INSTITUTO TECNOLÓGICO SUPERIOR DE MISANTLA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

TRABAJO:

“Diseño de la estrategia de automatización e implementación de herramientas para realizar pruebas de aplicaciones del sistema operativo móvil Android”

QUE PRESENTA

ALCIBIADES GARCIA REYES

DIRECTOR:

MSC. JOSÉ ANTONIO HIRAM VÁZQUEZ LÓPEZ

CODIRECTORES

MIA. ROBERTO ÁNGEL MELÉNDEZ ARMENTA

DR. SIMÓN PEDRO ARGUIJO HERNÁNDEZ

MISANTLA, VERACRUZ JUNIO 2020

INSTITUTO TECNOLÓGICO SUPERIOR DE MISANTLA
DIVISIÓN DE ESTUDIOS PROFESIONALES
AUTORIZACIÓN DE IMPRESIÓN DE TRABAJO DE TITULACIÓN

FECHA: 27 de Mayo de 2020.

ASUNTO: **AUTORIZACIÓN DE IMPRESIÓN DE TESIS PROFESIONAL.**

A QUIEN CORRESPONDA:

Por medio de la presente hago constar que el (la) C:

ALCIBIADES GARCÍA REYES

pasante de la carrera de INGENIERÍA EN SISTEMAS COMPUTACIONALES con No. de Control 152T0021 ha cumplido satisfactoriamente con lo estipulado por el **Manual de Procedimientos para la Obtención del Título Profesional de Licenciatura** bajo la **opción Titulación Integral (Tesis Profesional)**

Por tal motivo se **Autoriza** la impresión del **Tema** titulado:

"DISEÑO DE LA ESTRATEGIA DE AUTOMATIZACIÓN E IMPLEMENTACIÓN DE HERRAMIENTAS PARA REALIZAR PRUEBAS DE APLICACIONES DEL SISTEMA OPERATIVO MÓVIL ANDROID"

Dándose un plazo no mayor de un mes de la expedición de la presente a la solicitud del Acto de Recepción para la obtención del Titulo Profesional.

ATENTAMENTE

ING. GERBACIO TLAXALO ESPINOZA
DIVISIÓN DE ESTUDIOS PROFESIONALES

DIVISIÓN DE ESTUDIOS PROFESIONALES

Archivo.

VER. 01/03/09 F-SA--39

Agradecimientos

A mi gran familia por su apoyo constante, por ser mi pilar fundamental y haberme apoyado incondicionalmente, en todos los retos y adversidades que se presentaron, quienes me dieron la enseñanza de que con dedicación y esfuerzo las metas se hacen posible.

A todos mis maestros del tecnológico superior de Misantla especialmente a la academia de sistemas computaciones por brindarme las herramientas y conocimientos necesarios que fueron base para realizar este proyecto, por su entusiasmo en la enseñanza para que los alumnos sean profesionistas de calidad.

A mi director de tesis MSC. José Antonio Hiram Vázquez López por haberme guiado, no solo como director de tesis, sino a lo largo de mi trayectoria universitaria, por su rigidez como docente y la constante motivación de hacer las cosas dando el mayor esfuerzo posible.

A mis codirectores de tesis DR. Simón Pedro Arguijo Hernández y MIA. Roberto Ángel Meléndez Armenta, por su tiempo dedicado en el proceso de revisión en la tesis, así como también por sus enseñas a lo largo de mi carrera profesional.

A los gerentes de la empresa DigiPro S.A. de C.V. por la confianza y el apoyo contantes que me brindaron a lo largo de mis prácticas profesionales, y los valiosos conocimientos que aportaron a este proyecto.

Resumen

Para tener productos de calidad en el desarrollo de software, es necesario contar con algunos factores determinantes, esto son: la infraestructura humana, las tecnologías y los procesos, los cuales tienen un papel fundamental en la calidad del producto. En las actividades del *testing*, las pruebas de software sirven para detectar incidencias antes de que el producto salga a producción. Cada sistema tiene un nivel de complejidad, por lo cual ya no es suficiente con pruebas manuales, es aquí donde entra la automatización de pruebas de software ya que es una estrategia que hoy en día es muy utilizada por las empresas de desarrollo de software.

En la empresa DigoPro en la cual se desarrolló este proyecto, el proceso de *testing* era mediante pruebas manuales, algunos de los problemas identificados fueron: errores detectados de forma tardía, las pruebas duraban más del tiempo establecido, se utilizaba personal de otras áreas debido a los tiempos de entrega ya que el equipo de *testing* no podía satisfacer la carga de trabajo, no se realizaban las pruebas de regresión de forma completa.

Es por ello por lo que se elaboró el presente proyecto en el cual se desarrollaron pruebas automatizadas para el departamento de Android, esto con la finalidad de poder automatizar las pruebas de regresión y de funcionalidad para lograr reducir las incidencias y los tiempos de la ejecución de las pruebas.

La implementación de esta herramienta dio como resultado la formación de una nueva área en el departamento de testing, así como la formación de nuevos procesos de calidad que permiten cubrir un mayor margen de pruebas, los *scripts* desarrollados cubren una gran cantidad de caso de prueba para poder tener un producto de calidad.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades del proyecto

1.1 Introducción

El desarrollo de software siempre está sujeto a diversos errores, existe una gran necesidad por parte de las compañías de garantizar un producto de calidad, es por ello que han surgido diversas metodologías para el desarrollo y ejecución de pruebas de software, de esta manera el usuario final pueda ver un producto terminado de calidad. Los fallos que se presentan en el desarrollo de un sistema siempre son inesperados, es difícil y complejo poder evitarlos, es por ese motivo que es muy importante implementar un adecuado proceso de pruebas de software que permitan certificar desde el punto de vista de la calidad de un producto de software.

El proyecto que se realizó en la empresa DIGIPRO S.A. de C.V. con ubicación en la Ciudad de México, consistió en el desarrollo de pruebas automatizadas, para las pruebas funcionales y de regresión enfocadas a dispositivos Android para el mejoramiento en el área de Tecnologías de la informática en el área de "*Tester*" lo que permitió agilizar el proceso ya que se contaba con un cuello de botella en este departamento, y es importante para la empresa entregar productos de calidad. La aplicación debe ser verificada y validada en cada etapa del proceso de desarrollo.

El procedimiento que se efectuaba para realizar las pruebas funcionales de la empresa era de forma manual, esto representaba invertir muchas horas por cada liberación de versión lo cual lo convertía en un proceso muy tedioso, este tipo de pruebas a medida que va creciendo el proyecto no son viables, ya que las tareas repetitivas para una persona pueden ser cansadas y esto causa que se vayan dejando muchos errores en el camino.

La propuesta para mejorar esta área fue introducir la automatización de las pruebas funcionales para el área de Android, este tipo de pruebas fue una alternativa interesante para la empresa, desde hace mucho tiempo buscaban poder optimizar esta área ya que era de suma importancia asegurar cierto nivel de calidad antes de cada liberación de versión. Las pruebas automatizadas se realizaron utilizando Appium la cual es una herramienta de automatización de pruebas de código abierto que se puede utilizar en aplicaciones web nativas, híbridas y móviles, también se utilizó java para desarrollar los *scripts* necesarios para las pruebas.

En este documento se describen los objetivos y las actividades desarrolladas en esta empresa, se utiliza la metodología Scrum la cual es tendencia en la gestión de proyectos. El sector del desarrollo de software es el principal representante de este tipo de metodología ágil. Se trata de planificar el proyecto en pequeños bloques o *Sprints*, e ir revisando y mejorando el anterior. Es por ello que la etapa de *testing* es muy importante pues en cada etapa del proyecto se debe ir revisando que todo está en orden y que las funcionalidades desarrolladas tienen un buen comportamiento, así como la integración con las ya existentes.

1.2 Planteamiento del Problema

En el desarrollo de software, las pruebas funcionales y de validación buscan comprobar que el software hace lo que el usuario espera, esto convierte esta área en un foco importante porque es el *tester* el que debe poner el sello de calidad del producto final, se requiere de un alto compromiso por parte del equipo involucrado. Cuando se libera una nueva versión del producto se deben hacer diversas pruebas, de regresión, integración y funcionales, las pruebas deben ser ejecutadas en más de una ocasión lo cual causa un alto impacto en el tiempo invertido.

El proceso de *testing* de la compañía DigiPro presentaba los siguientes problemas.

1. Las pruebas manuales de regresión utilizaban más tiempo del que se tenía a disposición debido a las entregas y liberaciones continuas. Estas se realizan en cada *release* que se liberaba, pero dichas pruebas son limitadas en sólo una parte de su totalidad, y otro factor es que debido a que son pruebas manuales están sujetas a errores, debido a la repetitividad del proceso.
2. Las pruebas de calidad no se realizaban de forma conjunta con el ciclo de desarrollo, es decir no se hacían durante el *Sprint* en curso sino al finalizar estos, lo cual causaba que la detección de errores demorara mucho tiempo lo que en su mayoría de veces causaba entregas de productos con defectos los cuales posteriormente repercutían en más horas de mantenimiento por parte del equipo de desarrollo.
3. El *testing* manual era muy tedioso ya que las pruebas se debían repetir en ocasiones hasta más de 20 veces al día, esto para un humano puede ser cansado, y con cada prueba se podían ir pasando errores debido a la fatiga del equipo involucrado.

4. Debido a la magnitud de las pruebas y los acortados tiempos, en ocasiones se utilizaba personal de otras áreas para apoyar en las pruebas. Lo anterior, también afectaba el cumplimiento de las metas internas de las otras áreas al ceder parte de su personal para este tipo de actividades.
5. Por último, los casos de prueba de regresión ejecutados de forma manual no cubrían el 100% de las funcionalidades de los sistemas, impidiendo que se descubrieran posibles defectos adicionales.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar estrategias automatizadas que ayuden a optimizar las pruebas de regresión y funcionales existentes en la empresa DigiPro, implementado diferentes herramientas enfocadas al sistema operativo móvil Android, para mejorar la productividad y la calidad, reduciendo las incidencias, los tiempos y las horas hombres necesarias para las pruebas de calidad.

1.3.2 Objetivos específicos

Los objetivos específicos que se contemplaron para este proyecto son:

- Disminuir el tiempo que toma *testear* cada liberación de versión de la aplicación Android con las nuevas funcionalidades y correcciones que se van desarrollando,
- Bajar las horas hombres que se invierten en pruebas manuales por el equipo de *testing*.
- Reducir la cantidad de personal para las pruebas *UAT* donde la validación funcional considerará sólo realizar pruebas manuales de las nuevas funcionalidades de los *releases* y no de las ya existentes
- Hacer que lo *script* ejecuten una prueba funcional al menos 100 veces en un día.
- Generar información aleatoria utilizando un Json para que la información sea variada, como, por ejemplo: Nombre, apellido paterno, apellido materno, edad, etc.
- crear las pruebas automatizadas de manera manual para que se puedan ejecutar en tiempo programado.

- Hacer que las pruebas automatizadas corran en una granja de dispositivos conectada a la nube.

1.4 Justificación.

En la actualidad las empresas de desarrollo de software tienen un área de *testing* para darle mayor calidad a su producto final, pero no todas las empresas cuentan con un área de automatización de pruebas lo que hace que estas áreas tengan un fuerte impacto en los tiempos de producción lo cual se ve reflejado en los costó del proyecto.

Las empresas buscan tener una mayor productividad lo cual representa un claro desafío como es el caso de la empresa DigiPro S.A de C.V. Cada año las diferentes áreas tienen metas a cumplir para mejorar la productividad en los tiempos y costes de los diversos proyectos. En el área de *testing* se habían hecho diversas propuestas para tener mejores resultados ya que existen muchas alternativas al momento de enfrentar la eficiencia de la productividad, se detectó que se podía mejorar la realización de las pruebas de forma manual en el área de *testing*, teniendo una oportunidad muy fuerte de mejorar este proceso.

Las pruebas manuales para regresión y de funcionalidad suelen tardar entre 4 a 5 días por cada liberación de *release*. Cuando se encuentran incidencias de *stopper* en las liberaciones suele tardarse 2 semanas para volver a probar un nuevo *release*.

Las pruebas automatizadas tuvieron un acto impacto en mejora el proceso de calidad de las aplicaciones, ya que con ellas se pudo abarcar un mayor número de pruebas, algunos de los errores encontrados en las aplicaciones durante la automatización, tal vez no hubieran sido encontrados de forma manual debido a las limitaciones del tiempo.

Otra gran ventaja que proporcionó este proyecto fue que el equipo de desarrollo logró una estabilización temprana de su código, ya que al realizarse más pruebas en corto tiempo los errores eran localizados tempranamente y esto ayudo a tener una base de código sólida para la aplicación. Esto evitó trabajo posterior a los desarrolladores ya que al hacer correcciones tempranas se impidió construir encima de un código con defectos.

La mayor justificación para implementar este proyecto fue reducir el tiempo y esfuerzo que requiere el proceso de pruebas manuales, lo cual permitió al equipo de calidad realizar los

ciclos de pruebas de manera óptima, tuvo una gran ventaja reduciendo el costo por el tiempo invertido de las pruebas.

