UNIVERSIDAD AUTÓNOMA DEL BENI
"JOSÉ BALLIVIÁN"
ALSIE CONSULTORES PEDAGÓGICOS

FRAMEWORK DE AUTOMATIZACIÓN DE PRUEBAS PARA APLICACIONES WEB MULTIPLATAFORMA

Tesis de Grado para optar el Título de:

Magister en Ingeniería de Software

Postulante

Mauricio Giovanny Viscarra Rivera

Tutor

PDh. Luis Roberto Pérez Ríos

Marzo de 2020

Cochabamba - Bolivia

INDICE

INTRODUCCION 8

PROBLEMA CIENTÍFICO 9
OBJETO DE ESTUDIO 10
OBJETIVO GENERAL 10
OBJETIVOS ESPECÍFICOS 10
CAMPO DE ACCIÓN 10
HIPÓTESIS 11
APORTE TEÓRICO 11
SIGNIFICANCIA PRACTICA 11
MÉTODOS 11

CAPITULO I 12

1.1 METODOLOGÍAS EMPLEADAS PARA LA CARACTERIZACIÓN 12
1.2 DIAGNOSTICO DE LA AUTOMATIZACIÓN DE PRUEBAS DE SOFTWARE 13
1.2.1 ESTADO ACTUAL DE LA AUTOMATIZACIÓN DE PRUEBAS EN PROYECTOS DE SOFTWARE CON SOPORTE MULTIPLATAFORMA 13
1.2.2 IMPLEMENTACIÓN DE HERRAMIENTAS DE AUTOMATIZACIÓN EN PROYECTOS DE SOFTWARE CON SOPORTE MULTIPLATAFORMA 21
1.2.3 APPIUM VS SELENIUM 28
1.3 CONCLUSIONES 31

CAPITULO II 33

2.1 CALIDAD DE SOFTWARE 33
2.1.1 DEFINICIÓN 33
2.1.2 ATRIBUTOS CLAVE DE CALIDAD DE SOFTWARE 34
2.1.3 ASEGURAMIENTO DE CALIDAD DE SOFTWARE 35
2.1.4 ERRORES DE SOFTWARE 36
2.2 PRUEBAS DE SOFTWARE 36
2.2.1 NIVELES DE PRUEBAS 37
2.2.2 TIPOS DE PRUEBAS 38
2.3 AUTOMATIZACIÓN DE PRUEBAS DE SOFTWARE 38
2.3.1 QUE SON LAS PRUEBAS AUTOMATIZADAS DE SOFTWARE 39
2.3.2 FRAMEWORK DE AUTOMATIZACIÓN DE PRUEBAS 40
2.3.3 ARQUITECTURA GENERAL DE FRAMEWORK DE AUTOMATIZACIÓN DE PRUEBAS 40

CAPITULO III 45

3.1 ENFOQUE PARA EL MODELADO TEÓRICO 45
3.2 DEFINICIÓN DE COMPONENTES 45
3.2.1 CALIDAD DE SOFTWARE 46
3.2.2 NIVELES DE PRUEBA 46
3.2.3 TIPOS DE PRUEBA 46
3.2.4 TÉCNICAS DE PRUEBAS 47
3.2.5 AUTOMATIZACIÓN DE PRUEBAS 48

3.2.6 ARQUITECTURA 48
3.2.7 HERRAMIENTAS 48
3.2.8 AUTOMATIZACIÓN WEB 48
3.2.9 AUTOMATIZACIÓN MÓVIL 48
3.2.10 AUTOMATIZACIÓN MULTIPLATAFORMA 48
3.3 REPRESENTACIÓN DEL MODELO TEÓRICO 49
3.4 RELACIÓN ENTRE COMPONENTES 50
3.4.1 RELACIÓN R1: CALIDAD DE SOFTWARE – NIVELES DE PRUEBAS DE SOFTWARE 50
3.4.2 RELACIÓN R2: CALIDAD DE SOFTWARE – TIPOS DE PRUEBAS DE SOFTWARE 51
3.4.3 RELACIÓN R3: CALIDAD DE SOFTWARE – TÉCNICAS DE PRUEBAS DE SOFTWARE 51
3.4.4 RELACIÓN R4-R5-R6: AUTOMATIZACIÓN DE PRUEBAS – NIVELES, TIPOS, TÉCNICAS DE PRUEBAS DE SOFTWARE. 52
3.4.5 RELACIÓN R7-R8: AUTOMATIZACIÓN DE PRUEBAS – ARQUITECTURA BASE, HERRAMIENTAS. 53
3.4.6 RELACIÓN R9-R10: FRAMEWORK – ARQUITECTURA BASE, HERRAMIENTAS 54
3.4.7 RELACIÓN R11-R12: FRAMEWORK – AUTOMATIZACIÓN WEB, AUTOMATIZACIÓN MÓVIL 54
3.4.8 RELACIÓN R13-R14: AUTOMATIZACIÓN MULTIPLATAFORMA – AUTOMATIZACIÓN WEB, AUTOMATIZACIÓN MÓVIL 55
3.5 CONCLUSIONES 56

