asignación de reglas para la protección del código del repositorio en GitHub.

Matriz de Requerimientos

Como se mencionó anteriormente, los requerimientos del presente proyecto se convierten en test cases, por lo que a continuación se muestra una matriz de los requerimientos que se convierten en casos de pruebas que están automatizadas, así mismo como el objetivo que deben cumplir, el tipo de prueba, y su prioridad respectivamente.

Tabla 4
Matriz de requerimientos

Id Requerimiento	Objetivo	Tipo	Prioridad
TC 001 REQ 001	Verificar inicio de sesión exitoso con credenciales correctos. Verificar elementos están presentes en el inicio	Funcional Regresión	Alta
TC 003 REQ 003	Verificar subida de archivos exitosa.	Funcional Regresión	Alta
TC 004 REQ 004	Verificar subida de archivos exitosa.	Funcional Regresión	Alta
TC 005 REQ 005	Verificar subida de archivos exitosa.	Funcional Regresión	Alta
TC 010 REQ 010	Verificar guardado de datos exitosa.	Funcional Regresión	Alta
TC 011 REQ 011	Verificar eliminación de datos exitosa	Funcional Regresión	Alta
TC 012 REQ 012	Verificar guardado de datos exitosa.	Funcional Regresión	Alta
TC 013 REQ 013	Verificar eliminación de datos exitosa	Funcional Regresión	Alta
TC 014 REQ 014	Verificar guardado de datos exitosa.	Funcional Regresión	Alta

TC 013 REQ 013	Verificar eliminación de datos exitosa	Funcional Regresión	Alta
TC 014 REQ 014	Verificar guardado de datos exitosa.	Funcional Regresión	Alta
TC 015 REQ 015	Verificar eliminación de datos exitosa	Funcional Regresión	Alta
TC 016 REQ 016	Verificar inserción y almacenado de datos. Verificar eliminación de datos de forma exitosa.	Funcional Regresión	Alta
TC 017 REQ 017	Verificar guardado de datos exitosa.	Funcional Regresión	Alta
TC 018 REQ 018	Verificar edición y guardado de datos exitosa.	Funcional Regresión	Alta
TC 019 REQ 019	Verificar edición y guardado de datos exitosa.	Funcional Regresión	Alta
TC 020 REQ 020	Verificar edición y guardado de datos exitosa.	Funcional Regresión	Alta
TC 021 REQ 021	Verificar edición y guardado de datos exitosa.	Funcional Regresión	Alta
TC 022 REQ 022	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 023 REQ 023	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 024 REQ 024	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta

TC 025 REQ 025	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 026 REQ 026	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 027 REQ 027	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 028 REQ 028	Verificar búsqueda de datos previamente almacenados. Verificar elementos e interacción en la página funciona de manera correcta.	Funcional Regresión	Alta
TC 006 REQ 006	Verificar validaciones que la aplicación debe mostrar cuando se realizan ciertas acciones.	Negativo	Media
TC 007 REQ 007	Verificar validaciones que la aplicación debe mostrar cuando se realizan ciertas acciones.	Negativo	Media
TC 008 REQ 008	Verificar validaciones que la aplicación debe mostrar cuando se realizan acciones específicas.	Negativo	Media
TC 009 REQ 009	Verificar validaciones que la aplicación debe mostrar cuando se realizan acciones específicas.	Negativo	Media
TC 002 REQ 002	Verificar validaciones que la aplicación debe mostrar cuando se realizan acciones específicas.	Negativo	Media
TC 029 REQ 029	Verificar si la aplicación es vulnerable a un intento de injection.	Seguridad	Baja
TC 030 REQ 030	Verificar validaciones que la aplicación debe mostrar cuando se realizan ciertas acciones.	Negativo	Baja
TC 031 REQ 031	Verificar validaciones que la aplicación debe mostrar cuando se realizan ciertas acciones.	Negativo	Baja

Comparación de pruebas manuales con automatizadas

En las siguientes tablas se muestran la descripción de escenarios o pruebas que se ejecutan de forma manual, así como su tiempo en ejecución. Para luego demostrar el tiempo que tardan en ejecutarse estos procesos con la ejecución de pruebas tal y como lo muestra la figura 4.

Tabla 5
Ejecución de pruebas de forma manual

Modulo	Acción o Prueba manual

Administración	• Clientes y proveedores
	  • Mantenimientos de clientes y proveedores: Adición, edición, y eliminación.
	  • Búsquedas de clientes y proveedores.
	  • Mantenimientos de notificaciones para clientes y proveedores: Inserción, búsqueda, edición y eliminación de notificaciones.
	• Usuarios Internos
	  • Mantenimientos de usuarios internos: Adición, edición, y eliminación.
	  • Búsquedas de usuarios internos.
	• Proveedores Empresarios
	  • Mantenimientos de proveedores y empresarios: Adición, edición, y eliminación.
	  • Búsquedas de proveedores empresarios.
	• Roles
	  • Mantenimientos de roles: Adición, edición, y eliminación.
	  • Búsquedas de roles.

Tiempo total en ejecución manual:	1 hora

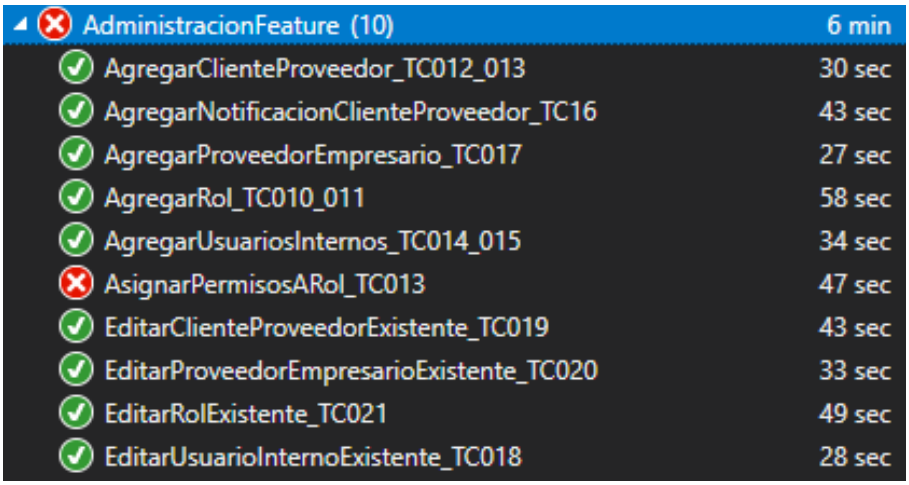
Modulo	Acción o Prueba Manual
Facturación	• Carga de Facturas. • Carga de Notas de Crédito y Débito. • Validaciones en la carga de documentos.

Tiempo total en ejecución manual:	30 minutos

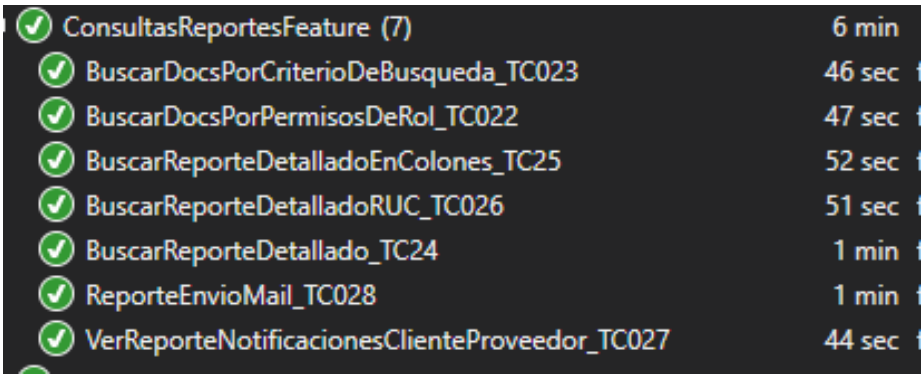
Modulo	Acción o Prueba Manual
Consultas y reportes	• Búsqueda de documentos por medio de criterios de búsqueda. • Filtrado de búsqueda.
Tiempo total en ejecución manual:	30 minutos

Figura 4
Ejecución de pruebas de forma automatizada

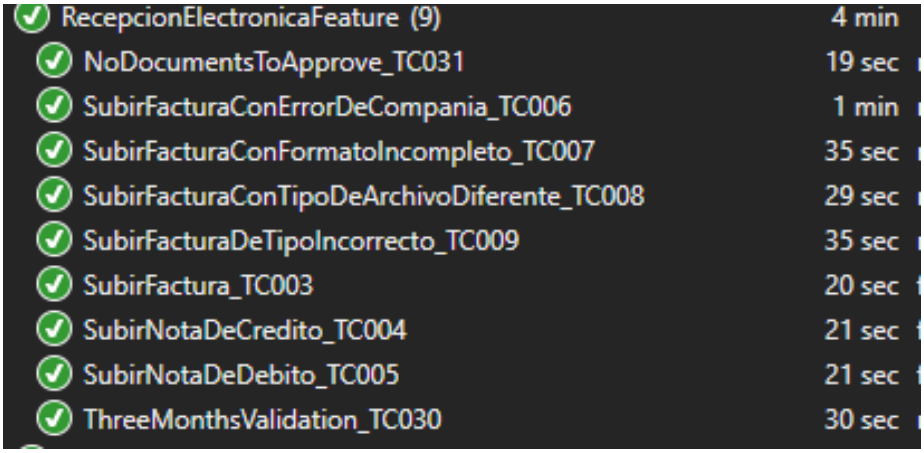
Tiempo total en ejecución automática de Administración: 6 minutos



Tiempo total en ejecución automática de Consultas y Reportes: 6 minutos



Tiempo total en ejecución automática de Facturación: 4 minutos



CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

Conclusiones

En la actualidad, existen proyectos y empresas que no tienen un adecuado proceso de realización de pruebas, o lo tienen, pero no poseen el profesional requerido para esta fase. Lo que conlleva a una definición y ejecución de pruebas ineficiente o casi nula. Por lo que la actividad de testeo automatizado tiene un papel importante en el ciclo desarrollo.

Cada cambio en un sistema o aplicación de cualquier tipo requiere una ejecución de pruebas en el *software*, y muchas veces por falta de tiempo esto no se realiza. Por esta razón, las empresas han aumentado la actividad de testeo de forma automática en vez de manual.

En el presente proyecto, se observó la aplicación bajo prueba, para realizarle una automatización de pruebas. Se ha creado un documento donde se ha definido y documentado todo el proceso de pruebas, herramientas y las pruebas que han sido automatizadas, llamadas *test cases*. Partiendo de esto, se ha codificado una automatización que contiene los *test cases* más importantes de las funcionalidades de la aplicación.

Uno de los propósitos de la automatización de pruebas en una empresa o proyecto de *software* es reducir el tiempo en que se ejecutan las pruebas manuales, así como automatizar tareas comunes para que el personal humano pueda dedicar mayor tiempo a las pruebas de prioridad, dejando a las más comunes en la automatización. Sin embargo, esto no quiere decir que la automatización no contenga pruebas críticas. En el caso de la presente automatización, se logró desarrollar pruebas que son críticas y esenciales para la aplicación. Lo cual significa que se tiene la posibilidad de verificar que funcionas importantes continúen trabajando de manera correcta en menos tiempo

La mayor limitación encontrada durante el desarrollo de la automatización es la incapacidad de automatizar ciertos módulos importantes, tales como el envío y recepción de documentos, entre otros. Esto se debe a que la aplicación contiene funcionalidades que se intercomunican con otra, así como la necesidad de la intervención humana y la utilización de un sistema de escritorio, por lo que no es posible realizar una automatización de esto.

Una vez que los desarrolladores implementen nuevos cambios en la aplicación, ellos pueden ejecutar las pruebas de automatización en forma de regresión o de *smoke,* para asegurarse de que los cambios no tengan un impacto negativo o inesperado en alguna otra funcionalidad dentro

del sistema. Esto mejora la seguridad y confianza de los desarrolladores, al insertar nuevos cambios en la aplicación. Además de que el tiempo en ejecución de las pruebas automatizadas será menor que si se ejecutan de forma manual.

Durante la creación del presente proyecto, se logró llevar a cabo los objetivos específicos propuestos al inicio.