1.5 Alcances

- Se determinó la viabilidad del proyecto, su rentabilidad, su ventaja y los posibles costos de inversión.
- Se analizaron las herramientas para automatización de pruebas que mejor se adecuaran con el proyecto.
- Se incorporó un archivo Json para que la información generada fuera de forma aleatoria
- Se pudo hacer que el *script* corriera de forma automática más de 100 veces una prueba funcional.
- Se definió un plan estratégico para que el proyecto se realice en tiempo y forma.
- Se automatizó un proyecto que estaba en desarrollo en sus partes más críticas, las cuales ocupaban más tiempo de inversión por parte del equipo de *testing*.
- Se logró reducir los tiempos de inversión en pruebas funcionales y de regresión.
- Se pudo reducir la cantidad de incidencias, errores y bugs antes de que la aplicación saliera a operaciones.

1.6 Limitaciones

- Los tiempos para investigar cada una de las herramientas fueron reducidos.
- La documentación existente para pruebas automatizadas es muy reducida.
- No se pudo incorporar al proyecto con la herramienta de control de versiones “SVN”
- No se logró que la ejecución de los *scripts* fuera de forma automática a una hora y fecha determinada.
- No se pudo conectar Appium en la nube para ejecutar las pruebas con varios dispositivos al mismo tiempo.

Capítulo 2. Marco teórico

2.1 Pruebas de software

Las pruebas de software se pueden describir como un conjunto de actividades y técnicas, cuyos principales objetivos es poder proporcionar información de manera objetiva e independiente sobre la calidad del producto realizado para los usuarios finales (ver figura 1).

Las pruebas de software se definen como una actividad en la cual un sistema o uno de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados se observan, registran y se realiza una evaluación de algún aspecto [1].

El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba) [1]. Cuando se detectan errores no significa que existe una mala práctica por parte del área de desarrollo ya que es imposible no cometer errores y la detección de ellos implica un éxito para la mejora de la calidad del producto.

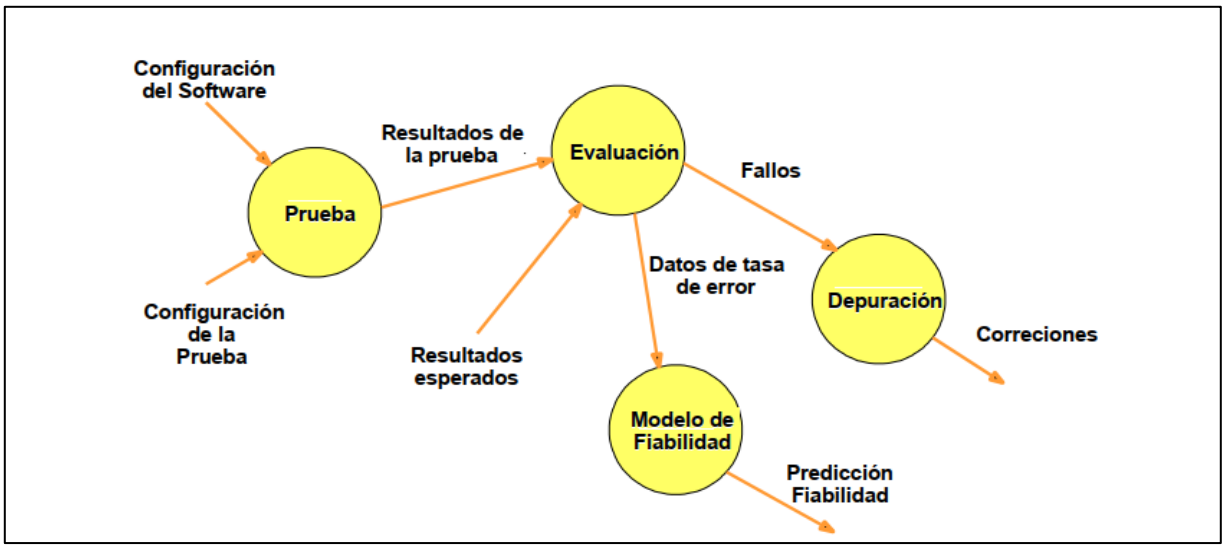

Figura 1 Contexto de pruebas de Software

Existen algunas definiciones importantes en el campo de las pruebas de software las cuales son [1]:

Prueba (test): Es la actividad en la cual un sistema o alguno de los componentes que lo conforman se ejecutan en situaciones controladas, las cuales fueron previamente

especificadas, los resultados arrojados se observan y se registran para realizar una evaluación de algún aspecto en concreto.

Casos de prueba (test case): Conjunto de valores concretos de entrada, condiciones de ejecución y resultados esperados, los cuales se desarrollan para un objetivo en específico.

Defecto (defect, fault, bug): Un defecto en el software son resultados no esperados durante la ejecución de la aplicación, dichos resultados son procesamientos incorrectos en un programa.

Fallo (failure): Es la incapacidad de un sistema o de alguno de sus componentes para ejecutar las funciones que son requeridas dentro de los requisitos del rendimiento específico.

2.2 Gestión de pruebas

Para llevar a cabo un desarrollo de pruebas de software no basta con el conocimiento de cómo ejecutarlas, también se necesitan realizar otras tareas adicionales; entre estas tareas se encuentra la planificación de pruebas, la generación de casos de uso, el seguimiento de los errores, manejar las configuraciones del sistema y otras tareas. La ejecución de todas estas tareas es lo que se puede denominar como gestión de pruebas dado que la mayoría de las veces son supervisadas por un individuo y realizadas por el equipo de trabajo [2].

2.3 Pruebas funcionales

Las pruebas funcionales se pueden definir como aquellas que buscan, identificar inconsistencias, asegurar requisitos funcionales, reducir costos, evitar reprocesos, mejorar la productividad, y aumentar la satisfacción del cliente [3].

El objetivo de la prueba funcional es validar cuando el comportamiento observado del software probado cumple o no con sus especificaciones. La prueba funcional toma el punto de vista del usuario [4]. Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna del programa raramente es considerada [5]. Esta etapa suele ser la última etapa de pruebas antes de que el producto salga a producción. A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra.

Para poder diseñar casos de prueba que proporcionen un buen resultado, el ingeniero de software debe tener en cuenta los principios de las pruebas [6].

Para poder ejecutar las pruebas funcionales, se deben analizar las especificaciones para poder generar los casos de prueba. Se deben tomar en cuenta respuestas inválidas e inesperadas de la entrada y considerar que la definición del resultado que se espera es una parte primordial de un caso de pruebas. El propósito de la prueba funcional es mostrar discrepancias con la especificación y no demostrar que el programa cumple su especificación [7]. Existen diferentes técnicas de pruebas funcionales como por ejemplo las pruebas caja blanca y caja negra.

2.4 Caso de prueba

Existen diferentes definiciones para este concepto, se pueden ver alguna de ellas como la siguiente:

Un caso de prueba (*test case*) es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y postcondiciones de ejecución, desarrollados con un objetivo particular o condición de prueba, tal como ejercitar un camino de un programa particular o para verificar que se cumple un requerimiento específico [8].

De acuerdo con el Estándar IEEE 610 (1990) [8] un caso de prueba se define como: “Un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico.”

Los casos de prueba deben incluir algunos elementos en su descripción, entre los cuales se encuentran:

- Flujo: Secuencia de pasos a ejecutar
- Datos entrada
- Estado inicial
- Valor de respuesta esperado
- Estado final esperado

2.5 Diseño básico de casos de prueba

Existen diferente técnica para diseñar casos de pruebas, algunas son básicas las cuales se pueden ocupar para comenzar un proyecto (ver figura 2), después conforme se vaya avanzando se pueden ir robusteciendo. Lo primero que se tiene que realizar es determinar el flujo de casos de prueba, lo cual se refiere a seguir los pasos de una funcionalidad específica por ejemplo el login de un sistema.

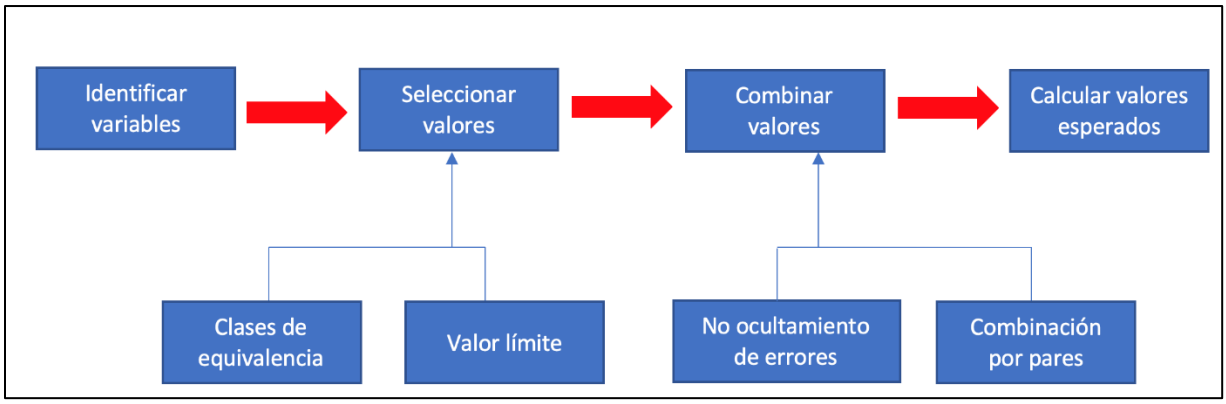

Figura 2. Proceso estándar de diseño de pruebas

Como primer paso se debe identificar las variables que interactúan con la funcionalidad a probar; después, para cada una de las variables se decidirá utilizar distintos datos, los cuales sean interesantes desde el punto de vista del *testing*, para lo cual es útil aplicar técnicas de partición en clases de equivalencia y valores límites; por último, estos valores deben combinarse de alguna forma, porque seguramente el producto cartesiano (todas las combinaciones posibles) daría una cantidad de casos de prueba muy grande, y quizá muchos de esos casos tendrán “poco valor agregado”.

Un caso de prueba (*test case*) es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y postcondiciones de ejecución, desarrollados con un objetivo particular o condición de prueba, tal como ejercitar un camino de un programa particular o para verificar que se cumple un requerimiento específico.

2.6 Pruebas dirigidas por datos

Esta técnica del *testing* resulta muy útil ya que es utilizada para construir casos de pruebas basados en los datos de entrada y salida, separando el flujo que se toma como parte de la

aplicación. Se puede decir que por un lado se representa el flujo (la serie de pasos para ejecutar el caso de prueba) y por otro lado se van almacenando los datos esperados.

2.7 Pruebas de caja negra y caja blanca

Pruebas de caja negra: Este tipo de pruebas también son llamadas pruebas de comportamiento, estas pruebas se basan en la especificación del programa o los componentes que serán probados para elaborar un caso de prueba, esto es muy independiente del diseño interno del programa. Es prácticamente basarse únicamente en los valores de entrada y de salida del sistema.

Prueba de caja blanca: A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Cuando se utiliza información interna del sistema que se está probando, tal como el código fuente, base de datos, etc., entonces se está hablando de un técnico de caja blanca. Este método se centra en como diseñar los casos de pruebas con base a la estructura del programa.

En realidad, estos dos tipos de pruebas son técnicas complementarias que han de aplicarse al realizar una prueba dinámica, ya que pueden ayudar a identificar distintos tipos de faltas en un programa (ver figura 3).

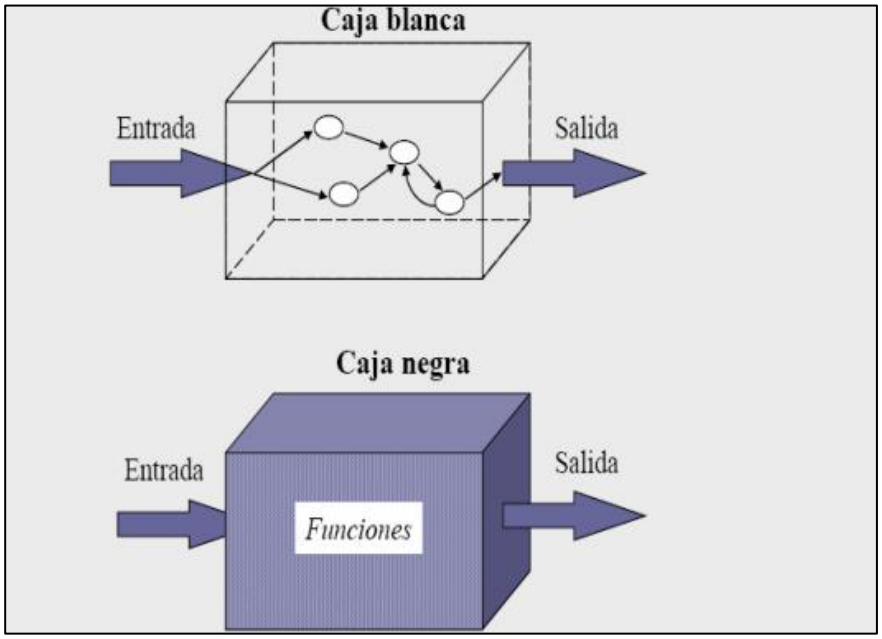

Figura 3. Representación de pruebas de caja blanca y caja negra

2.8 Test de regresión

Las pruebas de regresión se pueden considerar como un subconjunto de las pruebas planificadas las cuales se seleccionan para ejecutar cada cierto tiempo, por ejemplo, cada que se requiere liberar nueva versión del producto. Su principal objetivo es verificar que las nuevas versiones que están por salir no hayan sufrido regresiones. Se denomina pruebas de regresión cuando se intenta descubrir errores, carencias de funcionalidad con respecto al comportamiento correcto del software, causado por realizar cambios el programa por parte del equipo de desarrollo, se puede decir que es verificar que las partes que antes no fallaban sigan funcionando adecuadamente.