CAPITULO IV 57

4.1 OBJETIVOS DE LA PROPUESTA 57
4.2 METODOLOGÍA 58
4.3 ANÁLISIS 59
4.3.1 REQUERIMIENTOS FUNCIONALES 59
4.3.2 REQUERIMIENTOS NO FUNCIONALES 60
4.3.3 ESPECIFICACIÓN DE CASOS DE USO 61
4.3.3.1 ACTOR 61
4.3.3.2 CASOS DE USO 61
4.3.3.2.1 CASO DE USO: CREAR PRUEBAS 62
4.3.3.2.1 CASO DE USO: CORRER PRUEBAS 63
4.3.4 PAQUETES DE ANÁLISIS 63
4.4 DISEÑO 64
4.4.1 CLASES DE DISEÑO 64
4.4.2 DIAGRAMA DE SECUENCIA 65
4.5 IMPLEMENTACIÓN 66
4.5.1 HERRAMIENTAS DE IMPLEMENTACIÓN 66
4.5.1.1 SELENIUM 67
4.5.1.1.1 ARQUITECTURA DE SELENIUM 67
4.5.1.1.2 LOCALIZACIÓN DE ELEMENTOS WEB E INTERACCIONES 70
4.5.1.2 APPIUM 72
4.5.1.2.1 ARQUITECTURA DE APPIUM 73
4.5.1.2.2 LOCALIZACIÓN DE ELEMENTOS WEB E INTERACCIONES 74
4.5.1.3 CUCUMBER 74
4.5.1.4 JUNIT 75
4.5.1.5 SELENIDE 75
4.5.1.6 ASSERTJ 76

4.5.2 FRAMEWORK DE AUTOMATIZACIÓN DE PRUEBAS WEB MULTIPLATAFORMA 76
4.5.2.1 PRUEBAS EN TEXTO PLANO 76
4.5.2.2 DEFINICIÓN DE PASOS DE PRUEBAS 77
4.5.2.3 DEFINICIÓN DE PAGINAS 78
4.5.2.4 CONFIGURACIONES DE EJECUCIÓN 79
4.5.2.5 EJECUTOR DE PRUEBAS 80
4.5.2.6 LOCALIZADORES DE INTERFAZ DE USUARIO 80
4.6 PRUEBAS 81
4.7 MANTENIMIENTO 82
4.8 CONCLUSIONES 83