1. Se identificó la situación actual de la empresa, en donde se pudo observar que el proceso de pruebas no está estandarizado de forma óptima, y no poseen una automatización de pruebas. Las pruebas se realizan, pero en ocasiones no se emplean las mejores prácticas o técnicas.
2. Se han analizado los módulos y funciones del sistema, para crear *test cases* que han sido automatizados. Los casos de pruebas han sido realizados tomando en consideración la criticidad y prioridad de las funcionalidades dentro del sistema, de igual manera se han tomado en cuenta las entrevistas, y opiniones de colaboradores.
3. Un documento de plan de pruebas ha sido creado en el cual se ha descrito el proceso, características, casos de pruebas, y el alcance de la automatización. Este documento se le facilitará a la empresa, de manera que pueda complementar la propuesta y guiar a los colaboradores en el uso y ejecución de las pruebas automáticas.
4. El plan de pruebas realizado en el punto anterior se ha utilizado para realizar el prototipo de automatización, el cual ha sido revisado y aceptado por la empresa.
5. Se ha desarrollado en el lenguaje de programación seleccionado, una propuesta de automatización funcional, en donde se ejecutan casos de pruebas sobre la aplicación eDoc. Estas pruebas se encargarán de disminuir el testeo de forma manual y asistirán al equipo técnico de desarrolladores o testers para ejecutar de forma más sencilla una fase de pruebas luego de cada actualización del sistema.
6. Durante el desarrollo de la automatización, se realizaron pruebas para verificar que el *script* de los casos de prueba, estén realizando efectivamente las validaciones para cada escenario propuesto en el plan de pruebas. Se han llevado a cabo depuraciones en el código para comprobar la eliminación de errores durante la ejecución de las pruebas automáticas.

Recomendaciones

- Implementación del prototipo de automatización de pruebas: el proyecto se encuentra en el repositorio de un sistema de control de versiones, por lo que, para la implementación, depuración, y mantenimiento del proyecto se debe tener acceso al repositorio, clonar el proyecto de manera local y crear los respectivos permisos para cada miembro del personal de desarrollo que lo utilizará. Esta implementación local en las computadoras será realizada por la autora del presente proyecto y tendrá una duración aproximada de una semana.

- Implementación de Automatización con integración continua: la integración continua ayuda al desarrollo ya que permite al equipo entregar código con más frecuencia. Las pruebas pueden ser parte de esto, y se pueden ejecutar de manera automática, para luego recibir resultados después de cada integración de código. Para realizar esta integración el responsable deberá ser el líder del equipo de desarrollo o aquel desarrollador que tenga más experiencia, y su duración puede ser aproximadamente de dos a tres semanas.

- Implementación del proceso de pruebas: un proceso de pruebas es necesario para la planeación del mantenimiento de la automatización, además de la creación de nuevas pruebas manuales en nuevos módulos de la aplicación, los cuales serán candidatos a ser automatizados. Por lo tanto, debe existir una persona encargada de llevar a cabo planes de pruebas tanto como para la automatización como para el testeo manual de la aplicación, así como describir los roles que tendrá el equipo en el proceso de pruebas. La persona responsable de este proceso será el analista de calidad y el *project manager* o *product owner*, y tomará un tiempo de dos semanas.

- Capacitación de la automatización: todo el personal perteneciente al equipo técnico y de desarrollo debe ser capacitado de forma que se explique qué es, para qué se utilizará la automatización y como se debe ejecutar. Esta capacitación será dirigida principalmente por el analista de calidad de *software*, o el desarrollador que posea más experiencia en automatización y tendrá una duración aproximada de una semana. Estos entrenamientos deben ser programadas previamente por el project manager

- Capacitación de mantenimiento de automatización: los colaboradores que se encargarán del mantenimiento del código de la automatización, sean desarrolladores o *testers*, deben tomar una capacitación. Este entrenamiento estará dirigido por el desarrollador o un profesional de aseguramiento de calidad de software que tenga experiencia en automatización de pruebas. La duración de esta preparación tomará dos semanas.

- Mantenimiento continuo del código: el mantenimiento del código depende de los cambios que se realizan en la aplicación, si hay una nueva inserción de código en la aplicación, o eliminación de datos, se debe analizar y modificar las pruebas para que estas realicen las validaciones correctas. Esto deberá ser analizado por el desarrollador a cargo de la inserción del nuevo código, o bien del personal de *quality assurance* y tendría una duración aproximadamente y mínima de dos horas.

- Adición de *test cases*: la integración de nuevos casos de prueba dependerá de los cambios que tenga el sistema, o bien pueden añadirse nuevos escenarios que se consideran de alta prioridad, o de alto valor para el negocio. Las personas encargadas de crear escenarios y tomar la decisión de incluirlos como candidatos para automatizar serán el *project manager* y el analista de calidad de *software*, y tendrá una duración de aproximadamente dos días.

- Automatización en paralelo: agregar una automatización en forma paralela puede reducir más tiempo y brinda la posibilidad de ejecutar simultáneamente los casos de prueba en múltiples navegadores y en diferentes sistemas operativos. Los responsables para llevar esta forma de automatización son los desarrolladores, y tardarían tres semanas en realizar esta recomendación.

CAPÍTULO VI. PROPUESTA

Análisis

A continuación, se presenta el análisis del prototipo desarrollado.

Creación del documento de plan de pruebas

Como parte del proyecto se realiza un documento llamado Plan de Pruebas o Test Plan, en el que vendrá documentado: la selección de tecnología, el alcance de la automatización, los casos de pruebas con sus pasos, el ambiente utilizado para las pruebas, riesgos, entre otros.

Este documento es necesario para proyectos de calidad como el presente y va dirigido tanto al personal técnico (desarrolladores, *testers*) como al de negocios (*project managers*, *product owners*, entre otros).

A continuación, se describen algunos de los pasos y secciones por realizar para la creación del plan de pruebas:

1. Definir el alcance: el alcance define las características, los requisitos funcionales o no funcionales del *software* que se probará. Asimismo, aquellas características que no se probarán y quedan fuera del alcance de la automatización.
2. Objetivos: el motivo y finalidad de la automatización que se está desarrollando.
3. Metodología por utilizar: la metodología por utilizar para crear el *test plan*, además de los entregables. En este caso se utilizará Agile, ya que es la que se ha estado implementando en la empresa.
4. Tipos de pruebas: se deben describir todos los tipos de pruebas que se llevarán a cabo, por ejemplo: Regresión, Seguridad, Funcionales, entre otros.
5. Entregables: se deberán incluir los *test cases* y escenarios de la aplicación que se automatizarán. Deben describirse de forma detallada y con sus respectivos pasos para el entendimiento tanto del equipo técnico como el de negocio.
6. Recursos y ambientes: la aplicación eDoc cuenta con varios ambientes: Producción, Desarrollo y de QA. Se deberá describir el ambiente donde estará ejecutándose la automatización, así como credenciales, URL, XML, entre otros.

7. Herramientas de testeo: deberán ser descritas las herramientas para crear la automatización, tales como Selenium, C#, Nunit, Git, Extent y todas aquellas tecnologías y librerías por utilizar durante el desarrollo.

Diseño y desarrollo de la automatización

Como parte del proyecto, se creará y diseñará el *framework* de automatización. Se automatizan con la herramienta y tecnología seleccionada, todos los casos de prueba que fueron documentados en el *test plan.* Esta es la fase más amplia y la que lleva más tiempo, debido a que se lleva a cabo un desarrollo o programación del *framework* de automatización.

Para empezar con el desarrollo, es importante recalcar que se utilizarán los siguientes modelos y prácticas en el código: Orientación Programada a Objetos, Behavior Driven Development, Modular Based Test Framework y Project Object Model.

Partiendo de los *test cases* del test plan mencionado en el punto anterior, se realiza la codificación de estos empezando por describir los pasos de cada uno utilizando el Behavior Driven Development. Usando esta práctica, se pueden mencionar los pasos de los *test cases* en un idioma natural que puede expresar el comportamiento y pasos de las pruebas. Esto ayuda y mejora la comunicación entre equipos técnicos como desarrolladores y no técnicos como *project managers*, *stakeholders,* entre otros.

Las pruebas serán visualizas desde la herramienta Visual Studio, y se ejecutarán desde el IDE. Ya sea por medio de *playlist* o seleccionando los *test cases* de forma manual. Una vez ejecutadas las pruebas, se creará un reporte con los resultados de la ejecución, hayan sido varias pruebas o una única prueba.

Hardware requerido

El *hardware* requerido para la ejecución y mantenimiento de la automatización consiste en una computadora de escritorio o laptop con acceso a internet o una máquina virtual con acceso a internet. Requisitos específicos que se utilizaron para el desarrollo del prototipo:

- Procesador: AMD Ryzen 5 3600.
- Memoria RAM: 16.0 GB.
- Disco duro 2 Terabytes.

Requisitos mínimos para ejecutar y realizar mantenimiento del prototipo:

- Procesador superior a i3 para el caso de Intel o superior a un Athlon para el caso de AMD.
- Memoria RAM superior a 8 GB.
- Disco duro superior a 500 GB.

Requisitos mínimos para ejecutar y realizar mantenimiento del prototipo en una máquina virtual:

- Memoria RAM superior a 8 GB.
- Almacenamiento de 100 GB.

Telecomunicaciones

El prototipo de automatización depende totalmente de conexión a internet, ya que el sistema que está bajo las pruebas es de tipo web. Se requiere el acceso al internet para que las pruebas puedan ejecutarse en el ambiente de pruebas de la aplicación, el cual se encuentra en la nube pública. Se pueden utilizar las siguientes conexiones comunes, no obstante, es posible utilizar otro tipo de conexiones:

- Redes inalámbricas: conexiones de Wifi, en frecuencias de 2,5 o 5,8 GHz con un ancho de banda necesario de 3 a 4 Mbps.
- Conexión por cable: fibra óptica, coaxial, con una velocidad de 80 Mbps como mínimo.

Personal requerido para uso y mantenimiento de la automatización.

El personal que utilizará y dará mantenimiento a la automatización deben ser profesionales de informática, ya sean colaboradores que se dediquen al desarrollo de *software* o que se dediquen al área de pruebas y calidad del *software*. Estos deberán tener conocimientos avanzados de programación con el lenguaje C#, un conocimiento al menos básico con automatización de pruebas o con la herramienta Selenium y conocimientos básicos de inglés. No obstante, la automatización posee el lenguaje Gherkin, la cual muestra los pasos de las pruebas en lenguaje natural (inglés), para que, de esta forma, profesionales que no tienen conocimientos de programación, pero que desean ejecutar las pruebas, puedan hacerlo y entender los pasos que realizan las pruebas.

Para ejecutar la automatización, no se requiere algún conocimiento avanzado en programación, sin embargo, para el mantenimiento de las pruebas, es requerido tener conocimientos técnicos de programación.

La cantidad de personas que pueden realizar mantenimiento, ya sea para agregar o modificar pruebas existentes en la automatización, no es limitada, pero puede variar dependiendo del flujo de trabajo que tenga la empresa y los cambios que se realicen en el sistema donde se ejecutan las pruebas. Por lo general, una automatización suele llevar un mantenimiento extensivo, ya que, como cualquier otro sistema que tenga código, necesita mantenimiento recurrente.

La capacitación o entrenamientos requeridos para entender, ejecutar y mantener la automatización se listan seguidamente:

- Programación en C#: Programación orientada a objetos.
- Selenium.
- Behavior Driven Development: Specflow.
- Nunit.
- Microsoft Visual Studio.
- Git

Casos de uso

Tabla 6
Casos de uso

Prototipo: Propuesta para la automatización de pruebas de la aplicación eDoc en la empresa Gurusoft, ubicada en La Sabana.	
Número caso de uso:	CU01: Ejecución de pruebas automáticas.
Fecha elaboración:	15/08/21
Descripción caso de uso:	Flujo para ejecutar las pruebas de automatización.
Autor caso de uso:	Génesis Cubillo Salazar
Actores relacionados:	Equipo de desarrollo de Gurusoft. Project Manager del proyecto de eDoc.
Precondiciones:	Credenciales correctas para entrar al sistema eDoc en el código del *test script.* Configuraciones generales para ejecutar pruebas: Ambiente en donde se ejecutarán (URL de Producción, desarrollo o ambiente de pruebas). Navegador que se utilizará para su ejecución (Chrome o Edge). Playlist creada previamente por la autora del proyecto.
Flujo básico del caso de uso	
1. El actor verifica que el url y el navegador deseado se encuentren en el archivo de configuración.	