Se evalúa el correcto funcionamiento del software desarrollado frente a evoluciones o cambios funcionales. Si los usuarios tienen la versión N instalada, y le instalan N+1, y esta tiene fallos, el equipo de trabajo entrará en conflicto al tener que volver a la versión previa, hacer una regresión a la versión N. Generalmente cuando se diseñan las pruebas para determinadas funcionalidades, ya se definen cuáles son las pruebas que serán consideradas dentro del *set* de pruebas de regresión. O sea, las que serán ejecutadas ante cada nueva liberación del producto, o en cada ciclo de desarrollo [9].

Las pruebas de regresión tienen como objetivo verificar que no ocurrió una regresión en la calidad del producto luego de un cambio, asegurándose que los cambios no introducen un comportamiento no deseado o errores adicionales. Implican la reejecución de alguna o todas las pruebas realizadas anteriormente.

2.9 Pruebas funcionales

Las pruebas funcionales tienen el objetivo de validar cuando el comportamiento que se observa del software que se prueba cumple o no con las especificaciones requeridas. La prueba funcional toma el punto de vista del usuario. Las funciones se prueban teniendo en cuanta los valores de entrada y examinando su salida. La estructura interna del programa raramente es considerada.

Para poder ejecutar las pruebas funcionales, las especificaciones deben ser analizadas para que se deriven los casos de prueba. Existes diferentes técnicas pertinentes para las pruebas

funcionales, las cuales son partición de equivalencia, análisis del valor límite, grafo causa-efecto y conjetura de errores.

Siempre se debe tener en consideración las condiciones que pueden resultar inválidas e inesperadas de la entrada y tener en cuenta que la definición del resultado que se espera es una parte esencial de un caso de prueba.

El propósito de la prueba funcional es mostrar discrepancias con la especificación y no demostrar que el programa cumple su especificación.

2.10 Script de prueba

Un script de prueba (*test script*) son los datos y las instrucciones escritas con una sintaxis formal, almacenado en un archivo y usado por una herramienta de automatización de las pruebas. Un *script* de prueba puede automatizar uno o más casos de prueba, navegación, inicialización u operaciones de configuración del entorno. Un *script* de prueba previsto para la ejecución manual de las pruebas es un procedimiento de prueba [10].

Una suite de pruebas es uno o más conjuntos de pruebas reunidos para satisfacer un objetivo de prueba. Un conjunto de prueba incluye *scripts* y documentación.

2.11Testing automatizado

El *testing* automatizado consiste en que una máquina logre ejecutar los casos de prueba en forma automática, la manera en que se logran leer las especificaciones puede ser mediante *scripts* en un lenguaje genérico o propio de alguna herramienta a partir de planillas de cálculo, modelos, etc. Por ejemplo, Appium y Selenium tienen un lenguaje llamado Selence, brindando un formato para cada comando (acción) que puede ejecutar, y de esa forma un *script* es una serie de comandos que respetan esta sintaxis. La herramienta además permite exportar directamente a una prueba JUnit en Java, y a otros entornos de ejecuciones de pruebas.

En las automatizaciones de pruebas funcionales se pueden utilizar herramientas de ejecución de captura y reproducción, es decir, se sigue un caso de prueba especifico mientras que a la par una herramienta de grabación va grabando los pasos para posteriormente ejecutarlos por su cuenta, este tipo de herramientas es recomendado en los casos en lo que no se requieren pruebas robustas. Estas herramientas permiten al *tester* grabar pruebas, para luego editarlas,

modificarlas y reproducirlas en distintos entornos. Las pruebas automatizadas que ocupan estas herramientas se enfocan más en la interfaz y no en el código de la aplicación, suelen ser pruebas más sencillas.

Durante la grabación se capturan las acciones realizadas por el *tester*, posteriormente se pueden generar script en algún lenguaje de alto nivel para poder robustecer las pruebas. Luego el *tester* modifica el script para crear una prueba reusable y mantenible. En general estas herramientas vienen acompañadas de un comparador, que verifica automáticamente la salida en el momento de ejecutar el script con la salida grabada.

Existen algunas prácticas para el desarrollo de la automatización de pruebas [10]:

- Se deben planear los casos de prueba de manera rápida, ya que generarlos y ejecutarlos requieren recursos, tanto de hardware como software, poder agilizar este proceso sirve para dar más tiempo a la ejecución manual de los procesos que no se pueden automatizar. Es necesario contar con estos recursos lo más pronto para poder empezar a identificar posibles fallas en el desarrollo.
- Es necesario tener una planeación clara sobre los casos de prueba para requerimientos funcionales y no funcionales, por cada tipo se debe contar con un caso de prueba ya que los funcionales ayudan en la identificación de fallas sobre la aplicación, los no funcionales sirven para identificar configuraciones y características para un mejor desempeño de la aplicación
- Se tiene que establecer una arquitectura, siendo fundamental implementarla de manera correcta, ya que esto ayudara a construir casos de pruebas robustos, mantenibles y escalables.
- Los casos de prueba a implementar no deben tomar mucho tiempo en desarrollar y probar para que sean eficientes.
- Los ingenieros pueden ayudar a desarrollar casos de prueba que funcionen de acuerdo a las entradas definidas y no serían casos de prueba con datos fijos.
- Tener casos de prueba basado en keywords (palabras claves) ayuda en el modularidad de los casos de prueba, ayuda a identificar fácilmente el objetivo de la función, los casos de prueba son más fáciles de mantener.

- Cuando se realizan pruebas sobre bases de datos y esas pruebas cambian la información de la base de datos, se debe tener la posibilidad de reiniciar fácilmente la información, con el fin de poder reutilizar los casos de prueba.
- La automatización de los casos de prueba siempre va a verse afectado por cambio de requerimientos, cambios en los cronogramas, cambios en las prioridades, es por esto que adoptar principios de las metodologías ágiles puede ayudar a minimizar el impacto por ese tipo de cambios.

2.12 Criterios de Selección de Casos de Prueba.

En cuanto una empresa decide iniciar el proceso de automatización se debe tener en claro que no es posible automatizar todos los casos de prueba existentes para su aplicación, es importante tener claro cuáles son los casos de prueba que deben ser automatizados. Una de las ideas principales de las pruebas automatizadas es que puedan reutilizarse la mayor cantidad de veces. Existen diversos criterios a tener en cuenta para lograr tener beneficios exitosos, estos criterios son:

- Casos de pruebas repetitivos que se puedan ejecutar múltiples veces.
- Casos de prueba que tienden a causar error humano (al ejecutarse de forma manual).
- Casos de prueba que requieren múltiples conjuntos de datos.
- Funcionalidades que son usadas frecuentemente y que presentan condiciones de alto riesgo.
- Casos de prueba que son imposibles o difíciles de ejecutar manualmente.
- Casos de prueba que se deben ejecutar en múltiples plataformas (Hardware, Software).
- Casos de prueba que requieren de esfuerzo y tiempo al ser ejecutadas manualmente.

2.13 Metodología Scrum

Esta metodología se puede definir como un marco de trabajo para el desarrollo ágil que permite prácticas emergentes y un mantenimiento de productos complejos, permite la entrega de productos con un valor productivo elevado.

Scrum demuestra la eficiencia de las técnicas de gestión de proyectos y las técnicas de trabajo con la finalidad de que haya una mejora continua en el producto, el equipo involucrado y el entorno de trabajo.

Teoría de Scrum

Está basada en la teoría del control procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo. Tres pilares soportan toda la implementación del control de procesos empírico: transparencia, inspección y adaptación.

Scrum prescribe cuatro eventos formales, contenidos dentro del *Sprint*, para la inspección y adaptación.

- Planificación del *Sprint* (*Sprint Planning*)
- Scrum Diario (*Daily Scrum*)
- Revision Del Sprint (*Sprint Review*)
- Retrospectiva del Sprint (*Sprint Retrospective*)

El Equipo Scrum (Scrum Team)

En la guía oficial de Scrum los autores proponen que el equipo debe estar conformado por 3 partes y se debe conformar de al menos 3 a 9 personas. El primer actor del equipo Scrum es el dueño del producto (Product owner), le sigue el equipo de desarrollo (Development team) y un Scrum Master. Los integrantes del equipo deben ser auto organizados con múltiples funcionalidades, deben elegir la mejor forma de llevar a cabo el trabajo, es necesario que tengan competencias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo.

El modelo de equipo en Scrum (ver figura 4) está diseñado para optimizar la flexibilidad, la creatividad y la productividad. Los miembros del equipo Scrum deben entregar productos de forma iterativa e incremental, estos incrementos se deben integrar de manera correcta a los anteriores, de esta manera el producto va tomando forma mientras más iteraciones se realizan, con esto se maximizan las oportunidades de obtener retroalimentación. Las entregas incrementales de producto terminado aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto [11].

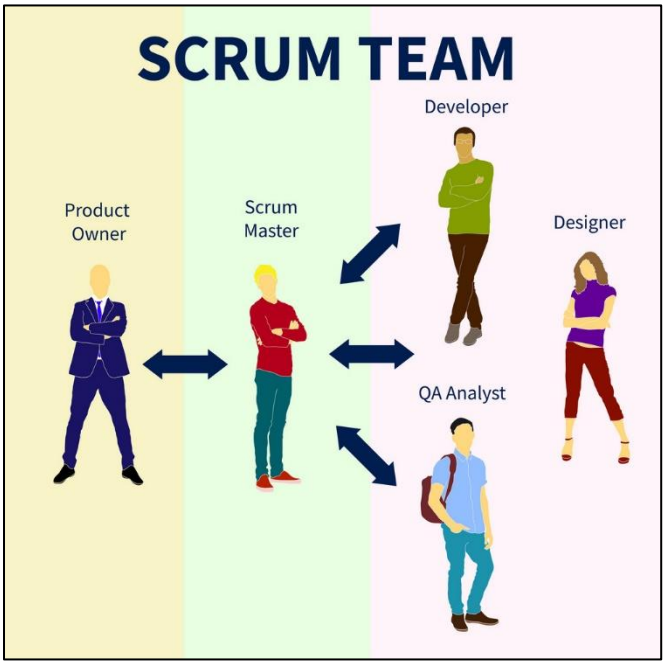

Figura 4. Equipo Scrum

Eventos de Scrum

En la guía oficial Scrum los autores definen que los eventos son las reuniones que tienen que mantener todo el equipo de trabajo. Scrum contempla eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas. Los eventos son bloques de tiempo (time-boxes), de tal manera que todos tienen una duración máxima. Iniciado un *Sprint*, la duración es fija y no tiene la capacidad de acortarse o alargarse.

El resto de eventos pueden terminar una vez que se logre el objetivo del evento, de igual manera, se debe asegurar que se emplee una cantidad apropiada de tiempo y sin desperdicio en el proceso (ver figura 5).

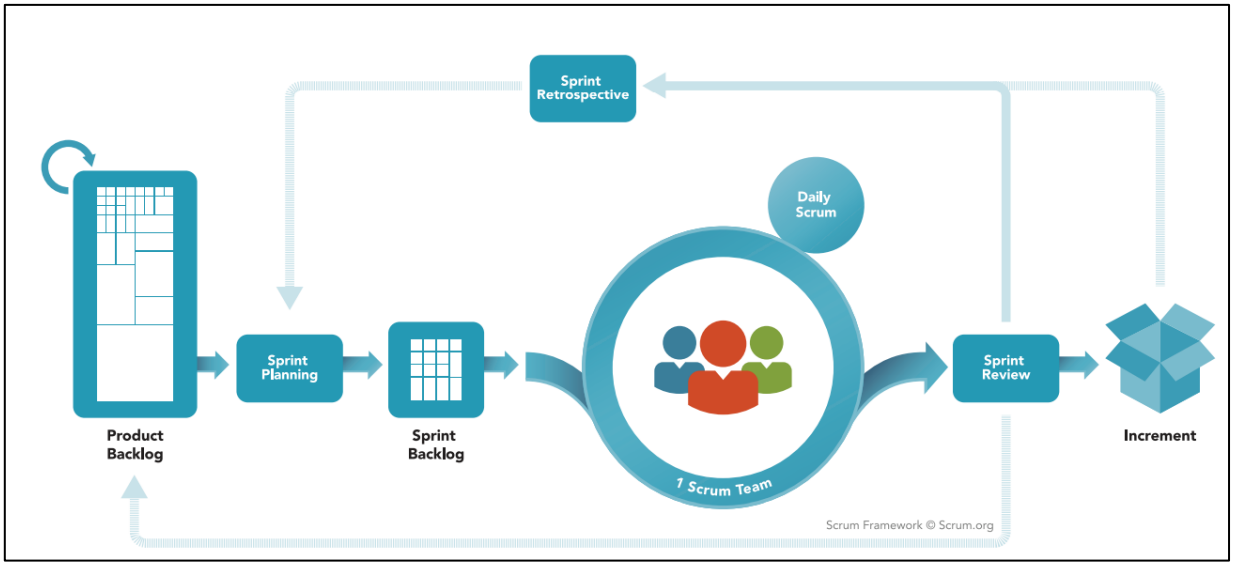

Figura 5. Eventos Scrum

El Sprint

La parte fundamental y tal vez el elemento más importante de Scrum es el *Sprint*, definido como un bloque de tiempo (time-box) que va de un mes o menos durante la cual se general entregas parciales del producto, conocido como incrementos, lo cuales tienen que poder ser utilizados y potencialmente desplegables, cada nueva etapa del *Sprint* comienza justo cuando termina la anterior. Se tiene que tener una planificación del Sprint (*Sprint Planning*), los Scrums diarios (*Daily Scrums*), el trabajo de desarrollo, la revisión del *Sprint* (*Sprint Review*), y la retrospectiva del *Sprint* (*Sprint Retrospective*).