CONCLUSIONES 84

RECOMENDACIONES 86

REFERENCIAS BIBLIOGRAFICAS 87

ANEXOS 88

ANEXO 1: GLOSARIO 88
ANEXO 2: ENCUESTA 88

INDICE DE FIGURAS

FIGURA 1.1 APLICACIÓN DE AUTOMATIZACIÓN DE PRUEBAS EN PROYECTOS 14
FIGURA 1.2 APLICACIÓN DE AUTOMATIZACIÓN DE PRUEBAS WEB Y MÓVIL 15
FIGURA 1.3 AUTOMATIZACIÓN DE PRUEBAS POR PLATAFORMA 16
FIGURA 1.4 AUTOMATIZACIÓN DE PRUEBAS POR PLATAFORMA 17
FIGURA 1.5 HERRAMIENTAS DE AUTOMATIZACIÓN DE PRUEBAS USADAS EN WEB 18
FIGURA 1.6 HERRAMIENTAS DE AUTOMATIZACIÓN DE PRUEBAS USADAS EN MÓVIL 19
FIGURA 1.7 USO DE HERRAMIENTAS BDD EN PRUEBAS AUTOMATIZADAS 20
FIGURA 1.8 LENGUAJES DE PROGRAMACIÓN USADOS EN AUTOMATIZACIÓN DE PRUEBAS 21
FIGURA 1.9 ESCENARIOS IMPLEMENTADOS PARA LA FUNCIONALIDAD DE LOGIN 23
FIGURA 1.10 ESCENARIOS IMPLEMENTADOS PARA LA FUNCIONALIDAD DE PROYECTOS 25
FIGURA 1.11 LÍNEAS DE CÓDIGO DE AUTOMATIZACIÓN WEB VS AUTOMATIZACIÓN MÓVIL 28
FIGURA 1.12 RELACIÓN ENTRE APPIUM Y SELENIUM 29
FIGURA 1.13 INICIALIZACIÓN DEL DRIVER EN APPIUM 30
FIGURA 1.14 INICIALIZACIÓN DEL DRIVER EN SELENIUM 30
FIGURA 1.15 TIPOS DE LOCALIZADORES USADOS EN APPIUM Y SELENIUM PARA ELEMENTOS WEB 31
FIGURA 2.1 CICLO DE VIDA DEL DESARROLLO DE SOFTWARE 37

FIGURA 2.2 LA PIRÁMIDE DE PRUEBAS 38
FIGURA 2.3 ARQUITECTURA GENÉRICA DE UN FRAMEWORK DE AUTOMATIZACIÓN 41
FIGURA 3.1 ELEMENTOS DEL OBJETO DE ESTUDIO 46
FIGURA 3.2 ELEMENTOS DEL CAMPO DE ACCIÓN 47
FIGURA 3.3 REPRESENTACIÓN GRÁFICA DEL MODELO TEÓRICO 49
FIGURA 3.4 RELACIÓN 1 50
FIGURA 3.5 RELACIÓN 2 51
FIGURA 3.6 RELACIÓN 3 52
FIGURA 3.7 RELACIÓN 4-5-6 53
FIGURA 3.8 RELACIÓN 7-8 53
FIGURA 3.9 RELACIÓN 9-10 54
FIGURA 3.10 RELACIÓN 11-12 55
FIGURA 3.11 RELACIÓN 13-14 55
FIGURA 4.1 METODOLOGÍA DE DESARROLLO DE SOFTWARE CASCADA 58
FIGURA 4.2 ACTOR EN UML 61
FIGURA 4.3 CASO DE EN UML 62
FIGURA 4.4 CASO DE USO – CREAR PRUEBAS 62
FIGURA 4.5 CASO DE USO – CREAR PRUEBAS 63
FIGURA 4.6 PAQUETES – FRAMEWORK DE AUTOMATIZACIÓN 64
FIGURA 4.7 DIAGRAMA DE SECUENCIA – CREAR PRUEBAS 65
FIGURA 4.8 DIAGRAMA DE SECUENCIA – EJECUTAR PRUEBAS 66
FIGURA 4.8 ARQUITECTURA DE SELENIUM 70
FIGURA 4.9 FLUJO BÁSICO DE AUTOMATIZACIÓN DE PRUEBAS 70
FIGURA 4.9 EJEMPLO DE PRUEBA AUTOMATIZADA EN SELENIUM 72
FIGURA 4.10 ARQUITECTURA DE APPIUM 74
FIGURA 4.11 PRUEBA USANDO CUCUMBER 75
FIGURA 4.12 DEFINICIÓN DE PRUEBAS EN TEXTO PLANO 77
FIGURA 4.13 DEFINICIÓN DE PASOS DE PRUEBAS 78
FIGURA 4.14 DEFINICIÓN DE PÁGINAS (ELABORACIÓN PROPIA) 78
FIGURA 4.15 DEFINICIÓN PLATAFORMA EN PÁGINAS 79
FIGURA 4.16 CONFIGURACIONES DE EJECUCIÓN 79
FIGURA 4.17 EJECUTOR DE PRUEBAS 80
FIGURA 4.18 LOCALIZADORES DE INTERFAZ DE USUARIO 80

RESUMEN

La presente investigación se aborda la implementación de una propuesta a la problemática existente en automatización de pruebas de software para aplicaciones web multiplataforma.