2. El actor ejecuta las pruebas. 3. La automatización abre el navegador y ejecuta las pruebas. 4. La automatización realiza un reporte con las pruebas que han sido exitosas y las que han fallado.
Subflujos
Flujo Alternativo No. 1 — Ante posibles errores en la automatización, se requiere la intervención del usuario actor para la revisión de configuraciones, tales como archivos de configuración o actualizaciones necesarias del *driver.*
Requerimientos especiales
Requerimientos de compatibilidad: la ejecución de pruebas se debe realizar en el entorno Windows 10.
Postcondiciones
Inmediatamente después de haber finalizado el *playlist* de pruebas, se guardan los resultados de las pruebas fallidas y las pruebas exitosas.

Tabla 7
Casos de uso

Prototipo: Propuesta para la automatización de pruebas de la aplicación eDoc en la empresa Gurusoft, ubicada en La Sabana.	
Número caso de uso:	CU02: Ejecución de pruebas desde la playlist.
Fecha elaboración:	15/08/21
Descripción caso de uso:	Flujo para ejecutar las pruebas de automatización desde un playlist.
Autor caso de uso:	Génesis Cubillo Salazar
Actores relacionados:	Equipo de desarrollo de Gurusoft. Project Manager del proyecto de eDoc.
Precondiciones:	Credenciales correctas para entrar al sistema eDoc en el código del *test script*. Configuraciones generales para ejecutar pruebas: Ambiente en donde se ejecutarán (URL de Producción, desarrollo o ambiente de pruebas). Navegador que se utilizara para su ejecución (Chrome o Edge). Playlist creada previamente por la autora del proyecto.
Flujo básico del caso de uso	
1. El actor verifica que el url y el navegador deseado se encuentren en el archivo de configuración. 2. El actor importa el archivo de la *playlist* (lista de pruebas) para ejecutar las pruebas. 3. El actor ejecuta las pruebas de la *playlist.*	

<table>
<tr><td colspan="2">4. La automatización abre el navegador y ejecuta las pruebas seleccionadas del playlist.
5. La automatización realiza un reporte con las pruebas que han sido exitosas y las que han fallado.</td></tr>
<tr><td colspan="2">Subflujos</td></tr>
<tr><td colspan="2">Corresponde a las diferentes opciones (alternativas funcionales) que un actor tiene al iniciar con el
flujo básico.</td></tr>
<tr><td>Crear playlist</td><td>1. El autor selecciona los test cases, para crear un nuevo playlist.
2. El actor importa el archivo de la playlist (lista de pruebas) para ejecutar las pruebas.
3. El actor ejecuta las pruebas de la playlist.
4. La automatización abre el navegador y ejecuta las pruebas seleccionadas del playlist.
5. La automatización realiza un reporte con las pruebas que han sido exitosas y las que han fallado.</td></tr>
<tr><td colspan="2">Flujos alternos</td></tr>
<tr><td colspan="2">Corresponde a lo que debe realizar el sistema ante posibles errores</td></tr>
<tr><td>Flujo Alternativo No. 1</td><td>Ante posibles errores en la automatización, se requiere la intervención del usuario actor para la revisión de configuraciones tales como: archivos de configuración o actualizaciones necesarias del driver.</td></tr>
<tr><td colspan="2">Requerimientos especiales</td></tr>
<tr><td colspan="2">Requerimientos de compatibilidad: la ejecución de pruebas se debe realizar en el entorno Windows 10.</td></tr>
<tr><td colspan="2">Postcondiciones</td></tr>
<tr><td colspan="2">Inmediatamente después de haber finalizado el playlist de pruebas, se guardan los resultados de las pruebas fallidas y las pruebas exitosas.</td></tr>
</table>

Tabla 8
Casos de uso

Prototipo: Propuesta para la automatización de pruebas de la aplicación eDoc en la empresa Gurusoft, ubicada en La Sabana.	
Número caso de uso:	CU03: Ejecución de pruebas por demanda.
Fecha elaboración:	15/08/21
Descripción caso de uso:	Flujo para ejecutar las pruebas de automatización por demanda del actor.
Autor caso de uso:	Génesis Cubillo Salazar
Actores relacionados:	Equipo de desarrollo de Gurusoft. Project Manager del proyecto de eDoc.
Precondiciones:	Credenciales correctas para entrar al sistema eDoc en el código del *test script.* Configuraciones generales para ejecutar pruebas: Ambiente en donde se ejecutarán (URL de Producción, desarrollo o ambiente de pruebas). Navegador que se utilizará para su ejecución (Chrome o Edge). *Playlist* creada previamente por la autora del proyecto.
Flujo básico del caso de uso	
1. El actor verifica que el url y el navegador deseado se encuentren en el archivo de configuración. 2. El actor selecciona las pruebas que desea ejecutar. 3. El actor ejecuta las pruebas. 4. La automatización abre el navegador y ejecuta las pruebas seleccionadas por el actor. 5. La automatización realiza un reporte con las pruebas que han sido exitosas y las que han fallado.	
Flujos alternos	
Corresponde a lo que debe realizar el sistema ante posibles errores	
Flujo Alternativo No. 1	Ante posibles errores en la automatización, se requiere la intervención del usuario actor para la revisión de configuraciones, tales como archivos de configuración o actualizaciones necesarias del controlador.
Requerimientos especiales	
Requerimientos de compatibilidad: la ejecución de pruebas se debe realizar en el entorno Windows 10.	
Postcondiciones	
Inmediatamente después de haber finalizado el *playlist* de pruebas, se guardan los resultados de las pruebas fallidas y las pruebas exitosas.	

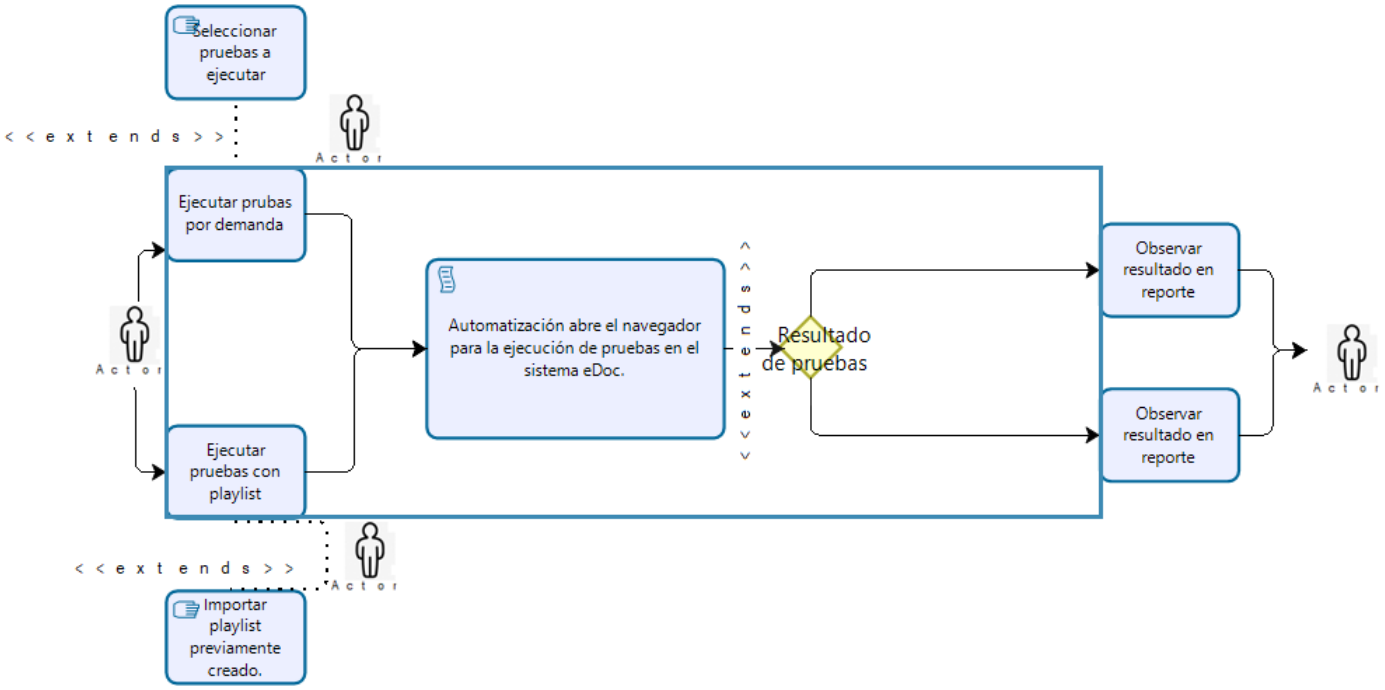
Tabla 9
Casos de uso

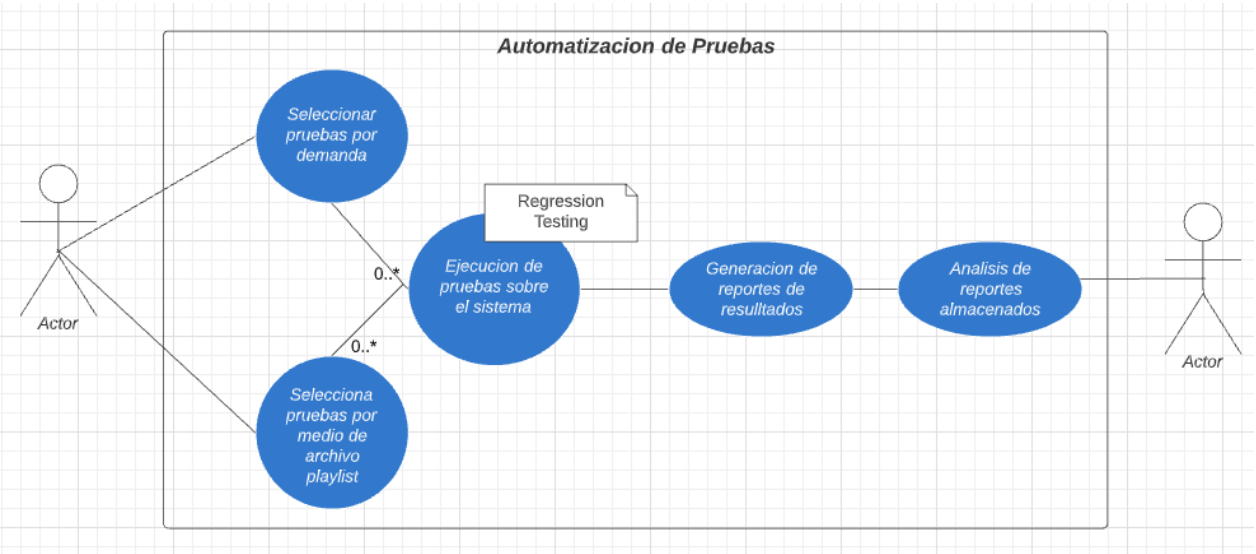
<table>
<tr><td colspan="2">Prototipo: Propuesta para la automatización de pruebas de la aplicación eDoc en la empresa Gurusoft, ubicada en La Sabana.</td></tr>
<tr><td>Número caso de uso:</td><td>CU04: Generación de reportes.</td></tr>
<tr><td>Fecha elaboración:</td><td>15/08/21</td></tr>
<tr><td>Descripción caso de uso:</td><td>Generación de reportes de pruebas de automatización luego de ser ejecutadas.</td></tr>
<tr><td>Autor caso de uso:</td><td>Génesis Cubillo Salazar</td></tr>
<tr><td>Actores relacionados:</td><td>Equipo de desarrollo de Gurusoft.
Project Manager del proyecto de eDoc.</td></tr>
<tr><td>Precondiciones:</td><td>Credenciales correctas para entrar al sistema eDoc en el código del test script.
Configuraciones generales para ejecutar pruebas: Ambiente en donde se ejecutarán (URL de Producción, desarrollo o ambiente de pruebas).
Navegador que se utilizará para su ejecución (Chrome o Edge).
Playlist o selección de pruebas para ejecutarlas.</td></tr>
<tr><td colspan="2">Flujo básico del caso de uso</td></tr>
<tr><td colspan="2">1. El actor verifica que el url y el navegador deseado se encuentren en el archivo de configuración.
2. El actor selecciona las pruebas que desea ejecutar.
3. El actor ejecuta las pruebas.
4. La automatización abre el navegador y ejecuta las pruebas seleccionadas por el actor.
5. La automatización realiza un reporte con las pruebas que han sido exitosas y las que han fallado.</td></tr>
<tr><td colspan="2">Flujos alternos</td></tr>
<tr><td colspan="2">Corresponde a lo que debe realizar el sistema ante posibles errores</td></tr>
<tr><td>Flujo Alternativo No. 1</td><td>Ante posibles errores en la automatización, se requiere la intervención del usuario actor para la revisión de configuraciones, tales como archivos de configuración o actualizaciones necesarias del controlador.</td></tr>
<tr><td colspan="2">Requerimientos especiales</td></tr>
<tr><td colspan="2">Requerimientos de compatibilidad: la ejecución de pruebas se debe realizar en el entorno Windows 10.</td></tr>
<tr><td colspan="2">Postcondiciones</td></tr>
<tr><td colspan="2">Inmediatamente después de haber finalizado la ejecución de pruebas, se guardan los reportes con resultados de prueba o pruebas ejecutadas, el autor analiza los resultados.</td></tr>
</table>

Diagramas de casos de uso

La figura 4 muestra el diagrama de caso de uso general.