Un *Sprint* es considerado como un proyecto usado para lograr una meta con un horizonte de tiempo no mayor de un mes. Cada *Sprint* tiene una meta definida a construir, diseño y plan flexible que guiará su construcción, el trabajo del equipo y el incremento de producto resultante. Cuando el horizonte de un *Sprint* considera un tiempo muy amplio entonces es pertinente realizar cambios en el producto a construir por lo que la complejidad y riesgo podrían incrementarse. Los *Sprints* habilitan la predictibilidad al asegurar la inspección y adaptación

del progreso al menos en cada mes calendario. Los *Sprints* también limitan el riesgo al costo de un mes calendario [11].

Planificación de Sprint (Sprint Planning)

Es uno de los puntos fundamentales de la metodología Scrum ya que es el punto de partida para saber que es el trabajo que se realizará Esta planificación tiene una duración de máximo 4 horas cuando son *Sprint* de dos semanas. Para Sprints más cortos las juntas suelen durar mucho menos. El Scrum master es el encargado de que el evento se lleve a cabo y que todos los miembros del equipo entiendas su propósito. El Scrum master se tiene que asegurar de que todo entre en los tiempos establecidos.

Primera etapa de la reunión

La primera etapa de la reunión tiene una duración aproximada de 2 horas para *Sprints* de dos semanas:

- El cliente expone al equipo una lista de requisitos, cada requisito tiene una cierta prioridad que van desde el más importante hasta los menos. Esta lista de requisitos sirve para nombrar las metas de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella.
- Los miembros del equipo deben examinar la lista, es necesario que se resuelvan las dudas que se tengan para no afectar los tiempos posteriormente. Añade más condiciones de satisfacción y selecciona los objetivos/requisitos más prioritarios que prevé completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

Segunda etapa de la reunión

Esta etapa se realiza en un tiempo aproximado de 2 horas. El equipo planifica la iteración, se elaboran técnicas que ayuden a facilitar las entregas para obtener mejores resultados y minimizar esfuerzo. Esta actividad la realiza el equipo puesto que son los responsables de organizar su trabajo.

- Definen las tareas necesarias para poder completar cada objetivo/requisito, es aquí cuando se crea la lista de tareas llamada *Sprint backlog*. Todo esto está basado en la definición de hechos (DoD).
- Realiza una estimación conjunta del esfuerzo necesario para realizar cada tarea.
- Cada miembro del equipo se autoasigna las tareas que creen convenientes que puede realizar, se pueden crear tareas en parejas con la finalidad de compartir conocimiento o para resolver juntos objetivos especialmente complejos.

Objetivo Del Sprint (Sprint Goal)

El objetivo del *Sprint* es una meta establecida para el *Sprint* que puede lograrse mediante la implementación de la Lista de Producto [11]. Su propósito es brindar una guía a los involucrados en el desarrollo. El objetivo se propone durante la planificación del *Sprint* de esta manera se brinda flexibilidad con respecto a las funcionalidades que se van implementadas. El objetivo del *Sprint* generalmente logra establecer un enlace de unión que permite determinar que el equipo de desarrollo trabaje en forma conjunta y coordinada. Una vez establecidos los objetivos del *Sprint*, el equipo de desarrollo inicia la implementación del producto bajo consideraciones de funcionalidad y tecnología.

Scrum Diario (Daily Scrum)

El Scrum diario se define como una reunión por un espacio de tiempo de 15 minutos para el equipo de desarrollo. El Scrum diario es llevado a cabo cada día del *Sprint* en el cual se realiza la planeación del trabajo para las 24 horas siguientes. Esta práctica permite elevar los índices de colaboración y desempeño del equipo de trabajo bajo la revisión del logro basado en el último Scrum diario. El Scrum diario es utilizado para evaluar el progreso hacia el objetivo del *Sprint* y analizar las tendencias seguidas para este progreso.

Revisión de Sprint (Sprint Review)

Al finalizar cada *Sprint* se realiza los que es una revisión con la finalidad de examinar el desarrollo del producto y verificar la lista de producto en caso de ser necesario. Esta revisión se hace involucrando a todo el equipo Scrum, junto con los interesados en el proyecto, si existe la necesidad de hacer algún ajuste a la lista de producto se puede llevar a cabo en esta reunión. Este tipo de reuniones se llevan de manera informal en donde los incrementos en el desarrollo del producto son presentados.

La revisión de sprint incluye los siguientes elementos:

- Los integrantes de esta reunión son el equipo Scrum y las personas interesadas que son clave para el proyecto.
- El dueño del producto pone en perspectiva los elementos de la lista que se lograron terminar y cuales aún no.
- El equipo de desarrollo menciona los puntos importantes que surgieron durante el *Sprint*, sobre los problemas que fueron surgieron y la manera en que se solucionaron.
- Se hace una demostración del trabajo que se terminó,
- El dueño de producto habla acerca de la lista de producto en su estado actual. Proyecta objetivos probables y fechas de entrega en el tiempo basándose en el progreso obtenido hasta la fecha (si fuera necesario).

Incremento

El incremento representa la suma de todo el desarrollo de los elementos establecidos en la lista de producto que son completados durante un *Sprint* y el valor de los incrementos de todos los *Sprints* pasados. Cuando se finaliza un *Sprint* el nuevo incremento debe estar terminado, lo cual implica que puede ser que las nuevas funcionalidades pueden ser utilizadas y que cumple la definición de terminado del equipo Scrum. Un incremento es un cuerpo de trabajo inspeccionable y terminado que respalda el empirismo al final del *Sprint*. El incremento es un paso hacia una visión o meta. El incremento debe estar en condiciones de utilizarse sin importar si el Dueño de Producto decide liberarlo o no [11].

Definición de Terminado (Definition of Done)

Cuando un elemento de la lista de producto o un incremento se describe como terminado, todo el mundo debe entender lo que significa “terminado”. Aunque esto puede variar significativamente para cada equipo Scrum, los miembros del equipo deben tener un entendimiento compartido de lo que significa que el trabajo esté completado para asegurar la transparencia. Esta es la definición de terminado para el equipo Scrum y se utiliza para evaluar cuándo se ha completado el trabajo sobre el incremento de producto [12].

Capítulo 3. Desarrollo.

3.1 Descripción del Método

Para realizar este proyecto se hizo uso de la metodología Scrum, ya que al ser una metodología ágil facilita la administración de proyectos haciendo que los procesos sean adaptables. Esta metodología es también utilizada por la empresa Digipro así que se retomó cierta información para tener un punto de partida para el proyecto de automatización, a continuación, se presentar los procesos de la metodología Scrum utilizados (ver figura 6).

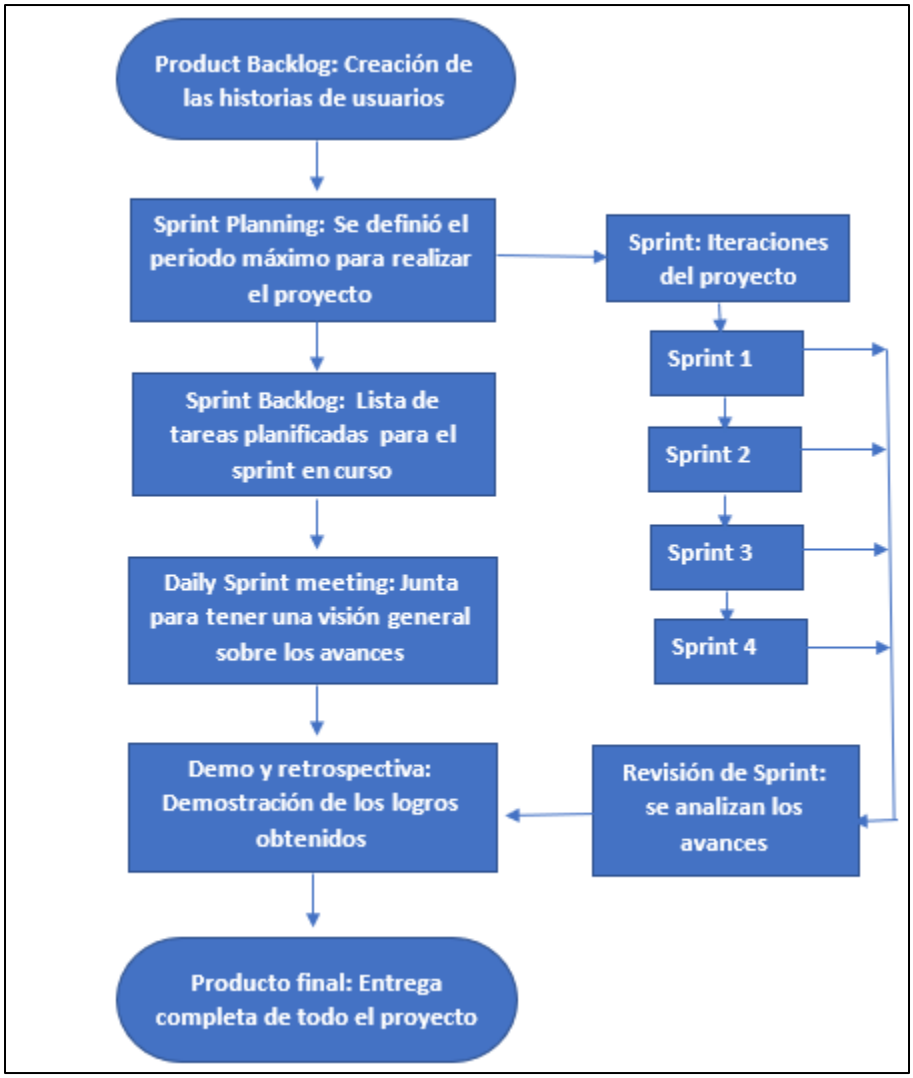

Figura 6. Metodología.

Product Backlog: Representa los requerimientos los cuales son denominados historias de usuario, aquí se ven reflejados los requisitos del cliente, en el caso del proyecto de pruebas automatizadas el cliente es el gerente de desarrollo, ya que el determina las prioridades a considerar sobre este proyecto. Para realizar el proyecto de las pruebas automatizadas se generó una lista de las tareas necesarias para su desarrollo, este punto fue muy importante ya que fueron las bases para tener en claro lo que se quería obtener, fue necesario asignar prioridades a cada una de las tareas basado en la necesidad del cliente (gerente de desarrollo) y la complejidad de cada actividad.

Cuando se generó esta lista de características no se colocaron todas las necesarias para el desarrollo de las pruebas automatizadas ya que una de las ventajas de la metodología Scrum es que conforme avanzan los *Sprints* se pueden ir añadiendo nuevos requerimientos, ya que al ir transcurriendo el tiempo surgieron nuevos detalles adicionales, en el diseño, en la funcionalidad y la verificación de actividades.

Sprint Planning: Durante esta reunión el Product owner presentó las historias del backlog en un orden de prioridades. Es aquí donde se determinó el alcance de las entregas para el Sprint en curso, se definieron las estrategias para lograr los resultados de forma satisfactoria. Antes de comenzar el proyecto fue necesario tener en claro los objetivos y metas a conseguir, detallando cada una de ellas para poderlas ajustar a los *Sprints* correspondientes y así de esta forma saber el tiempo que tomaría la ejecución de cada tarea, para poder entregar avances en tiempo y forma.

Se realizó una junta para planificar la duración del *Sprint* (Tabla 1 página 35), teniendo como objetivo identificar y comunicar cuanto del trabajo se puede entregar en determinado tiempo, así como identificar los puntos claves que serían necesarios para comenzar el proyecto.

Esta parte de la metodología Scrum se debe realizar de forma correcta ya que es aquí donde se describe la planeación para el éxito del proyecto. El *product owner* especifico claramente como tenía que ser llevado el proyecto de automatización y también se dieron sugerencias ya que es muy importante la opinión de la persona que desarrollará el proyecto, se discutió sobre el plan inicial para completar las tareas asignadas para el *Sprint* inicial. Esta fase de la metodología se dividió en dos partes.

- Alcance: De toda la lista de tareas del *product backlog* se seleccionó las que podrían generarse para el primer Sprint. Se definió cada uno de los objetivos que se querían para cada Sprint y los riesgos que podrían aparecer para anticiparse a ellos y tener una estrategia para su solución.
- Plan: Se analizó a detalle cómo se iban a entregar los elementos del *product backlog* seleccionados para cada Sprint, se identificó con más claridad cada una de las tareas a desarrollar para tener una meta establecida de las entregas.