Mediante el método de la medición se pudo observar que la mayoría de los proyectos de desarrollo de software web que tiene soporte multiplataforma (de escritorio y móvil) aplican automatización de pruebas solo a una de ellas, porque implica mucho más esfuerzo y están limitados en recursos humanos y tiempo, como también se pudo colectar información las tecnologías más usadas en estas áreas para enfocar la solución.

Automatizar pruebas de para aplicaciones web capaces de ser ejecutadas en plataformas de escritorio y móviles requiere más esfuerzo debido a que para cada plataforma se usan herramientas distintas, y pudo ser comprobada de manera práctica.

Para dar solución al problema se analizó las herramientas de automatización más usadas para aplicaciones web y móviles y se pudo evidenciar una estrecha relación entre tecnologías web y móviles disponibles en la actualidad, lo cual hacia posible una combinación de esas herramientas para generar un solo framework.

Al implementar la propuesta se evidencio que, mediante la combinación de tecnologías de automatización de pruebas web y móviles, se puede obtener un solo framework que cuente características estándar capaz de ejecutar pruebas en ambas plataformas, reduciendo el esfuerzo, y tiempo en desarrollo de pruebas en gran medida.

INTRODUCCION

Desde hace un tiempo atrás el software ha sido parte de día a día de las personas, el software es usado para temas laborales, comunicación, entretenimiento y mucho más.

Hace años atrás al hablar de software hacía referencia a programas que corrían en un computador para facilitar y automatizar tareas, con el paso del tiempo y el avance de la tecnología aparecieron los teléfonos inteligentes, tabletas y dispositivos portátiles.

Actualmente debido al gran número de personas que usan dispositivos portátiles inteligentes, las compañías de desarrollo de software se vieron en la necesidad de no solo desarrollar aplicaciones para computadores de escritorio, sino también sus versiones para dispositivos portátiles.

Al tener dispositivos móviles capaces de ejecutar programas, las personas tienen una gama alta de posibilidades en sus bolsillos, como ser manejar sus cuentas bancarias y transacciones desde sus teléfonos, comunicación mediante redes sociales, usar aplicaciones de entretenimiento y mucho más.

Una gran mayoría de las empresas de desarrollo de software aparte de contar con programas para computadores, desarrollan sus versiones móviles, pero hablando de desarrollo de software esto implica mucho más esfuerzo, recursos y tiempo.

El aseguramiento de calidad de software es una parte muy importante, debido a que asegura la entrega de software que cumplen ciertos estándares de calidad y garantizan su buen funcionamiento, hace un tiempo todo el proceso de aseguramiento de calidad de software se lo realizaba de forma manual, es decir ejecutando las aplicaciones y realizando los flujos que el usuario final realizaría para comprobar su correcto funcionamiento.

Hace un tiempo esta práctica se fue reemplazando por el aseguramiento de calidad automatizado, es decir ese flujo de trabajo que se espera que los usuarios finales realicen son ejecutados y validados automáticamente por un computador.

Los “scripts” que hacen que un flujo de trabajo se ejecute automáticamente son desarrollados por ingenieros de automatización de pruebas de software.

Actualmente existen herramientas para desarrollar pruebas automatizadas para aplicaciones móviles, web y de escritorio, pero tomando en cuenta que, si una empresa de desarrollo de software decide tener la versión web y móvil de la aplicación, se debe desarrollar las pruebas para web y para móvil en sus respectivas tecnologías, lo cual incrementa el tiempo del desarrollo de las pruebas, se requieren más ingenieros, y se deben mantener dos proyectos de automatización.

Hacerlo de forma manual significaría mucho más tiempo, debido a que tanto en computadores de escritorio y dispositivos móviles tenemos varios sistemas operativos (Widows, Linux, Android, iOS) y muchos navegadores (Chrome, Firefox,

opera) por lo tanto la combinación de ambientes en los cuales la aplicación podría ejecutarse es muy extensa.

Es por esto que la mayoría de las empresas de desarrollo de software buscan un equilibrio, realizan las pruebas a la aplicación web de forma automatizada, y ala móvil de forma manual y viceversa, esto para tener un equilibrio entre tiempo de pruebas y recursos.