Figura 5
Diagrama de caso de uso general



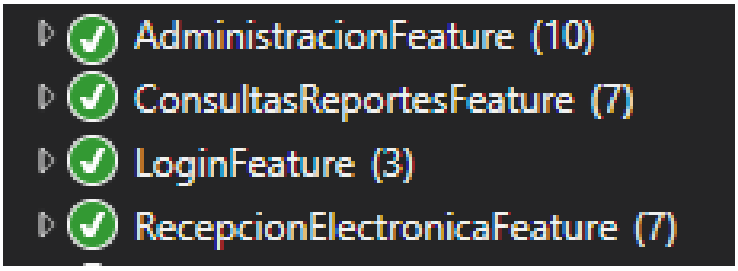


Diseño

Arquitectura del prototipo

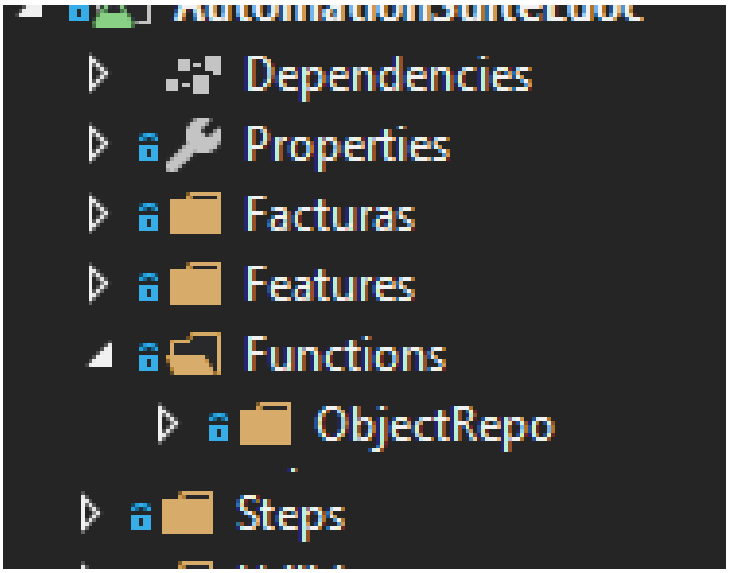
Las pruebas en la automatización se mostrarán según el módulo al que pertenecen en la aplicación eDoc, en el Explorador de Pruebas del Visual Studio, de esta manera, las pruebas pueden ser encontradas más fácilmente y lucen de forma ordenada. La presentación de las pruebas se muestra en la figura 6:

Figura 6
Módulos



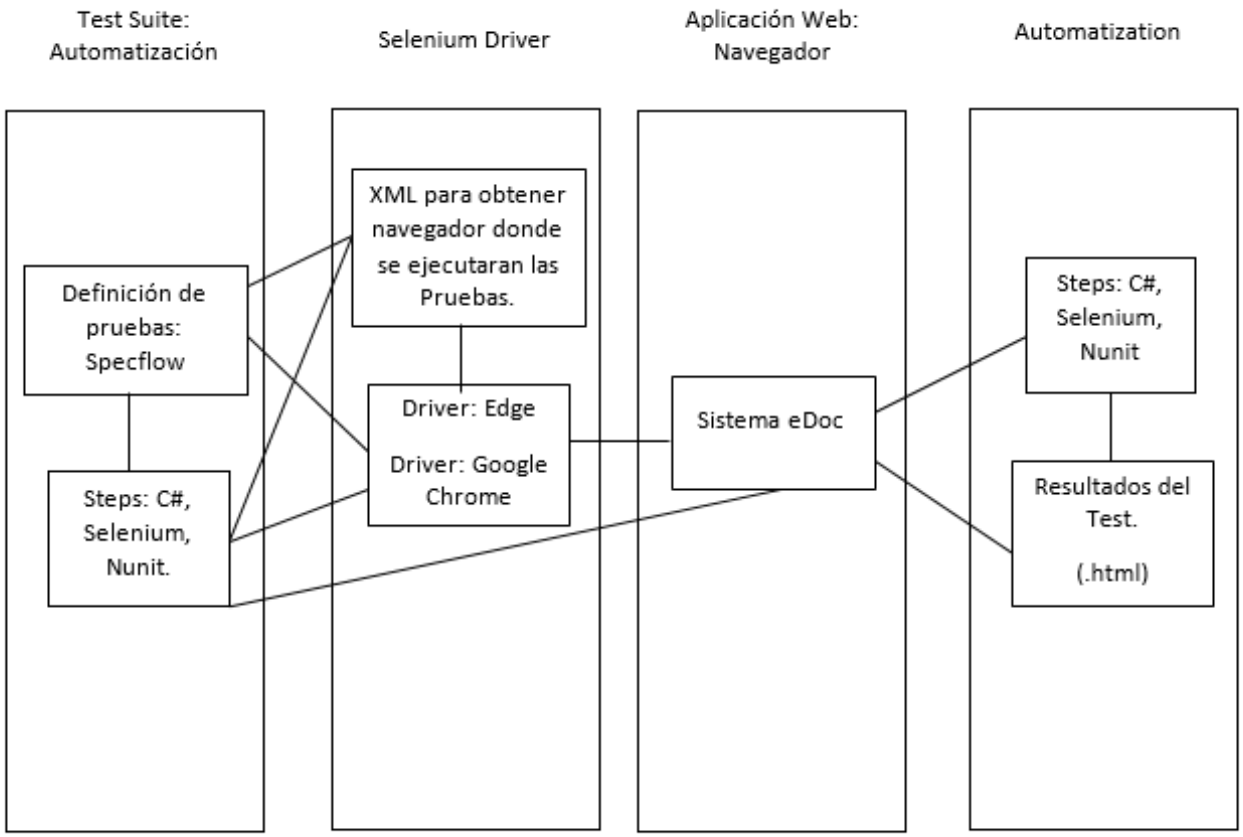
En la arquitectura del código, se utiliza el patrón de diseño Page Object Model, el cual consiste en apartar los elementos de la página, de los métodos e instrucciones de cada prueba automatizada. Así mismo, se utiliza el Modular Based *framework*, el cual afirma que los *test cases* deben ser divididos en diferentes módulos en el código. Así que, para este caso, se han creado carpetas para el ordenamiento del código, donde las carpetas denominadas Features, Functions, ObjectRepo y Steps representan una capa o módulo siguiendo el *modular based framework*. Así mismo, el ObjectRepo representa el patrón de diseño de Page Object Model separando así los objetos de las instrucciones de cada *script.* En la figura 6, se muestra la vista general interna del prototipo, donde cada carpeta representa una capa. Estas capas se detallarán en la siguiente sección.

Figura 7
Vista general de estructura del prototipo



En la siguiente figura 8, se muestra la arquitectura de la automatización en relación con el sistema eDoc, en el cual se ejecutaran las pruebas:

Figura 8
Arquitectura de automatización

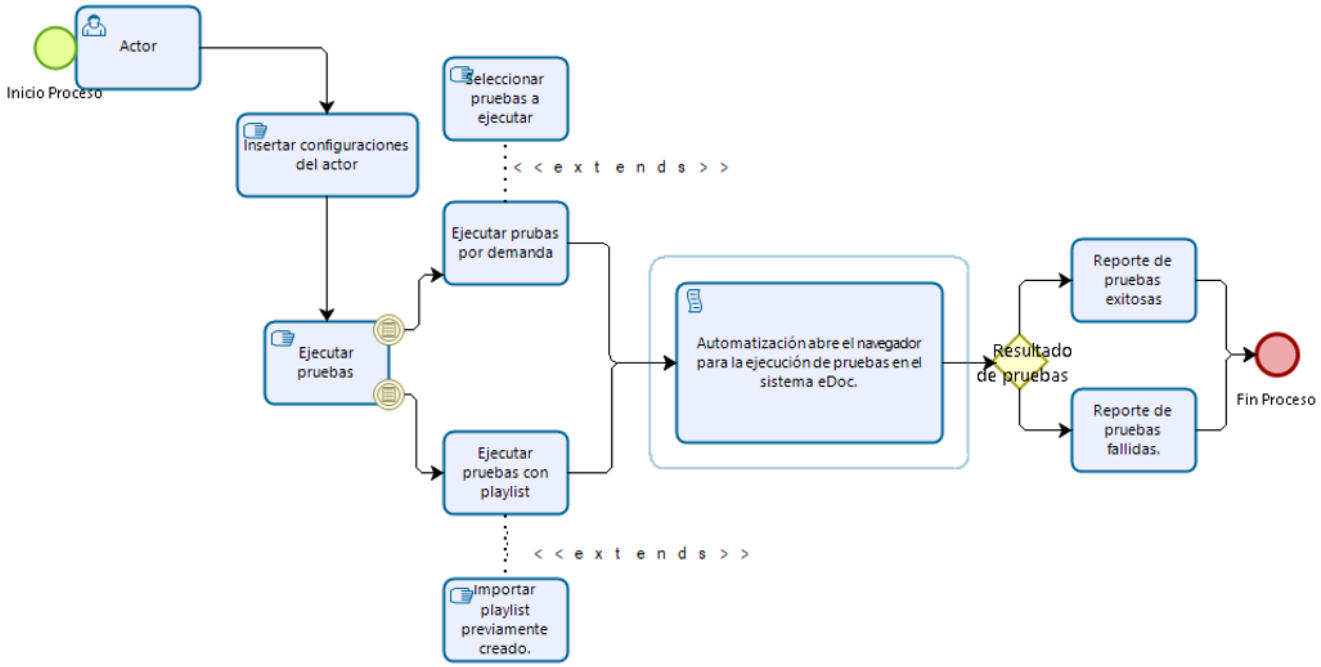


Diseño de procesos

A continuación se presenta un diagrama de los procesos de la automatización. En donde se muestran las diferentes formas de uso y ejecución del prototipo de pruebas. En donde, como se ha mencionado anteriormente en los casos de uso, el actor puede ejecutar las pruebas por demanda, o desde un *playlist* previamente creado dependiendo de la necesidad del actor al momento de utilizarla.

A continuación, la figura 8 presenta un diagrama de los procesos de la automatización.