Sprint: Esta parte de la metodología fue donde ocurrió el desarrollo del proyecto de automatización, ya que son las iteraciones de la evolución del producto, durante cada uno de estos periodos se realizó el desarrollo de las historias del *product backlog* que fueron designadas para cada *Sprint*.

Se realizaron las tareas asignadas a cada iteración y así mismo fueron surgiendo inconvenientes ya que por más que un proyecto esté bien definido siempre surgen problemas en el momento del desarrollo, para el proyecto de pruebas automatizadas no fue la excepción y todos los problemas que iban surgiendo ser platicaban en los *daily sprint* meeting para dar opiniones de cómo solucionarlos, también se lograron detectar mejorar que no estaban contempladas.

Es en esta fase de la metodología donde es donde se logró ver el proyecto de forma más clara ya que todas las tareas pasan a ser parte de la evolución. Al finalizar cada *Sprint* se fue integrado lo hecho en *Sprint* anteriores para de esta forma ir teniendo poco a poco un producto final.

Sprint Backlog: Esta sección de la metodología representó la lista de elementos del backlog del producto seleccionados para cada *Sprint,* también definió un plan para la entrega de los incrementos del producto para alcanzar los objetivos de cada *Sprint*. El backlog de *Sprint* fue un pronóstico sobre que funcionalidades estarían disponibles para cada incremento y el trabajo necesario para poder entregar dichas funcionalidades en un nuevo incremento. En este punto se hizo visible todo el trabajo que el equipo involucrado identificó como necesario para que los objetivos se cumplieran para garantizar la mejora continua.

El *Sprint backlog* fue plan que incorporó suficientes detalles para que los cambios en progreso se pudieran entender en el *daily sprint meeting*. Se hicieron mucha modificación al *Sprint*

backlog para que el proyecto se desarrollará de forma correcta. El *Sprint backlog* fue una imagen muy visible en tiempo real del trabajo realizado durante cada *Sprint*.

Daily Sprint meeting: Para tener un contexto claro de todo lo que iba surgiendo a lo largo del proyecto fue necesario tener juntas diarias de máximo 15 minutos, en dichas juntas se revisaban los avances del proyecto, se realizaban sugerencias y se exponían los riesgos o problemas encontrados. Estas juntas tenían la finalidad de mantener sincronizado a todo el equipo involucrado. Estas reuniones ayudaron a que las pruebas automatizadas se enfocaran en el camino correcto ya que había muchas tareas que no se habían contemplado en la planeación.

Demo y retrospectiva: Al finalizar cada *Sprint* se llevó acabo la presentación de cada historia de usuario que fue desarrollada, se hizo una demostración de los logros conseguidos. Posteriormente se llevó acabo la retrospectiva para analizar que se hizo bien, en que se podía mejorar y se discutió la mejora forma de perfeccionar aquello que no era óptimo.

3.2 Procedimiento y descripción de las actividades realizadas.

En este apartado se describe a detalle la construcción de las pruebas automatizadas para la parte funcional. Se presenta un alcance que se define a partir del análisis realizado, se visualiza el diseño, la construcción y la etapa de transición en el subproceso de *testing*.

La metodología adoptada para este proceso es Scrum, esta metodología es utilizada actualmente en la empresa DigiPro por lo cual fue de vital importancia que este proyecto se adecuará las normas y principios establecidos para cada uno de los proyectos de la empresa.

3.3 Alcance del prototipo

El prototipo de automatización de los casos de prueba, aplicado en un proyecto en fase de implantación, inicia con la etapa de análisis, para ver la viabilidad de las pruebas automatizadas, y así mismo poder levantar los requisitos funcionales y no funcionales que fueron considerados en el proyecto. También se revisaron los casos de pruebas que existentes analizando el alcance funcional de estos.

En la etapa de diseño se elaboró una matriz de caso de prueba de acuerdo a los requisitos especificados, permitiendo desarrollar un *happypath* para saber si las prueba desarrolladas cumplen las condiciones requeridas.

La etapa de construcción fue realizada en una pequeña parte del proyecto, debido a la magnitud de este mismo y por los tiempos. En la parte que se implementaron las pruebas fueron las más críticas a tener en cuenta ya que tenía muchas reglas de negocio a tener en consideración.

Finalmente, en la etapa de transición se modifica el proceso de *testing* para incorporar las pruebas automatizadas del sistema.

3.4 Primer Sprint del Proyecto

3.4.1 Análisis

Esta es la primera fase del proyecto correspondiente al Sprint número 1. Para entender qué es lo que se requería automatizar, lo primero fue conocer el modelo de negocio de la empresa, para poder tener el conocimiento necesario de cómo funciona el sistema para la aplicación Android. La forma de poder entender el negocio fue a través de la documentación de la empresa, también se realizaron entrevistas con los encargados del proyecto que se requiera automatizar, para tener en consideración los puntos claves en donde poner atención. La recolección de datos sobre las pruebas que anteriormente se habían realizado fueron proporcionadas por el equipo de *testing*.

Para poder continuar con el análisis se realizaron diversas tareas de capacitación, las cuales fueron asignadas por el líder del área de *testing*, ya que antes de hacer pruebas automatizadas lo primero es saber cómo realizar pruebas manuales, las tareas realizadas fueron las siguientes.

Pruebas de aceptación: Para realizar esta tarea fue necesario tener claro los conceptos de *testing* para ello se consultaron diversos libros y páginas web para tener una idea clara de lo que se iba a realizar [1,2]. Para empezar a automatizar primero se debe tener en claro cómo funcionan las diferentes pruebas de calidad. Entre las pruebas realizadas se encuentra las de aceptación, en las cuales se analizó las necesidades del usuario, requerimientos y procesos de negocio, todo esto con la finalidad de determinar si el sistema satisfacía los criterios de aceptación que permitía el cliente, así se determina si se acepta o no el sistema. Estas pruebas son fundamentales para asegurar el éxito de la implementación final de los proyectos. Es obligatorio incluir este tipo de pruebas en el proceso de calidad.

Pruebas funcionales: Este tipo de pruebas también es denominado caja negra, para realizar esta tarea fue necesario centrase en analizar los datos de entrada y salida definidos en los casos de prueba, se centró en los requerimientos funcionales de la aplicación para determinar si la aplicación realizaba todo lo que debería hacer y si lo ejecutaba correctamente.

Pruebas de regresión: Estas pruebas se llevaron a cabo durante cada lanzamiento de la aplicación para poder validar que los componentes existentes funcionarán de forma correcta y que el desarrollo que se integró se ejecutará de forma ideal con lo ya hecho en versiones pasadas. Estas pruebas se centraban en componentes ya probados, para verificar que no presentaran nuevos defectos. Este tipo de pruebas fueron fundamentales para entender por qué era necesario realizar las pruebas funcionales, ya que durante la ejecución de este tipo de pruebas de forma manual se entendieron puntos claves, uno de estos puntos es que realizar esta tarea es demasiado tedioso y está sujeto a muchos errores humanos.

Seguimiento de las incidencias encontradas hasta su resolución: Cada que se realizaban los ciclos de pruebas se encontraban incidencias en la aplicación, para ello lo primero era validar que realmente era un bug, este paso era fundamental ya que de no ser un fallo de desarrollo implicaba un retraso en los tiempos. Cuando se tenía identificado que realmente era un bug se reportaba la incidencia al equipo de desarrollo en la plataforma Azure Devops y se le asignaba una prioridad para saber si era de carácter urgente o no, posteriormente se realizaba el seguimiento hasta que el equipo de desarrollo liberara una nueva versión con la incidencia solucionada, para ello se debía comprobar que realmente se había solucionado el problema de lo contrario se rechazaba la nueva liberación.

Cierre de historias de usuario y tareas: Durante cada *Sprint* se generaban diversas tareas o historias de usuario y estas debían cerrarse al finalizar el Sprint. Para ello era indispensable tener la documentación necesaria, para de esta forma poder hacer las pruebas de calidad para validar que los nuevos componentes cumplían con lo especificado. Realizar esta tarea es primordial ya que es aquí donde se hace la integración con el *Sprint* pasado.

Creación de matrices de caso de pruebas: Fue necesario entender el concepto de casos de prueba debido a que son estos los que se requerían automatizar. Para ello se realizaron diversos casos de prueba de las nuevas funcionalidades que se iban desarrollando.

Documentación: Se realizó documentación sobre los nuevos componentes desarrollados.

Debido a que existen muchas herramientas para automatizar se realizó un estudio para saber cuál era la mejor solución para el proyecto, para ello se consideraron 3 herramientas las cuales fueron:

- Espresso
- XCUITests
- Appium

Cada una de estas herramientas tiene sus características propias, con ventajas y desventajas. Appium es la que mejores características tiene, entre ella se puede mencionar que es un software open source, los *scripts* se pueden desarrollar en diversos lenguajes de programación, lo cual facilita mucho la integración al lenguaje que mejor se adapte a las necesidades, otra de sus grandes ventajas es la velocidad en que se realizan las pruebas y que además tiene su propio sistema para identificar los elementos de la aplicación.

Para implementación de Appium se requirió de muchas horas de investigación para la parte documental de la aplicación, para saber la forma en que funcionan, su integración e implementación en los proyectos, se realizaron pequeñas pruebas de forma sencilla, para ver cómo se adaptaba a la aplicación de la empresa.

3.5 Segundo Sprint del proyecto

Durante la segunda iteración del proyecto fue donde se empezó a ser uso de la herramienta seleccionada para las pruebas automatizadas Appium, para ello se tuvo que considerar todas las herramientas necesarias que van de la mano con dicha aplicación. Se tuvo de que diseñar el escenario para que todo funcionara de forma correcta.

3.5.1 Diseño

A continuación, se presenta el diseño de los *scripts* manuales, diagramas y la lógica que se generaron para la solución.

El primer diseño estuvo enfocado a las pruebas funcionales referentes a las cajas de texto, aquí es donde se utilizó el inspector de elementos de Appium para conocer el id y poder llamarlos en el *script*. Estas pruebas fueron necesarias para tomar en cuenta los tiempos

que se tarda en llenar los campos e identificar si los elementos habían sido mapeados correctamente.

El segundo diseño, con el enfoque centrado en los datos, permitió abordar limitaciones importantes encontradas en la validación de factibilidad. Una de ellas referido al momento de la ejecución de una nueva prueba, era necesario editar los *scripts* automatizados para modificar los datos de la prueba.

Una vez identificados los puntos claves fue necesario instalar todas las herramientas necesarias para ejecutar las pruebas automatizadas.

La herramienta que se utilizó se llama Appium Desktop la cual es una versión que se puede encontrar en su página oficial, esta herramienta requiere de diversos componentes para poder ejecutarse de manera correcta. Es necesario instalar todos los componentes requeridos así también como las librerías que se necesitan para el proyecto.

Para poder validar que todas las herramientas que se necesitan para el funcionamiento de Appium estuvieran correctamente integradas, se cuenta con un comando que se ejecuta desde el “CMD” para poder saber el estado de la aplicación. El comando utilizado es “Appium-doctor” el cual devuelve información de los componentes necesarios y marcando con rojo aquello que hacen falta (ver figura 9 en la página 39).

Librerías utilizadas para las pruebas automatizadas.

Se utilizaron distintas librerías (ver figura 10 en la página 40) las cuales sirven para hacer las interacciones necesarias entre Appium y la aplicación a automatizar. Es fundamental que se tenga un gestor de proyectos como lo es Maven ya que permite tener el control de las versiones de las librerías utilizadas, para la tapa inicial de este proyecto no se consideró dicho requerimiento, pero a lo largo del desarrollo se vio la necesidad de poder migrar al gestor mencionado.

Cuando se escribe s*cripts* de prueba con Appium, generalmente implica iniciar una aplicación y luego realizar algunas acciones en ella. Para la aplicación que se requiere probar en este proyecto, así como sus casos de pruebas, para lograr probar esto primero se necesitan conocer los elementos que interactúan en la aplicación

Después de que todo el entorno de Appium está instalado correctamente se puede empezar a generar los *scripts*, es necesario que se tengan todos los identificadores que estarán en juego para la prueba, texto, listas, checkbox, firma electrónica, botones, etc. Para ello Appium cuenta con un potente buscador de elemento con el cual se puede identificar los elementos claves.

Para que este inspector funcione se debe enviar un Json con la informar necesaria de la aplicación que se requiera examinar, teniendo dos elementos clave:

- appPackage: Es el nombre técnico de la aplicación
- appActivity: Se refiere a las diferentes funcionalidades que proporciona la aplicación

Esto es necesario para poder lanzar la aplicación en el dispositivo para sacar el nombre de los elementos (ver figura 11 en la página 40).

Cuando se lanza la aplicación en el inspector, se muestra el interfaz de la misma y en ella se encuentra una pantalla espejo de la aplicación que se requiere inspeccionar. Cada movimiento que se realice en un dispositivo real se ve reflejado aquí. En esta herramienta que proporciona Appium se pueden identificar todos los elementos con solo seleccionarlos en la pantalla, ya que envía toda la información necesaria referente a los elementos. Cada elemento se puede identificar por:

- Accessibility ID
- Class name
- ID
- Name of element
- XPath

En el caso de la aplicación de la empresa la identificación de la mayoría de los elementos es por un XPath debido a que los elementos provienen desde una página web, esto con la finalidad de que sea una aplicación multiplataforma.