Figura 9
Diagrama de flujo



En las siguientes figuras: 9, 10 y 11 se muestras los diagramas con cada uno de los principales procesos de la propuesta, es decir, los principales procesos que serán testeados en la aplicación:

Figura 10
Diagrama de flujo para mantenimiento de datos en el sistema

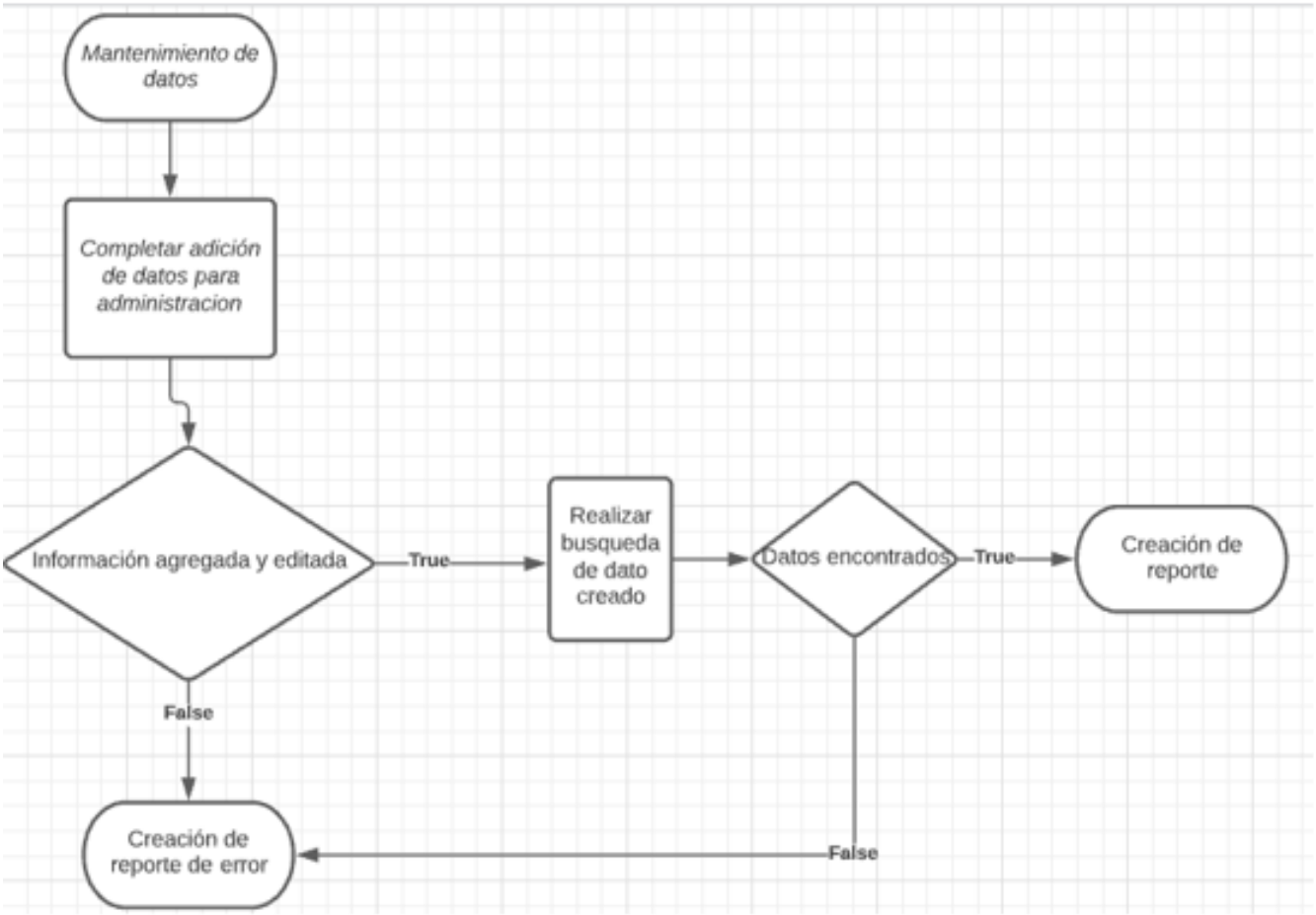


Figura 11
Diagrama de flujo para búsqueda de datos en el sistema

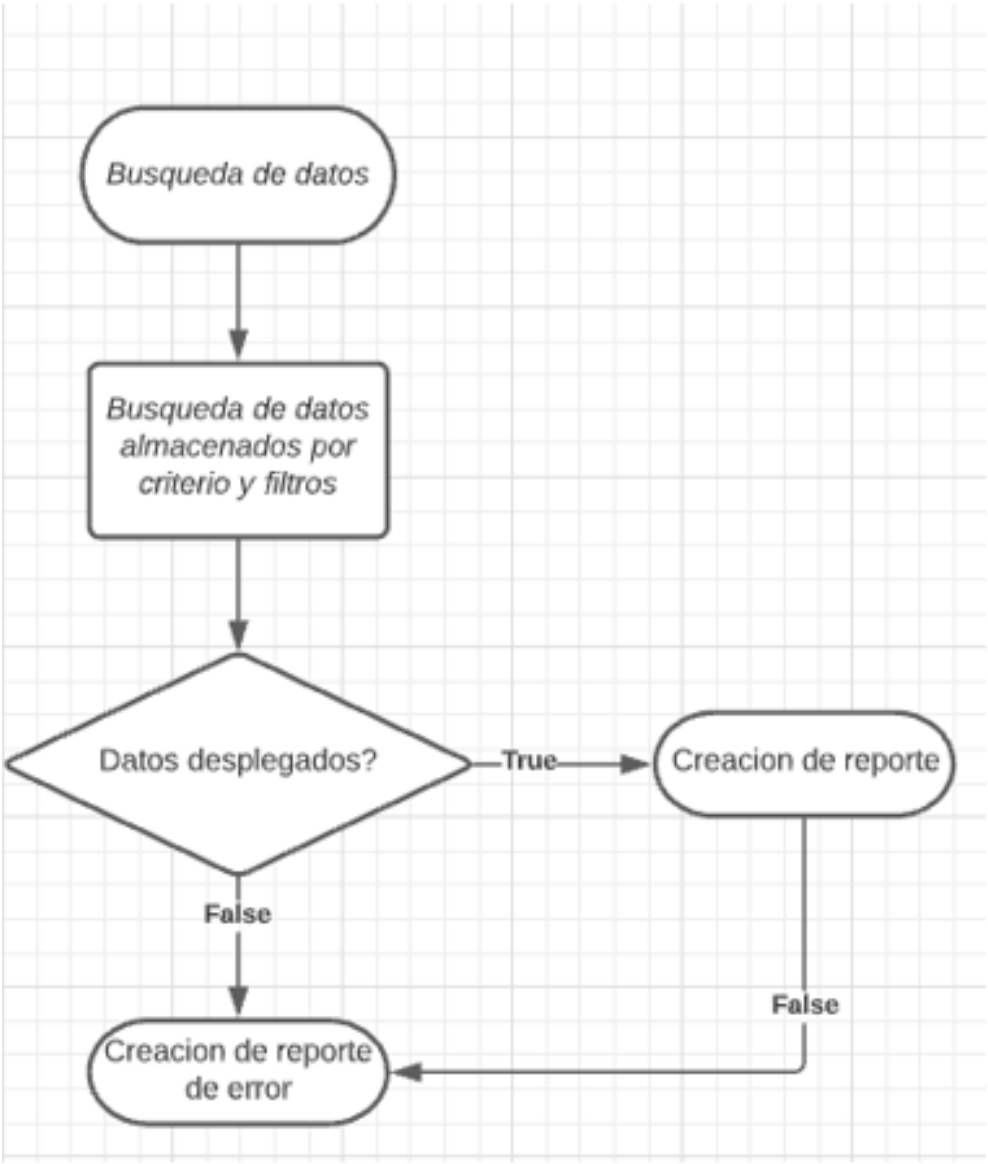
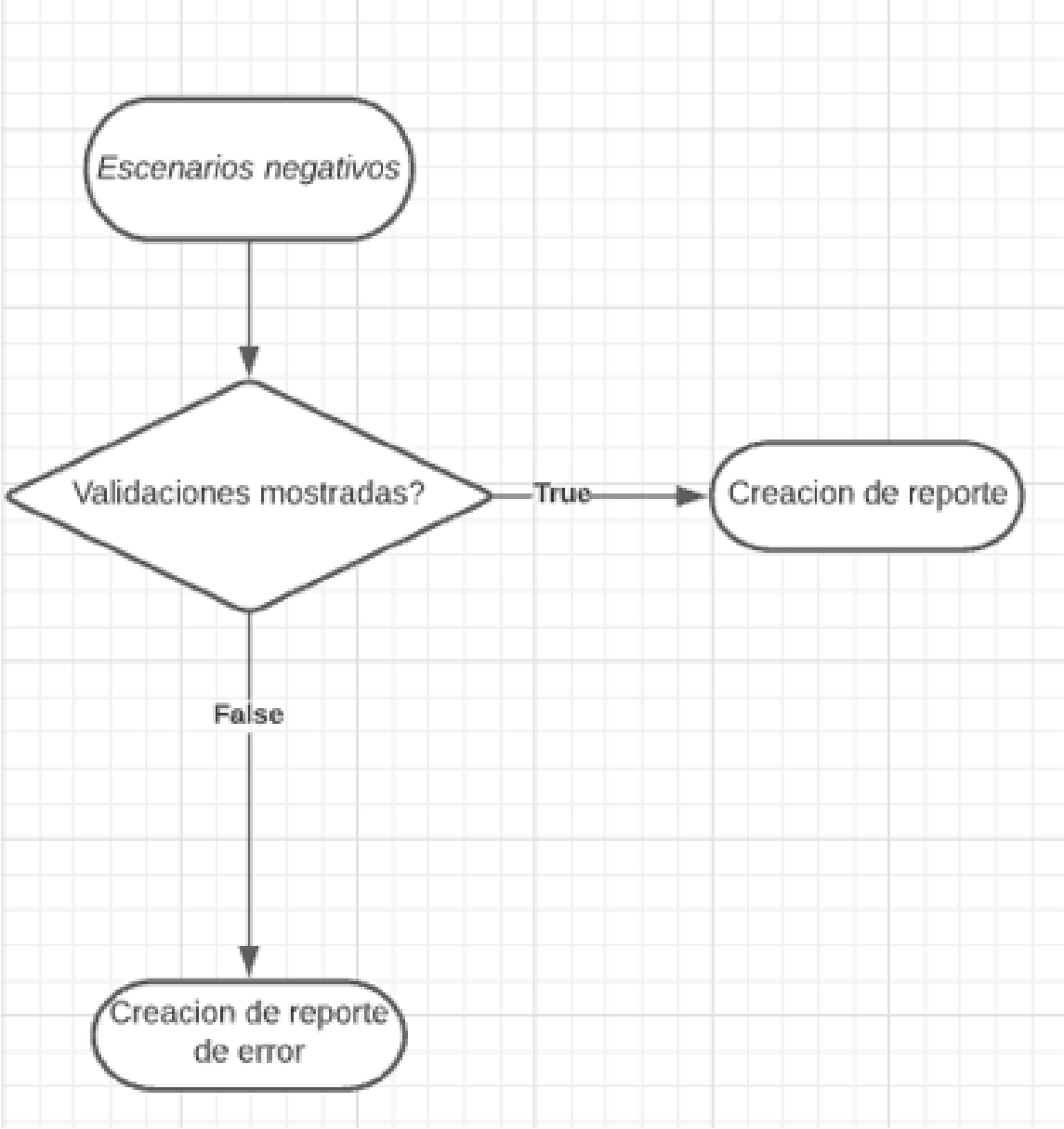


Figura 12
Diagrama de flujo para escenarios negativos y validaciones del sistema



Diseño de salidas

Cada vez que se ejecute uno o todos los casos de prueba, se creará un reporte de la prueba automática. De esta manera, se podrá conocer los resultados cada vez que se ejecute la automatización y en caso de que existan fallos durante la ejecución (ya sean por algún cambio en la aplicación o defectos), el reporte será útil para conocer detalladamente el porqué. El tipo de contenido de los reportes que se mostrarán una vez ejecutada la automatización, son los siguientes:

- Cantidad de pruebas ejecutadas exitosamente.
- Cantidad de pruebas que fallaron.
- Descripción de errores cuando las pruebas fallaron: errores y muestra del código fallido.
- Duración en total de la ejecución de las pruebas.
- *Screenshot* de la aplicación cuando se generaron errores en la ejecución de la prueba.

Figura 13
Reporte con escenario

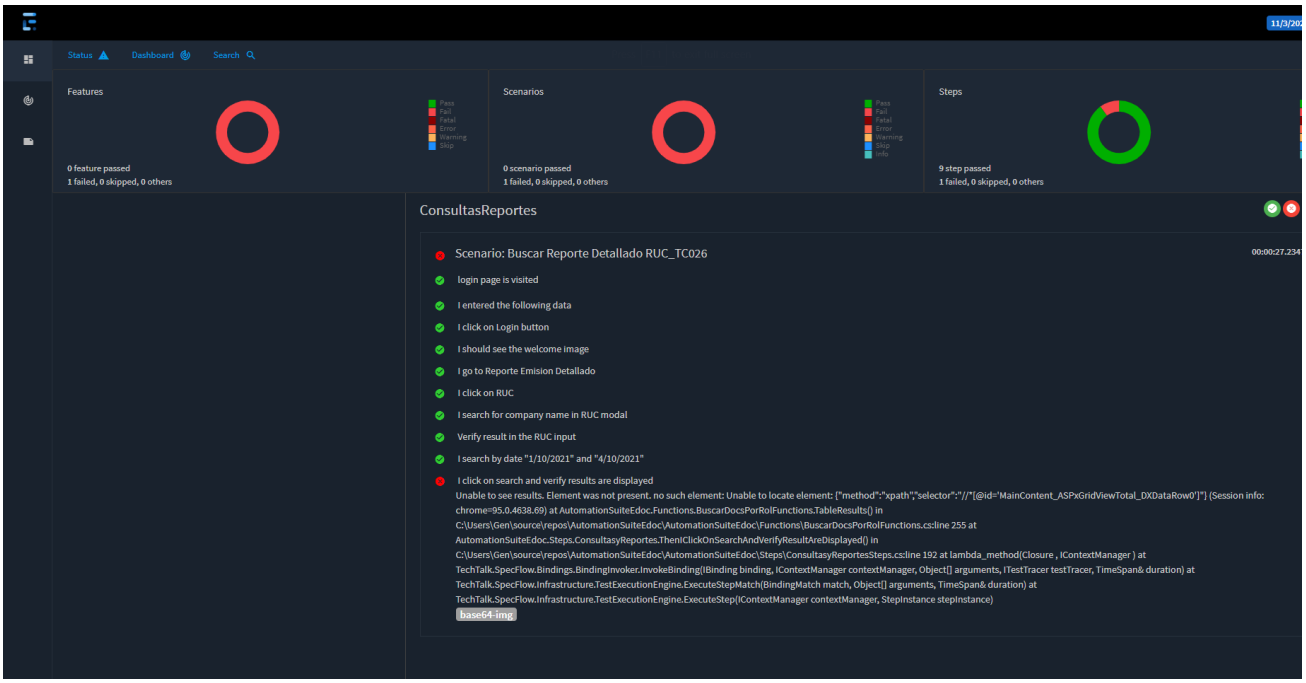


Figura 14
Reportes Dashboard

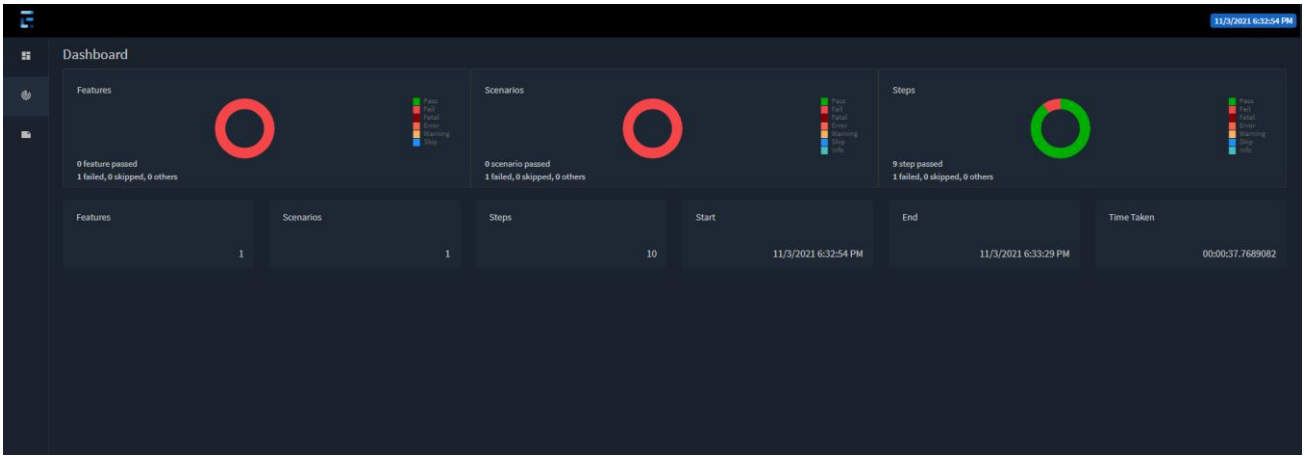
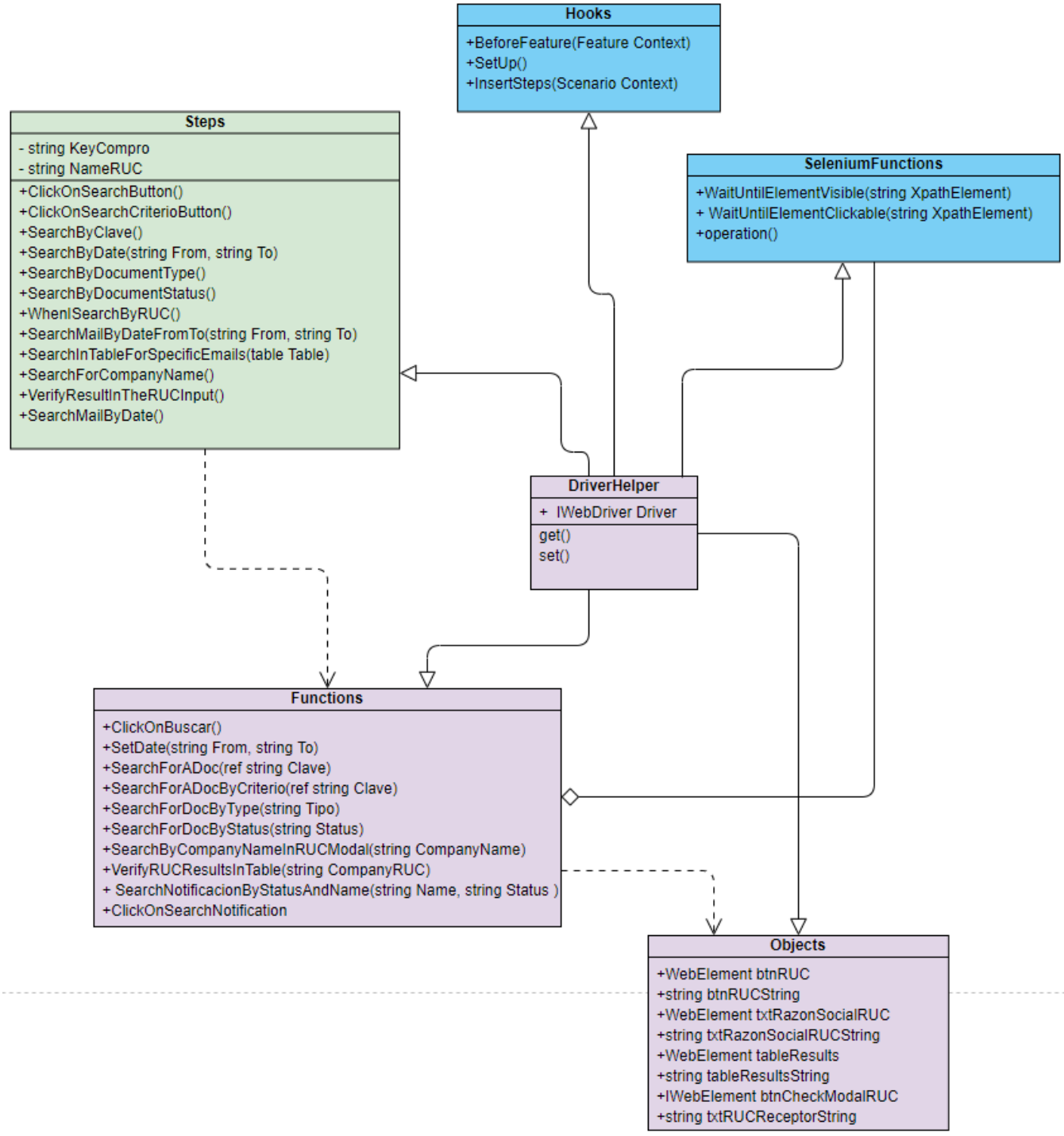


Diagrama de clases

A continuación, en la figura 10 se presenta un diagrama de clases general de las clases, métodos y atributos que contiene un caso de prueba. Cada *test case* se conforma de diferentes métodos, y atributos, por lo que la cantidad de funciones dependerá de la cantidad de pruebas automatizadas. El siguiente diagrama representa las clases, de solamente un caso de prueba.

Figura 15
Diagrama de clases



Programación

Dentro de cada módulo o carpeta, existen clases y dentro de ellas, los métodos, que realizan la interacción de las pruebas con el sistema. A continuación, se describen las clases que tiene la automatización:

Objetos (*Object Repo*): para cada elemento en la aplicación (sean botones, cajas de texto, *dropdowns*, etc.), se debe crear un objeto donde la información del elemento se guardará y, de esta manera, las demás clases o funciones puedan llamarlo e interactuar con ellos. Para guardar la información del objeto, se pueden usar las siguientes características: Clase, Nombre, XPath, Css Selector, entre otros. Todos estos elementos deberán ir declarados en folders separados para hacer más sencilla la interacción con estos.

Funciones (*Functions*): para cada *test case*, existirán diferentes pasos por ejecutar, en este módulo es donde se codificarán todas aquellas funciones o clases, que sean una instrucción en el *script*. Este módulo va directamente relacionado con los objetos, ya que aquí es donde se empiezan a codificar las instrucciones para interactuar con ellos.

Pasos (*Steps*): esta clase se comunica con la anterior. Su objetivo es llamar todas las funciones de la clase anterior y verificar que estas han sido ejecutadas de forma exitosa; en caso de que alguna no haya sido ejecutada exitosamente, significa que hay algún error, por lo que esta clase debe encargarse de comunicarle al reporte cuál fue el método que falló.

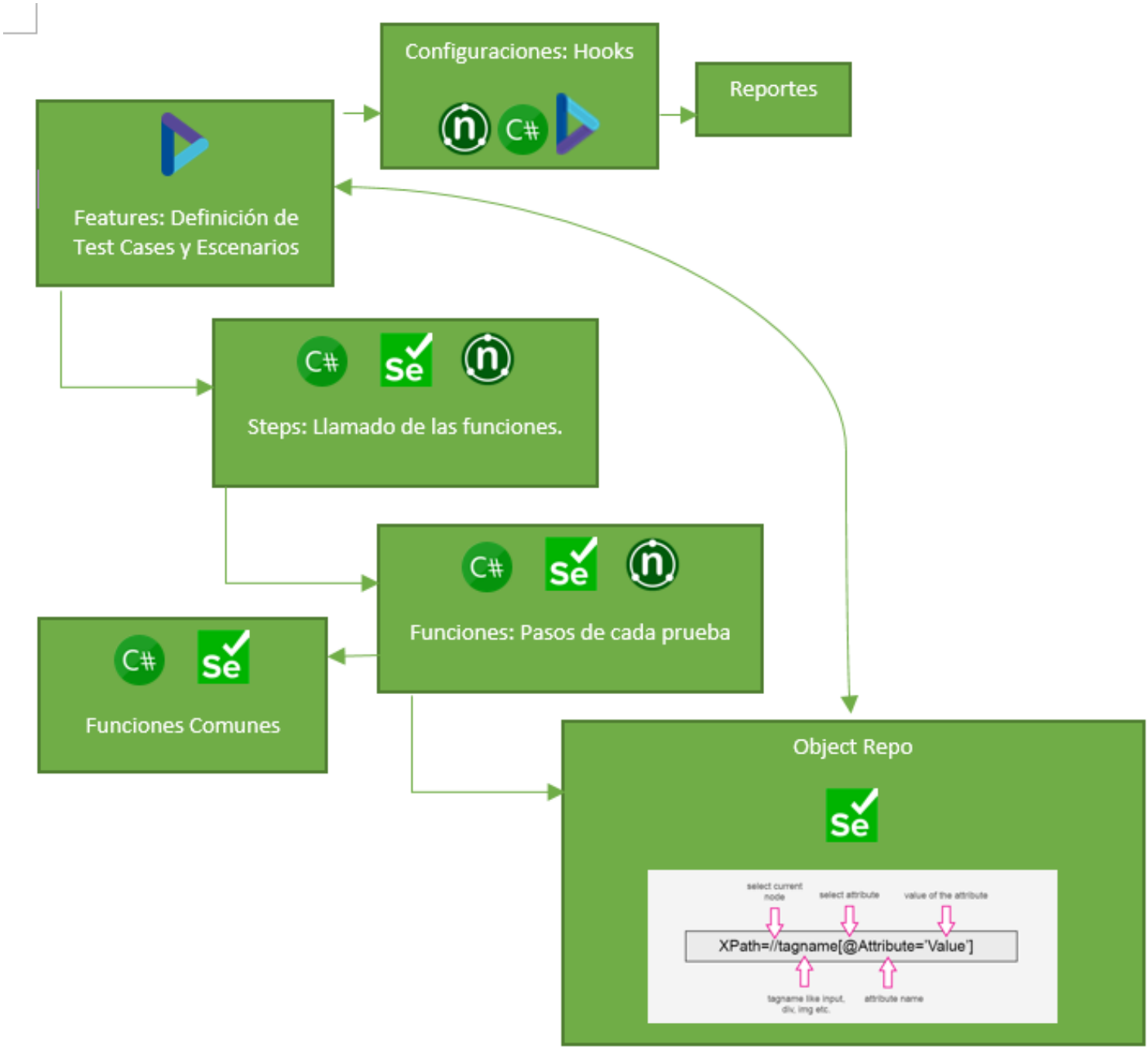
Características (*Features*): esta es la clase o módulo de más alto nivel dentro del código. En la cual se iniciarán los *test cases* y *scripts.* Se hace el llamado de la clase anterior (Steps) para que así tenga un efecto en cascada como lo sugiere la metodología de Modular Based. Se agrega el nombre del *test case*, sus pasos en leguaje natural llamado *Gherkin* y la categoría (*Regression*, ***Security***, entre otros). En este módulo, se escriben las pruebas de manera que se pueden visualizar en el IDE como un *Test Explorer.* Esta clase es un tipo de archivo llamada *feature*, el cual deriva de la librería de Specflow utilizada en la automatización.