Una vez que los elementos necesarios fueron identificados (ver figura 12 en la página 41) se procedió a desarrollar los *scripts* de la aplicación, para ello se utilizó java como lenguaje de programación con IntelliJ IDEA, los primeros *scripts* desarrollados fueron casos de

prueba sencillos esto con la finalidad de ver como se adaptaban las pruebas automatizadas a la aplicación en desarrollo.

3.6 Tercer Sprint del proyecto

Con la finalidad de robustecer las pruebas automatizadas fue necesario crear métodos para funcionalidades específicas de la aplicación, ya que para poder interactuar con ellos fue necesario invertir tiempo en desarrollar una solución que pudiera ajustarse a cualquier proyecto a futuro. Se crearon diferentes métodos para dar solución a esta tarea.

Primer método

- Firma digital: Este método realiza la firma digital de manera automática, ya que se le envían coordenadas en la pantalla para formar la firma (Ver figura 13 en la página 41).
- Segundo método: Engloba 3 elementos ya que funciona de forma similar, y dicho método se construyó para que Appium pudiera detectar la funcionalidad para:
    - Captura de pantalla
    - Captura de audio
    - Captura de video
- Scroll: Este método es de los más importante ya que la mayoría de las aplicaciones funcionan con scroll, y para poder automatizar una aplicación es necesario que todos elementos estén visibles en la pantalla, esto quiere decir que si tengo 6 elementos en la pantalla pero el 6 no se logra ver por las dimensiones del dispositivo, el *script* fallará ya que no encontrará el elemento 6, pero con este método el *script* hará scroll hasta que el elemento aparezca y se pueden ejecutar las acciones relacionadas con dicho elemento. La gran ventaja con este método es que se adapta automáticamente a cada pantalla sin importar las dimensiones.
- Pulsaciones largas para los botones: Dentro de la aplicación existen botones que debe ser pulsados determinado tiempo, para poder automatizar esta era tediosa ya que se requieran de muchas acciones para lograr el objetivo, la solución fue crea un solo método para utilizar menos líneas de código, ya que este elemento aparece muchas veces en la aplicación.

- Reglas de negocio: La aplicación que se requiera automatizar tenía muchas reglas de negocios, las cuales son muy importantes por lo cual era un punto muy importante a tener en cuenta para la automatización.
- Detección de elementos requeridos dentro de la aplicación: Con la creación de un método se pudo detectar todos los elementos requeridos dentro de la aplicación, este método lo que hacía era mandar un txt con la información de los elementos y si los elementos iban vacíos marcaba una excepción por no cumplir el funcionamiento requerido.
- Detección de los elementos ocultos dentro de la aplicación: La aplicación tiene elementos que no son visibles para el usuario pero que se utilizan para ejecutar acciones que son necesarias para el correcto funcionamiento, el método desarrollado permitía obtener la cantidad de elementos visibles esperados y comparar si existía algún que no estuviera contemplado.
- Detectar el prellenado de los elementos: Para este tipo de acción fue necesario poder validar un elemento txt que fue llenado con información, enviará dicha información a otro campo texto (Ver figura 14 en la página 42).

3.8 Cuarto Sprint

Las pruebas automatizadas pueden generar más de 100 casos de uso al día, pero fue necesario que la información que se generar en la aplicación fuera distinta, es decir que hubiera al menos 100 nombres, apellidos, ciudades, etc., por lo tanto, que por cada ciclo la información almacenada fuera diferente, esto con la finalidad de que la información se pudiera analizar en la base de datos y que las condiciones fueran apegadas a un ambiente productivo. Para logra esto se consultó una página "https://mockaroo.com/" la cual genera datos aleatorios (ver figura 15 en la página 42), es una herramienta muy potente ya que pueden crear cualquier tipo de dato y gracias a su api se puede incluir en el script de automatización para hacer peticiones a su servicio y que este mande un archivo Json con los datos necesarios.

3.9 Scrum diario

Durante el desarrollo de proyecto, una vez definido el *Sprint* inicial era necesario generar juntas diarias llamadas Daily Stand up, todo esto durante cada iteración. Esta parte de la

metodología Scrum es muy importante ya que durante la reunión se puede tener claro el estado del proyecto. Su objetivo es que el equipo se mantenga actualizado unos a otros sobre los trabajos realizados, y se tenga informado a los líderes sobre los progresos, así mismo poder identificar los riesgos o problemas que se encuentren o los que se prevén encontrar, para poder diseñar un plan el cual no afecte con el *Sprint* en curso.

Las reuniones diarias no suelen tardar más de 15 minutos, son reuniones muy breves que buscar agilizar los proceso y no ocupar tiempo valioso, debe ser centra en sus objetivos. Es muy importante siendo casi obligatorio que todo el equipo involucrado esté presente.

3.10 Revisión de Sprint

Al finalizar cada iteración, se tenía que analizar detalladamente los objetivos del *Sprint* que se cerró con la finalidad de poder ver que, si se logró y que no, para de esta forma aquellas tareas que no fueron cumplidas pasen al siguiente *Sprint* con la finalidad de que no se pierdan y se logren cumplir.

En esta fase se presentaron los trabajos que fueron realizados, los alcances obtenidos, para poder saber el avance general del proyecto. Este punto es clave ya que se debe cumplir con los objetivos marcados al principio del Sprint para no retrasar el proyecto.

3.11 Retrospectiva del Sprint

Después de cada *Sprint*, se llevó a cabo una retrospectiva, en la cual todos los miembros del equipo dejan sus impresiones sobre el *Sprint* recién cursado. El propósito de la retrospectiva es realizar una mejora continua del proceso.

Capítulo 4. Resultados y conclusiones.

4.1 Resultados

Para poder validar el alcance que tuvo el proyecto fue necesario la participación de todo el equipo de *testing* para poder verificar que las pruebas automatizada realmente cumplía con el objetivo, se analizó la planeación del *Sprint* (Tabla 1 página 35). También se presentó el proyecto ante la junta directiva de la empresa con la finalidad de que se autorizara un nuevo departamento para el área de calidad, demostrando los beneficios obtenidos con las pruebas automatizadas.

El objetivo de la investigación es introducir pruebas automatizadas en los procesos de calidad de la empresa, para así de esta forma, agilizar los tiempos que se requieren para las pruebas de regresión y pruebas funcionales. De esta manera la carga de trabajo del área se redujo y se logró reducir las incidencias en el sistema. Se logró una etapa de transición, modificando el proceso de *testing* de forma manual a automatizado. Con lo anterior da paso a poder gestionar una nueva área en el departamento de *testing* enfocada a pruebas automatizadas.

El proyecto en el que se aplicaron las pruebas automatizadas se logró una mejora de calidad y se pudieron realizar entregas tempranas del sistema, esto redujo el número de incidencias ya que dicho proyecto tenía muchas complicaciones y retrasos. En el mes de noviembre y diciembre se logró ver los resultados ya que fueron los meses en que las incidencias disminuyeron, y donde la carga de trabajo bajo.

En la figura 7 se puede ver una comparativa de las diferentes plataformas, la cual demuestra que la plataforma Android fue la que mejor resultados han tenido a lo largo de estos meses, ya que sus incidencias disminuyeron mucho. Esto no quiere decir que las pruebas automatizadas reducen automáticamente muchas incidencias, pero gracias a que el tiempo que se utiliza para realizar ciertas pruebas que tomaban muchas horas-hombre ahora sé hacer de forma automatizada, esto permite que el equipo de testing tenga más tiempo libre para realizar otro tipo de pruebas manuales, las cuales antes no eran posible debido a la carga de trabajo. Esto quiere decir que los resultados logrados son gracias al compromiso del área de calidad.

Primera Fase																
Sprint	Agosto				Septiembre				Octubre				Noviembre			
	07-9	12-16	19-23	26-30	02-6	9-13	16-20	23-30	1-5	6-12	13-19	20-31	1-8	11-15	18-22	25-29
Sprint 1: • Familiarización con el modelo de negocio de la empresa. • Estudiar los requerimientos funcionales de la aplicación a automatizar. • Investigar las diversas herramientas de automatización.																
Sprint 2 • Crear el escenario necesario para las pruebas automatizadas, herramientas, liberarías, programas a utilizar. • Crear un caso de prueba funcional sencillo, para un ejemplo de automatización. • Crear el script en java para el caso de prueba funcional especificado el cual																

demuestre como implementar la automatización en el proyecto deseado.																	
Sprint 3 Crear métodos en java para poder automatizar los siguientes elementos de la aplicación: ➢ Firma Digital ➢ Captura de pantalla ➢ Captura de audio ➢ Captura de video ➢ Scroll ➢ Pulsaciones largas para los botones ➢ Detección de elementos requeridos dentro de la aplicación ➢ Detección de los elementos ocultos dentro de la aplicación ➢ Detectar el prellenado de los elementos																	

Sprint 4 • Implementar un archivo Json en las pruebas automatizada para generar datos aleatorios, los cuales puedan ser analizados en la base de datos.																

Tabla 1.Planeación de Sprints

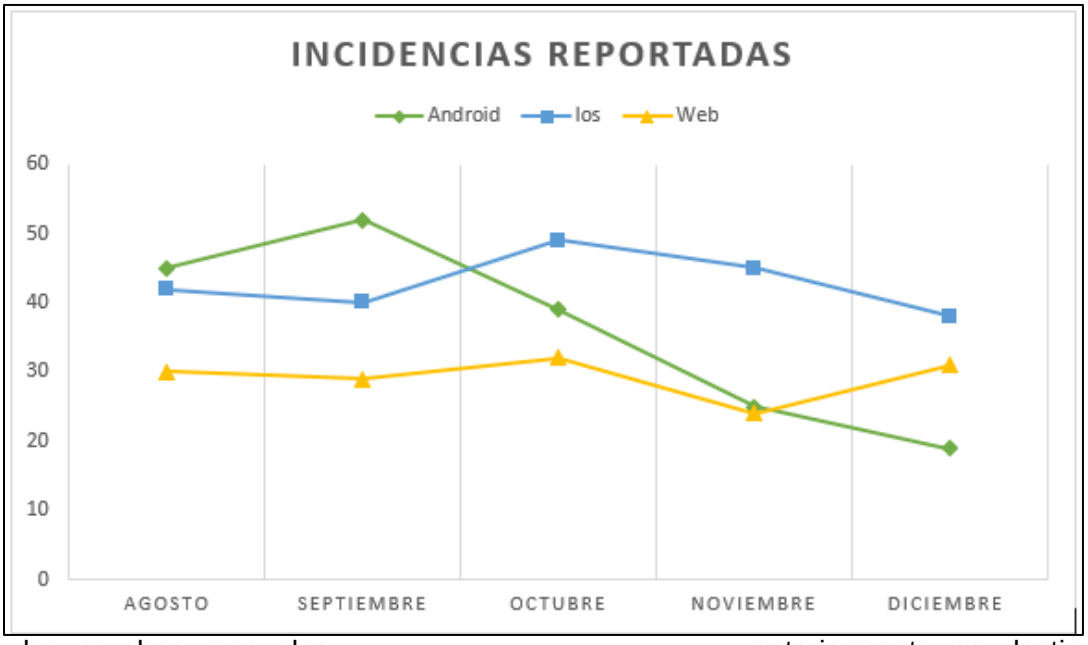

Figura 7. Gráfica de incidencias.

Con las pruebas manuales anteriormente se destinaba alrededor de más del 50% del tiempo por cada nuevo requerimiento, esto implicaba que mensualmente se utilizarán más de 100 horas hombre de las 160 horas hombre destinadas al desarrollo general de las pruebas, esto dejaba poco tiempo para realizar otro tipo de pruebas a la aplicación.

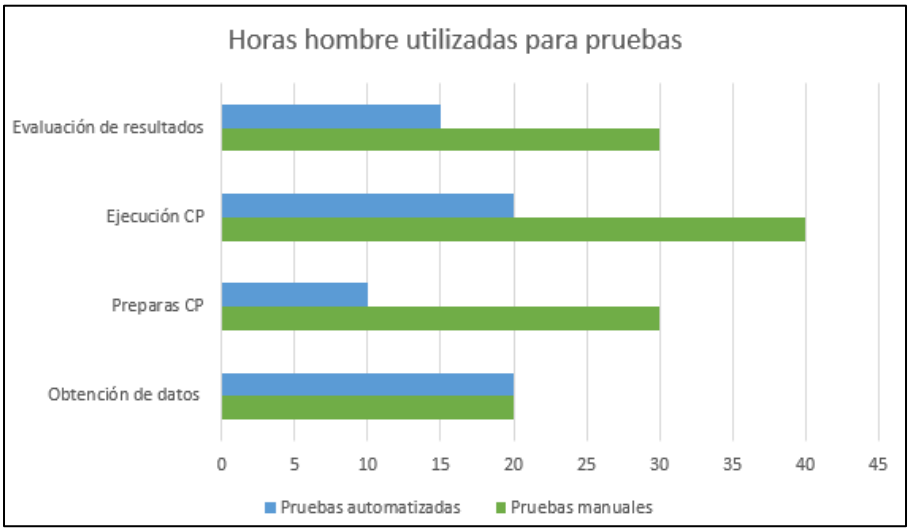

Figura 8. Horas hombre utilizada para pruebas

Se logró reducir casi a la mitad las HH utilizadas para pruebas de calidad, antes se utilizaba personal de otras áreas para poder cubrir todas las pruebas necesarias, se llegaba a requerir de 3 a 4 usuarios para la certificación de las nuevas funcionalidades desarrolladas

y para realizar las pruebas de regresión, actualmente solo se utiliza 1 persona para realizar estas mismas funciones, ya que las pruebas de regresión se hace con la herramienta de automatización invirtiendo tiempo solo en la etapa final de evaluación de resultados.