Funciones comunes: se necesitará crear una clase para funciones que son comunes o útiles para la interacción con los objetos. Pueden ser funciones para interactuar con algún tipo de *dropdown* en especial o incluso alguna lectura de un archivo. Este tipo de funciones se codifican en un folder aparte de donde se encuentran los *scripts* de cada *test case*, para que, de esta forma, todas las clases puedan acceder a estos métodos.

***Hooks*:** los *hooks* son una clase utilizada para realizar una lógica en la automatización en momentos específicos, como alguna configuración necesaria antes de ejecutar un escenario o configuración de reportes.

En la siguiente figura 8, se muestra la interacción de cómo cada carpeta o folder es dependiente, dentro de estas carpetas se encuentran todas las clases y funciones para cada caso de prueba.

Figura 16
Clases



Entradas y salidas

La mayoría de métodos relacionados con acciones o instrucciones en las pruebas que se encuentran en la clase llamada *Functions* (por ejemplo, un método para realizar un *click* en un objeto en la página web) contienen una entrada y una salida.

Dependiendo de la funcionalidad del método, éste recibirá un parámetro exclusivo para llevar a cabo su misión, ya sea algún dato en específico como fechas o nombres. Durante la ejecución del método, se tomará como entrada el parámetro definido para la función.

Esta forma de recibir entradas hará que el mantenimiento futuro sea más sencillo, y que el compartimiento de métodos sea posible, de esta manera, los métodos o réplicas de código innecesario se disminuye. A continuación, en la figura 12 se muestra un ejemplo de una entrada como parámetro en una función, el cual se encarga de recibir un parámetro de tipo "**s***tring*".

Figura 17
Función con parámetros

```
public bool VerifyCompanyRUCInInput (string CompanyRUC)
{
    bool flag = false;
    try
    {
        log.Info("Entering Method: VerifyCompanyRUCInInput ");

        SeleniumFunctions.WaitUntilElementVisible(ConsultasyReportesObjects.txtRUCReceptorString);
        string txtRUC = ConsultasyReportesObjects.txtRUCReceptor.GetAttribute("value");
        Assert.AreEqual(CompanyRUC, txtRUC);
```

Las entradas también pueden ser definidas en la capa o clase en donde se definen los casos de pruebas en lenguaje natural o *Gherkin*, que se denomina *Features*. En la siguiente figura 13, se muestra un ejemplo de un dato que será utilizado como parámetro, son fechas que una función utilizará como entrada.

Figura 18
Parámetros de fecha

```
45      Then I should see the welcome image
46      And I go to Consulta Por Permisos Rol
47      When I search by date "1/10/2021" and  "25/10/2021"
48      And I click on search button
```

Con respecto a las salidas, la mayoría de métodos tienen la posibilidad de retornar un resultado de tipo *booleano*, de esta forma cada función mostrará si su resultado durante la ejecución fue exitoso, el cual se mostraría como una variable con valor: *true*. De lo contrario, si el método no fue exitoso tendrá una salida como valor de tipo *false*. En caso de que un método tenga una salida de tipo *false*, este resultado vendrá acompañado de otra salida, que es el tipo de error que tuvo la función durante su ejecución. De esta forma se podrá saber cuál fue el error y por qué la instrucción que se ejecutó falló. En las figuras 14 y 15 se muestran los errores cuando una instrucción falla en la ejecución de una prueba.

Figura 19
Errores de métodos

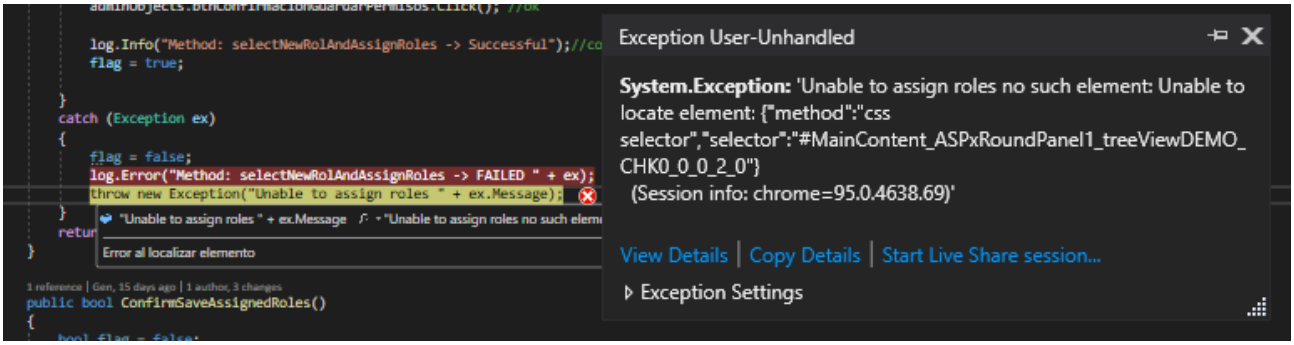
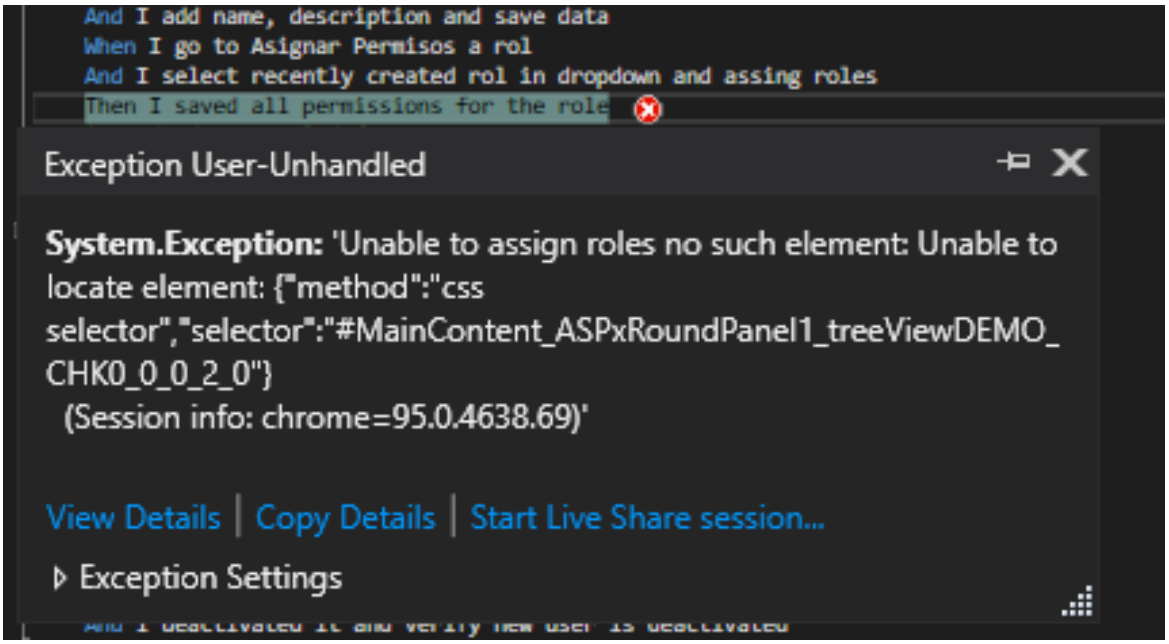


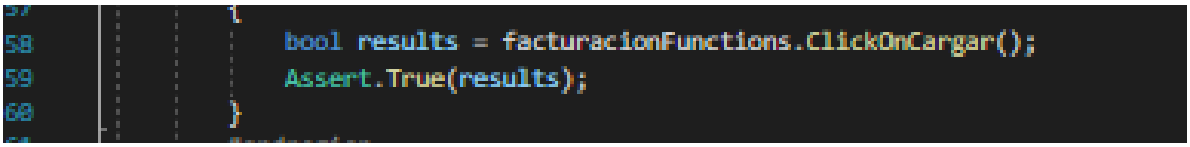
Figura 20
Errores en instrucción de escenario



Validaciones

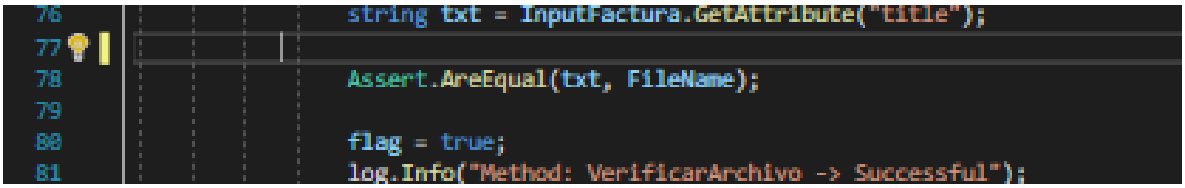
Las validaciones en este tipo de proyectos, son realmente importantes para confirmar la ejecución exitosa del caso de prueba, por lo tanto el prototipo utiliza el framework denominado Nunit, el cual provee clases llamadas *Asserts*. Estas clases se utilizan para para determinar si un conjunto de pruebas han sido exitosas o fallidas. Como se mencionó en la previa sección de entradas y salidas, la mayoría de las funciones retornan un resultado o bandera de tipo booleano, este resultado se toma para ser comparado con una clase *Assert*. Por lo tanto, la salida de cada método será siempre evaluado por un *Assert*.

Figura 21
Validación con Assert



Así mismo, las clases *Assserts*, participan en los métodos de los casos de prueba, validando las instrucciones que se encuentran dentro de las funciones. De esta forma se realizan validaciones inmediatas y sin crear código innecesario.

Figura 22
Evaluación de instrucción con un Assert



Pruebas

A pesar de que el presente proyecto está enfocado en realizar una automatización de pruebas, estas también deben ser testeadas y depuradas. Cada prueba debe tener un resultado correcto esperado, de manera que, para testear que las pruebas estén realizando las validaciones de forma óptima, se hacen fallar de forma intencional.

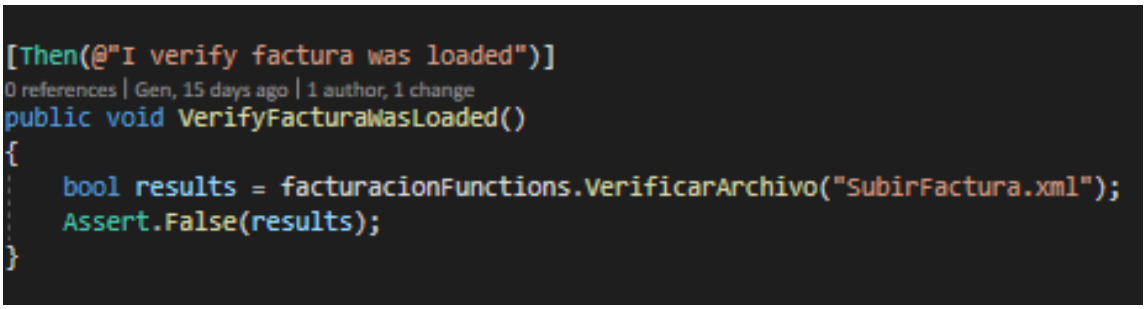
1. Si un método busca un elemento de la página, el código deberá cambiarse de forma que busque un elemento en la página que no se encuentra disponible.
2. Si un método busca un archivo, este se debe eliminar, para comprobar que la prueba automatizada no lo encontró.
3. Si la prueba del código confirmaba algún mensaje presente en la página, se debe cambiar este mensaje esperado, para comprobar que la prueba falle.
4. Si la prueba espera como resultado un *booleano* como *true*, se cambia el código para que verifique el *false*, lo cual debe fallar porque, originalmente, se esperaba el retorno de una variable *true*.

De esta forma, se confirma que las pruebas automáticas estén comprobando el sistema eDoc de forma correcta. Las figuras 18 y 19 muestran un ejemplo de la prueba en el código.

Figura 23
Código original

```
[Then(@"I verify factura was loaded")]
0 references | Gen, 15 days ago | 1 author, 1 change
public void VerifyFacturaWasLoaded()
{
    bool results = facturacionFunctions.VerificarArchivo("SubirFactura.xml");
    Assert.True(results);
}
```

Figura 24
Código modificado para prueba



```
[Then(@"I verify factura was loaded")]
0 references | Gen, 15 days ago | 1 author, 1 change
public void VerifyFacturaWasLoaded()
{
    bool results = facturacionFunctions.VerificarArchivo("SubirFactura.xml");
    Assert.False(results);
}
```

En las ilustraciones anteriores, se demuestra que en el código original, se hace una validación de que el primer método debería devolver un resultado como *true*, si esta validación se cambia a *false*, la prueba debería fallar, ya que la variable llamada *results*, con el método "VerificarArchivo" retorna un *true*, si esta es exitosa.

REFERENCIAS

Atlassian. (2021a). *What is Software Development?* Recuperado el 1 de noviembre de 2021 de https://www.atlassian.com/es/software-development

Atlassian. (2021b). *The different types of software testing.* Recuperado el 1 de noviembre de 2021 de https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Barrantes, R. (2013). *A la búsqueda del conocimiento científico.* EUNED.

Bernal, C. (2010). *Metodología de la investigación* (3a ed.). Pearson Education.

Boyde, J. (2014). *A Down-to-Earth Guide to SDLC Project Management*. CreateSpace.

Chopra, R. (2016). *Software Quality Assurance: A Self-Teaching Introduction.* New Delhi.

Cucumber. (2019). *Cucumber Installation*. Recuperado el 1 de noviembre de 2021 de https://cucumber.io/docs/installation/

Cucumber. (2019). *Cucumber Reference*. Recuperado el 1 de noviembre de 2021 de https://cucumber.io/docs/cucumber/api/

Evertt, G. D. y McLeod J. (2006). *Software Testing: Testing Across the Entire Software Development Life Cycle.* Wiley-IEEE Computer Society

Grande, E. y Abascal, E. (2014). *Fundamentos y técnicas de Investigación. Fundamentos y técnicas de Investigación.* ESIC.

GeeksforGeeks. (4 de octubre de 2021). *DOM (Document Object Model).* Recuperado el 1 de noviembre de 2021 de https://www.geeksforgeeks.org/dom-document-object-model/

Gómez, M. (2006). *Introducción a la Metodología de la Investigación Científica.* Editorial Brujas.

Hernández, R. (2017). *Fundamentos de Investigación.* Mcgraw-Hill.

International Business Machines Corporation. (2006). *Concepto: Prototipos.* Recuperado el 1 de noviembre de 2021 de https://cgrw01.cgr.go.cr/rup/RUP.es/SmallProjects/core.base_rup/guidances/concepts/prototypes_9D1E67A.html

International Business Machines Corporation. (s.f.). *What is software testing?* Recuperado el 1 de noviembre de 2021 de https://www.ibm.com/topics/software-testing

International Software Testing Qualifications Board. (2019). *Certified Tester: Foundation Level Syllabus*. https://astqb.org/assets/documents/CTFL-2018-Syllabus.pdf

Mahfuz, A. S. (2016). *Software Quality Assurance: Integrating Testing, Security, and Audit.* Auerbach Publications

QA Craft. (26 de febrero de 2021). *Software Testing Life Cycle.* Recuperado el 1 de noviembre de 2021 de https://qacraft.com/software-testing-life-cycle/

Oncins de Frutos. (s.f.). *NTP 283: Encuestas: metodología para su utilización.* https://www.cso.go.cr/legislacion/notas_tecnicas_preventivas_insht/NTP%20283%20-%20Encuestas%20metodologia%20para%20su%20utilizacion.pdf

O'Regan, G. (2019). *Concise Guide to Software Testing.* Springer.

Parasoft. (27 de mayo de 2021). *How to Write Test Cases for Software: Examples & Tutorial.* Recuperado el 1 de noviembre de 2021 de https://www.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/

Román, L. L., y Ramírez, F. (2012). *Lógica para Computación.* EUNED.

Rojas, J. A., Candamil, C. H. y Roa, D. E. (2020). *Gestión de proyectos aplicada al PMBOK 6ED*. Editorial de la Universidad Pedagógica y Tecnológica de Colombia

SCALED AGILE, INC. (10 de Febrero de 2021). *SAFe*. Recuperado el 1 de noviembre de scaledagileframework.com/behavior-driven-development/

Specflow. (2021a). *Specflow by Tricentis*. Recuperado el 1 de noviembre de 2021 de https://specflow.org/learn/gherkin/?_gl=1*1by8rkj*_ga*MTM5Mjc5MDQyMC4xNjIwNzk4MDk5*_ga_BZ55XKTXC6*MTYzNjA3MDcwMy45OS4xLjE2MzYwNzA5MDkuMA..&_ga=2.164018365.365227418.1636002729-1392790420.1620798099

Selenium. (2021a). *Understanding the components.* Recuperado el 1 de noviembre de 2021 de https://www.selenium.dev/documentation/webdriver/understanding_the_components/

Selenium. (2021b). *The Selenium Browser Automation Project.* Recuperado el 1 de noviembre de 2021 de https://www.selenium.dev/documentation/

SmartBear. (2021a). *What is Automated Testing?* Recuperado el 1 de noviembre de 2021 de https://smartbear.com/learn/automated-testing/what-is-automated-testing/

SmartBear. (2021b). *Test plan vs Test design: What's the difference?* Recuperado el 1 de noviembre de 2021 de https://smartbear.com/test-management/test-design-vs-test-plan/

SmartBear. (2021). *Test Automation Frameworks.* Recuperado el 1 de noviembre de 2021 de https://smartbear.com/learn/automated-testing/test-automation-frameworks/

SmartBear. (2021). *A Handy Guide to Using Agile Methodology in Testing: Processes, Best Practices & Tools.* Recuperado el 1 de noviembre de 2021 de https://smartbear.com/test-management/agile-testing-best-practices/

Techopedia. (2021). *Module*. Recuperado el 1 de noviembre de 2021 de https://www.techopedia.com/definition/3843/module

Wiebe, A. J., y Chan, C. W. (2012). Ontology driven software engineering. En *IEEE: 2012 25th IEEE Canadian Conference on Electrical and Computer Engineering.* https://ieeexplore.ieee.org/abstract/document/6334938

W3schools. (2021). *C++ OOP.* Recuperado el 1 de noviembre de 2021 de https://www.w3schools.com/cpp/cpp_oop.asp

APÉNDICE A. GHERKIN

Gherkin es un tipo de lenguaje perteneciente al *framework Cucumber*, el cual está diseñado para escribir casos de uso o casos de prueba de forma no técnica, y de forma legible para un humano (Specflow, 2021). Utiliza diferentes palabras reservadas o palabras clave para la descripción del escenario.

- *Given*: se utilizan para describir el contexto inicial del sistema.
- *When*: define la acción de prueba que se ejecutará.
- *Then*: define el resultado esperado de los pasos previos,
- *And*, *Or*: estas palabras son utilizadas para darle un mejor contexto y descripción al escenario.

Un escenario sencillo escrito con *Gherkin*, se muestra de la siguiente forma:

Figura 25
Gherkin

```
1   Feature: Calculator
2
3   Calculator for adding two numbers
4
5   Scenario: Add two numbers
6   Add two numbers with the calculator
7   Given I have entered 50 into the calculator
8   And I have entered 70 into the calculator
9
10  When I press add
11  Then the result should be 120 on the screen
```

Fuente: Specflow (2021).

APENDICE B. CUCUMBER

Cucumber, es una herramienta para la cual soporta el framework y método de desarrollo y pruebas denominado “Behavior Driven Development.” Dicha herramienta se utiliza para implementar pruebas de automatización basadas en escenarios escritos en lenguaje Gherkin en archivos de texto de tipo .feature. (Cucumber, 2019).

Esta herramienta puede ser utilizada en diferentes lenguajes de programación, como *Java (Cucumber-JVM) Python (Behave), C# (Specflow), Node.js (Cucumber,js)* entre otros lenguajes. (Cucumber, 2019).

APENDICE C. BEHAVIOR DRIVEN DEVELOPMENT

Es un método de desarrollo y pruebas también denominado "*Specification by Example*". Como se explica en el sitio web de Scaled Agile, Inc. (2021), *Behavior Driven Development* se define como: "una práctica que proporciona calidad incorporada al definir (y potencialmente automatizar) las pruebas antes o como parte de la especificación del comportamiento del sistema. BDD es un proceso colaborativo que crea una comprensión compartida de los requisitos entre la empresa y los equipos ágiles. Su objetivo es ayudar a guiar el desarrollo, disminuir la repetición y aumentar el flujo." (Scaled Agile, 2021).

APENDICE D. AGILE

Agile es un enfoque para la gestión de proyectos y el desarrollo de software. Agile ayuda a los equipos a entregar más rápido a sus clientes.

La metodología ágil se enfoca en que los requisitos de software evolucionan durante el desarrollo de los equipos de desarrollo y sus clientes. Las pruebas de software en desarrollo ágil, involucra a todos los miembros del equipo de desarrollo y del equipo que es parte del negocio. Por lo que todo el personal involucrado es responsable de la calidad, lo que quiere decir que no solo el personal de *quality assurance* o los *testers* son los únicos que realizan pruebas en los sistemas y el producto. (Smartbear, n.d.)

APENDICE E. MODULAR BASED FRAMEWORK

El *modular base framework* se centra en dividir la aplicación bajo prueba en funciones o separadas, por lo que cada módulo de la aplicación será probada por aparte. Cuando la aplicación es dividida en módulos, se crea un script de prueba para cada función y luego se combina para construir pruebas más grandes de forma jerárquica. Estos conjuntos más grandes de pruebas comenzarán a representar varios casos de prueba. (Smartbear, 2021)

APENDICE F. UNIT TESTING

El *unit testing* o pruebas unitarias son las pruebas de más bajo nivel. Están hechas de los métodos y funciones que utiliza la aplicación. Son las pruebas más fáciles y rápidas de ejecutar y suelen ser llevadas a cabo solamente por los desarrolladores cuando realizan el código que la aplicación Atlassian. (2021b). Estas pruebas comprueban pequeñas secciones del código, y son ejecutadas en insolación. Existen diferentes frameworks para realizar pruebas unitarias, dependiendo del lenguaje de desarrollo, como por ejemplo: Nunit, XUnit, Jasmine, Mocha, entre otros.