También se pudieron ejecutar más casos de pruebas que anteriormente era imposible lograr, estos nuevos casos de prueba lograron robustecer más las pruebas para poder encontrar mayor número de incidencias.

Con respecto a las herramientas utilizadas se logró una correcta adaptación ya que con Appium doctor se logró identificar si todo el entorno estaba perfectamente integrado.

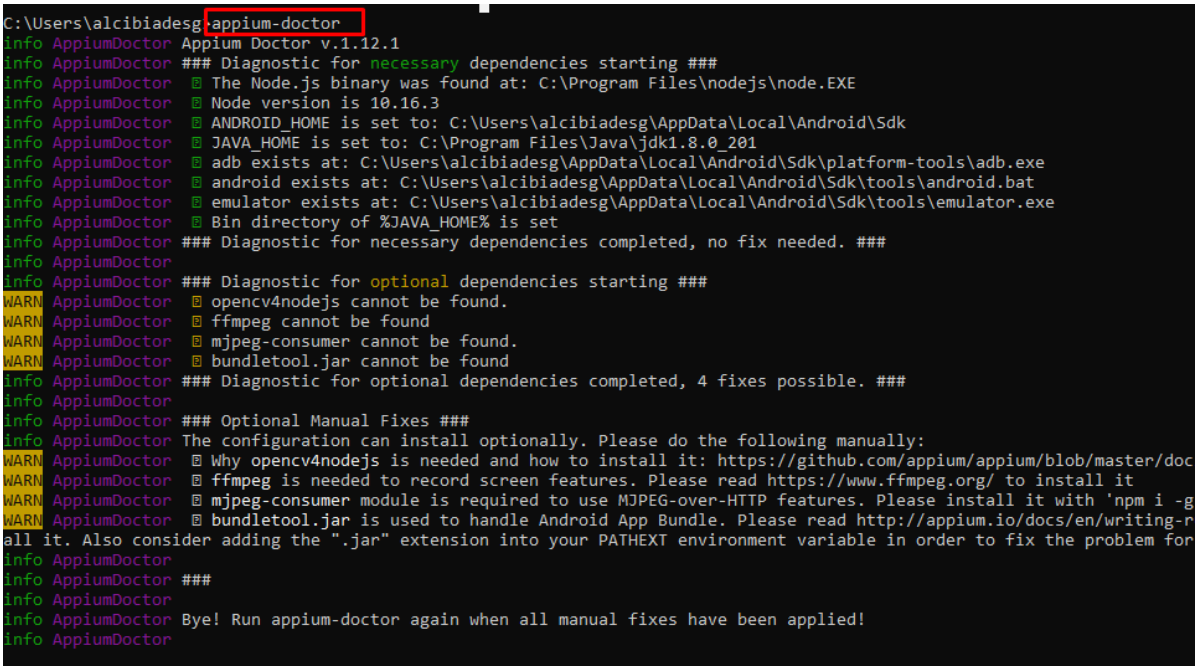

Figura 9. Appium Doctor

También se pudieron integrar todas las librerías necesarias para las pruebas automatizadas

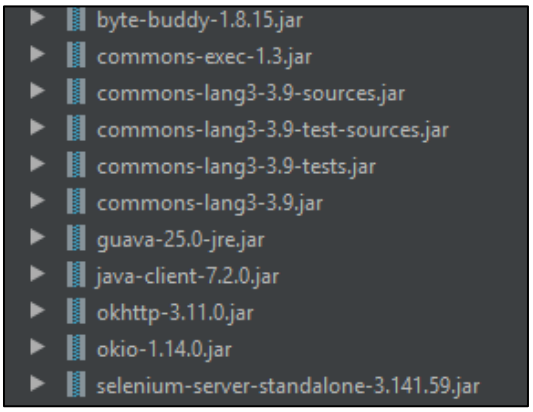

Figura 10. Librerías utilizadas

Se pudieron obtener los parámetros necesarios para que Appium pudiera conectar con la aplicación que se requería automatizar.

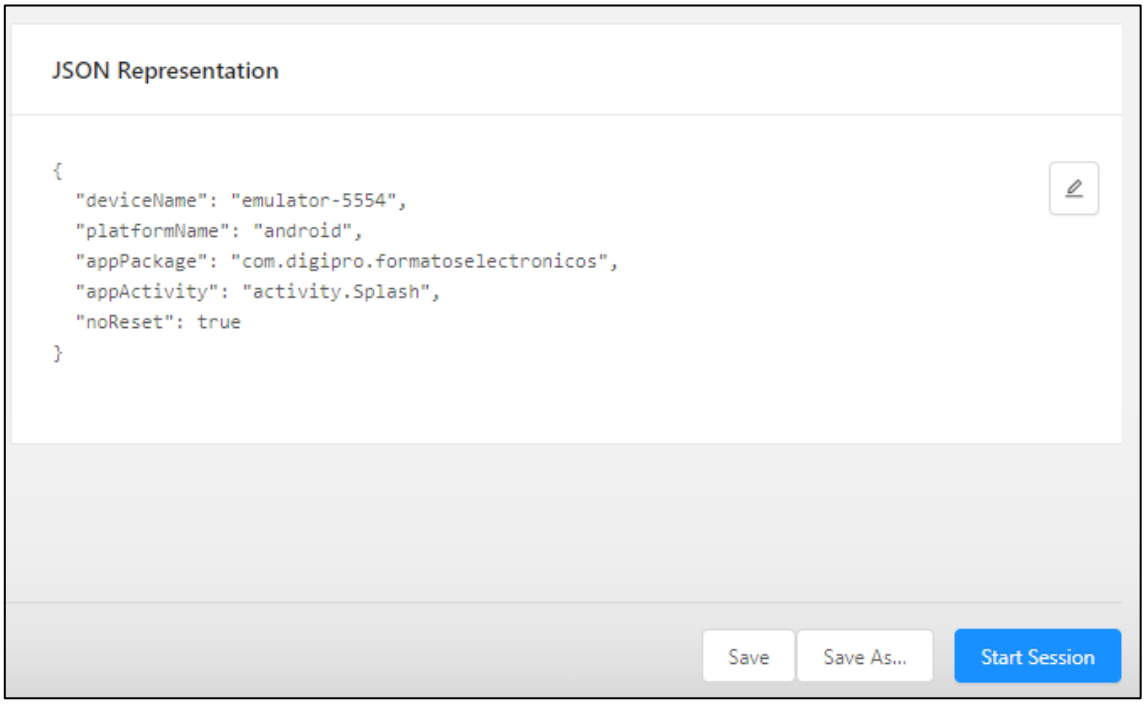

Figura 11. Parámetros para correr la aplicación con Appium

La obtención de los atributos de los elementos de la aplicación se logró realizar de manera satisfactoria haciendo uso del inspector de Appium, para posteriormente poder llamarlos desde el *script* automatizado.

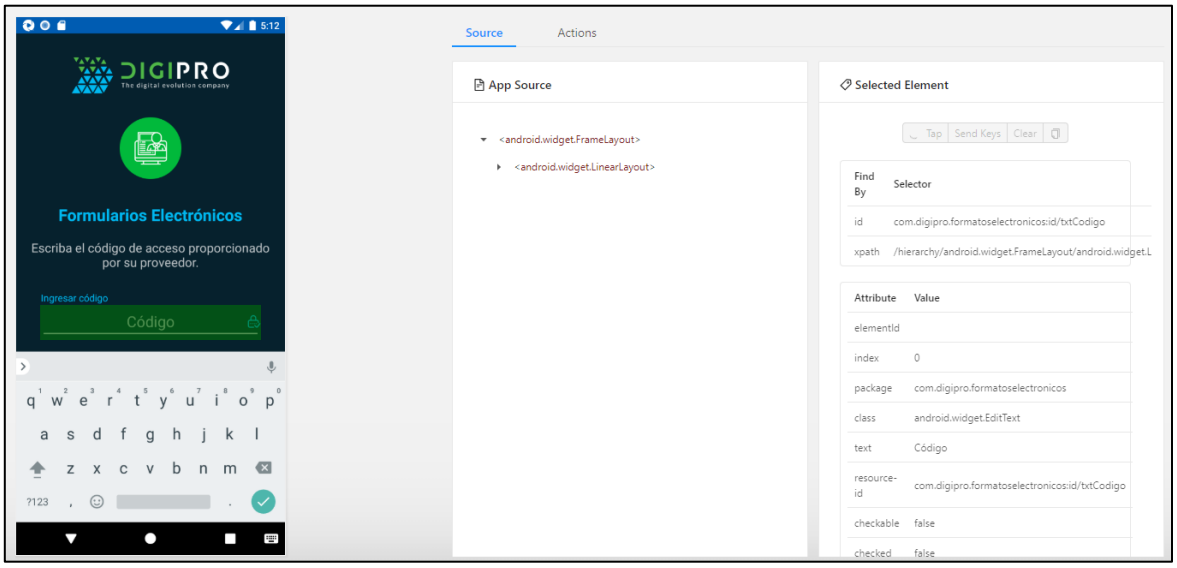

Figura 12. Inspector de elementos de Appium

Para el elemento firma, se pudo realizar de forma automatizada, esto gracias al método desarrollado el cual se adaptaba a las diferentes pantallas para poder enviar las coordenadas de la firma.

Figura 13. Elemento firma.

Para los elementos que realizaban la funcionalidad de prellenado, es decir mandar los mismos datos almacenados de un elemento a otro, para ello se logró que el script pudiera detectar si esta funcionalidad se estaba realizando de forma correcta.

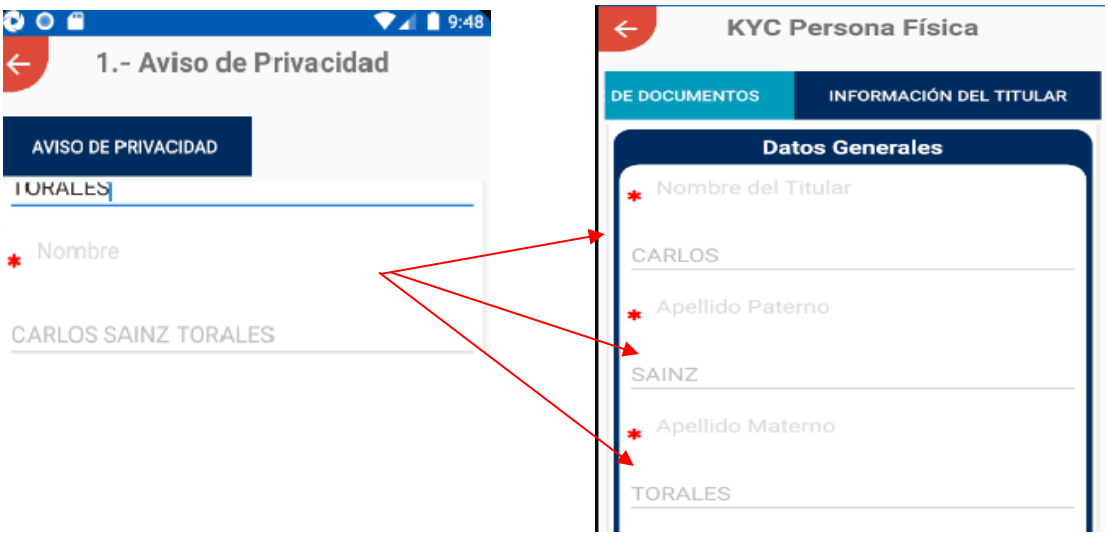

Figura 14. Prellenado de los elementos

Para poder mandar a la aplicación datos aleatorios se logró implementar un API en el script que generaba datos dependiendo de los parámetros asignados, esta información llegaba en forma de Json por parte del api utilizada, la integración de este api permitió poder generar información más realista que sirvió para que el área de base de datos pudiera analizar todas las peticiones que se realizaban.

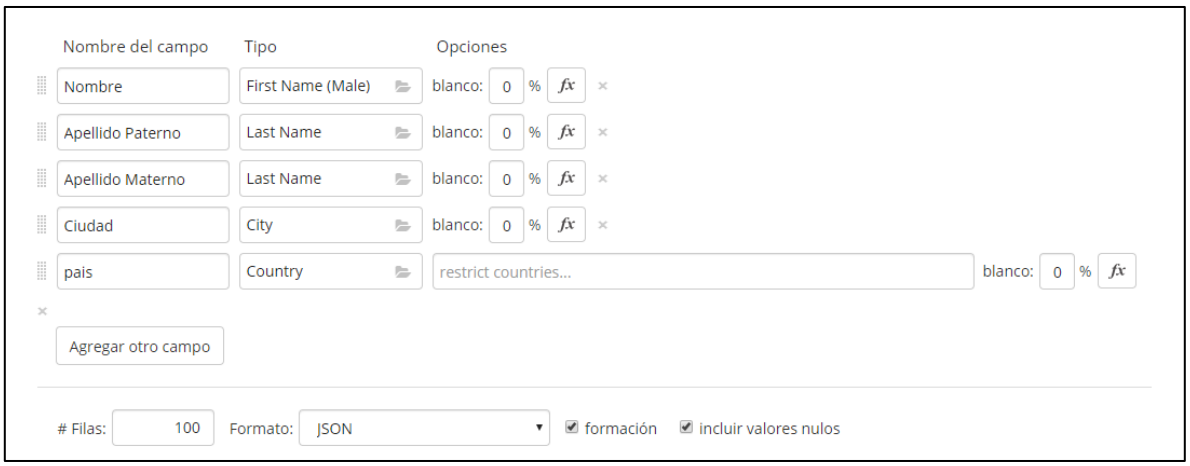

Figura 15. api para generar datos aleatorios

Las pruebas automatizadas pudieron cumplir su propósito, realizando las pruebas necesarias reduciendo tiempos e incidencias, se logró que el *script* llenara de forma automatizada los elementos básicos, así como los más complejos.

Figura 17. Elementos básicos de la aplicación

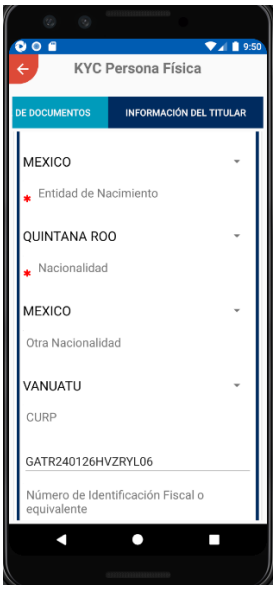

Figura 16. Elemento complejo de la aplicación

El resultado de las pruebas automatizadas se logra ver en las mejoras del área de calidad, pero se puede decir que se logró generar diversos *Scripts* para pruebas automatizadas, cabe mencionar que para poder ejecutar estos *Scripts* es necesario contar con Intelliji IDE ya que no se cuenta con una interfaz gráfica ya que las pruebas automatizadas son desarrolladas directamente desde código, pero esto no quiere decir que seas difícil de interpretar con alguien con conocimientos básicos ya que gracias a la separación de código mediante el patrón de diseño "*Page Object Model*" se logró una mejor lectura del código.

Un mejor enfoque para el mantenimiento de *scripts* fue crear un archivo de clase separado que encuentre elementos de la aplicación, los complete o los verifique. Esta clase se puede reutilizar en todos los scripts que utilizan ese elemento. En el futuro, si hay un cambio en los elementos de la aplicación solo se tendrá que hacer cambios en un solo archivo de clase y no en 10 scripts diferentes.

El *Page Object Model* es un patrón de diseño de objetos, donde las diferentes partes de la aplicación se representan como clases y los diversos elementos de la página se definen

como variables en la clase. Todas las interacciones de usuario posibles se pueden implementar como métodos en la clase.

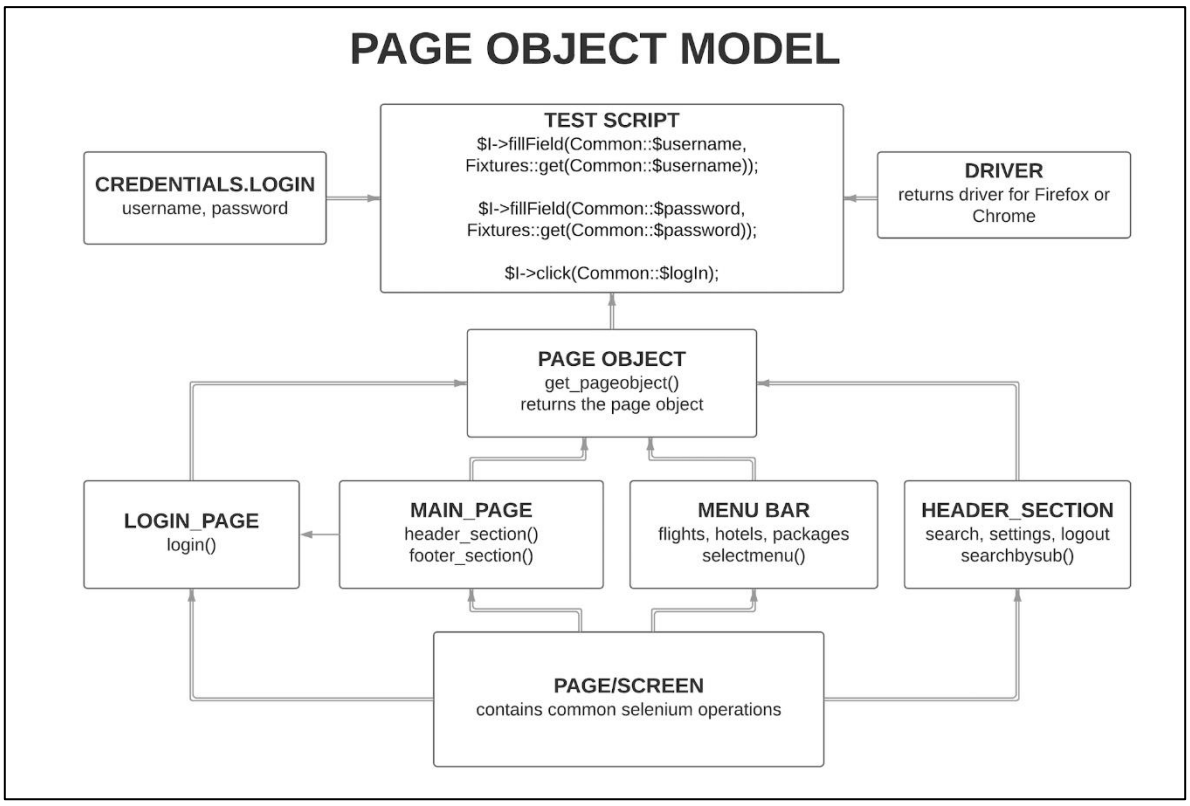

Figura 18 Page Objet Model

4.2 Evaluación del producto.

Como producto final se obtuvo un proyecto java con todos los *Script* de casos de prueba más relevantes, cada test de caso de prueba fue separado por clases diferentes y los elementos que manipulan la aplicación separados en una clase distinta, para tener un mejor control sobre ellos. Cada uno de los métodos utilizados fueron organizados en otra clase, con la finalidad de que en la clase de los test únicamente estuvieran desarrollados los casos de prueba, todo esto se realizó con la finalidad de que estuviera mejor estructurado el código de la automatización, de esta forma el manteamiento posterior de los scripts será más rápido y sencillo

Al finalizar el proyecto se logró llevar a cabo las pruebas de regresión de forma más constante, de esta manera al ejecutar las pruebas se logran ver los resultados de los test y se identifica cuales casos de prueba habían fallado y cuales resultaban exitosos, realizar estas pruebas de forma automatizada llevaban muy poco tiempo lo cual permitió elaborar test de al menos más de 100 casos de prueba robusteciendo las pruebas de regresión y las funcionales.

Como dato importase debe mencionar que este proyecto abre las puertas a una nueva área en el departamento de TI, ya que las pruebas automatizadas se deben adaptar a cada proyecto nuevo a desarrollar, pero el proyecto “Diseño de la estrategia de automatización e implementación de herramientas para realizar pruebas de aplicaciones del sistema operativo móvil Android” deja las bases y la documentación necesaria que permitirán seguir desarrollando pruebas automatizadas e integrando nuevas herramientas que ayuden a conseguir mejores resultados.

Conclusión

La realización de este proyecto permitió la depuración de los casos de pruebas existentes, ya que se adaptaron una forma más completa, y se eliminaron todos los CP que son repetidos y que no ayudaban a mejorar la calidad del producto. Esto permitió al equipo de testing mejorar sus prácticas de calidad ampliando los conocimientos a las demás plataformas.

Respecto a la metodología utilizada permitió una correcta adaptación al proyecto de la empresa, permitiendo una mejora en los tiempos de entrega, al entregar Script conforme se iban terminado las iteraciones, esto hizo posible ejecutar procesos combinados ejecutando los scripts de automatización y las pruebas manuales.

Sin embargo, esta mejora del tiempo del proceso de prueba se vio afectada por los constantes cambios al proyecto que se automatizo, al requerir varias horas de programación para transformar un script de prueba en un script automatizado y cualquier cambio en el sistema utilizado en el prototipo como piloto requería también varias horas para realizar las modificaciones en los scripts ya automatizados.

Las pruebas automatizadas son una gran herramienta para mejorar la calidad de los sistemas, pero se debe tener en claro que no es una solución inmediata ya que requiere de mucho tiempo de inversión, esto hace que se genera más carga de trabajo al área, entonces si el objetivo es reducir los tiempos inmediatamente será una mala práctica, ya que los beneficios se logran ver a largo plazo.

Al desarrollar las pruebas automatizadas quedaron en evidencia los grandes aportes y beneficios que se obtienen, así como los puntos clave que se tienen que tener en consideración:

Es importante definir correctamente el propósito de iniciar este tipo de pruebas, ya que se debe tener en claro que no todo en un sistema se puede automatizar, ya que se perderá más tiempo sin obtener beneficios al no hacer un estudio correcto.

Para que las pruebas automatizadas tengan éxito se deben seguir las siguientes recomendaciones, ya que gracias a ellas se logró concluir el proyecto de forma satisfactoria

- Se debe analizar en qué parte del ciclo de desarrollo es más conveniente comenzar a automatizar, es el primer paso para desarrollar una estrategia de automatización.
- Se debe desarrollar una estrategia adecuada con todo el equipo involucrado. Al igual que con todos los esfuerzos de prueba se debe construir una estrategia o plan de pruebas para implementarla.

- Cada proyecto desarrollado es diferente, por ello se debe considerar que herramientas de automatización se adecua mejor al proyecto, y también elegir adecuadamente en que lenguaje se van escribir los scripts. Se deben tratar los procesos de automatización como los demás proyectos.
- Para lograr los beneficios esperados hay que considerar el esfuerzo de la automatización, teniendo en cuenta que requiere de tiempo y recursos. Esto implica que regularmente las primeras versiones no ofrecen los beneficios esperados.

Este proyecto se cumplió de manera satisfactoria, dando al área de QA mejores herramientas para poder realizar pruebas de calidad, el objetivo principal de diseñar una estrategia de automatización para el sistema operativo Android cumplió con lo esperado ya que la estrategia diseñada y las herramientas utilizadas proporcionaron al área una mejor optimización en los tiempos y coste de los proyectos.

Se logró tener *scripts* para ejecutar las pruebas regresión y funcionales de manera automatizada, es decir por cada caso de prueba que se realizaba manualmente ahora se realiza con las estrategias diseñadas en el proyecto. Es importante mencionar que este proyecto no se trata de diseñar un programa o aplicación, ya que el *testing* automatizado lo que hace es utilizar diferentes herramientas para hacer una integración de todas ellas y así lograr los objetivos; esta afirmación se refiere a que los *scripts* fueron diseñados en código java utilizando el entorno de desarrollo intellij IDEA, junto con Appium para así lograr el objetivo. Con esta nueva estrategia ahora se puede simular el proceso que antes se hacía de forma manual.

En conclusión, poder desarrollar este proyecto permitió adquirir conocimiento, así como aplicar las habilidades que ya se tenían. Este proyecto dejó claro que una buena práctica de pruebas automatizadas genera muchos beneficios a las empresas, pero para poder implementar este tipo de herramientas es necesario contar con el apoyo de los directivos, y de los equipos involucrados ya que esto facilita el trabajo desarrollado.

Fuentes de información

[1] Juan Ferrera Martínez (2016). “Desarrollo de interfaces”

[2] Pantaleo Guillermo (2016) “Calidad en el desarrollo de software - 2ª Edición”

[3] Stephan Goericke (2019). “The Future of Software Quality Assurance”

[4] Beizer B. (1990) “Software testing techniques- 2ª Edición

[5] Kaner C., Falk J. Nguyen H. (1999). “Testing Computer Software- 2ª Edición”

[6] Rodríguez (2 de abril de 2016). Recuperado de:” https://bit.ly/30DOwHy”

[7] Myers G. (2004) “The art of software testing- 2ª Edición”

[8] IEEE Standard Glossary of Software Engineering Terminology Institute of Electrical and Electronics Engineers (1990).

[9] Federico Toledo Rodríguez (2014). “Introducción a las pruebas de sistemas de información 1ª Edición”

[10] MarkFewster& Dorothy Graham (2013). “Software Test Automation, Effective use of test execution tools”

[11]Altman, H. (2016). SCRUM: La Primera MetodologíaÁgil para Gestionar el Desarrollo de Productos Paso a Paso. Sevilla, España: Kindle Direct Publishing).

[12] Alaimo, M., & Salías, M. (2015). Proyectos Ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos.

[13] Lasa Gómez, C., &AlvarezGarcía, A. D. (2017). Métodos Ágiles: Scrum, Kanban, Lean.