El presente estudio aborda la investigación y desarrollo de un “framework” de automatización capaz de ejecutar pruebas automatizadas en múltiples plataformas web y móvil para reducir el tiempo de desarrollo de pruebas, tener una mayor cobertura en pruebas, y minimizar el esfuerzo y los recursos.

Problema científico

Con la introducción de la automatización de pruebas en el desarrollo de software se redujo en gran manera los tiempos de pruebas en aplicaciones, existen herramientas para aplicaciones de escritorio, web y móviles, pero cada herramienta está enfocada en cada uno de esos tipos de aplicaciones.

En las empresas de desarrollo, en el caso de aplicaciones web multiplataforma, normalmente se automatiza la parte web para computadoras de escritorio, y las pruebas en la aplicación móvil de realizan de forma manual, o viceversa.

Esta modalidad es aplicada debido a que se tendría que diseñar/desarrollar/mantener dos “frameworks” para cada tecnología, lo cual toma mucho tiempo, ya que se debe escribir código para las mismas pruebas usando las respectivas herramientas para cada tecnología, por esta razón en la mayoría de los casos se prefiere abarcar una tecnología, y la otra realizarla de forma manual.

Objeto de estudio

El objeto de estudio es la calidad de software y su relación que tiene con la automatización de pruebas de software.

Objetivo general

Diseñar e implementar un “framework” para la automatización de pruebas combinando tecnologías “móviles” y “de escritorio” para la ejecución de pruebas a aplicaciones multiplataforma.

Objetivos específicos

- Formular la problemática de la automatización de pruebas para aplicaciones web multiplataforma.
- Recopilar información de “framework” y herramientas para la automatización de pruebas en aplicaciones web y móviles.
- Identificar similitudes y diferencias entre “frameworks” actualmente usadas en la automatización de pruebas para aplicaciones web (móviles y de escritorio).
- Generar e implementar un “framework” para la automatización de pruebas a aplicaciones web multiplataforma, combinando tecnologías de escritorio y móviles.

Campo de acción

Automatización de pruebas de software.

Hipótesis

La combinación de tecnologías de automatización de pruebas (móviles y de escritorio) podrían permitir el diseño e implementación de un “framework” capaz de ejecutar las mismas pruebas en diferentes navegadores y tipos de dispositivos, ya que existen similitudes entra las herramientas usadas para ambas tecnologías.

De esta manera solo sería necesario diseñar/desarrollar/ejecutar un solo “framework” para correr pruebas de software en aplicaciones web multiplataforma.

Aporte teórico

Generación y diseño de un “framework” para la automatización de pruebas de aplicaciones web multiplataforma basada en la combinación de tecnologías web de escritorio y móviles.

Significancia practica

El “framework” generado podrá ser aplicado por ingenieros de control de calidad de software que tengan la necesidad de ejecutar pruebas web en computadores de escritorio y dispositivos móviles.

La combinación de tecnologías actuales para la automatización de pruebas para aplicaciones web de escritorio y móviles permitirá generar un “framework” capaz de ejecutar pruebas en múltiples plataformas.

Métodos

Los métodos usados en la presente investigación son: el método hipotético deductivo, el método de la medición y observación.

CAPITULO I

CARACTERIZACION DEL ESTADO ACTUAL DE LA AUTOMATIZACION DE PRUEBAS DE SOFTWARE PARA APLICACIONES MULTIPLATAFORMA

En el presente capítulo se abordará todo lo referente a la aplicación de los instrumentos necesarios para medir el estado actual de la automatización de pruebas de software para aplicaciones con soporte web y móvil.

Se expondrá de forma detallada las metodologías empleadas y los resultados obtenidos como consecuencia de su aplicación. Finalmente se hará la caracterización del estado actual de la automatización de pruebas de software para aplicaciones con soporte web y móvil, lo cual permitirá elaborar con mayor precisión la propuesta que dé solución al problema.

1.1 Metodologías empleadas para la caracterización

Para la realización de la caracterización se hizo le diagnóstico, para los cuales se usó el método numérico y el de la observación.

El método numérico se usó para obtener datos cuantitativos acerca del estudio de la problemática y el estado actual de la automatización de pruebas de software para aplicaciones multiplataforma (con soporte web y móvil).

Según el conocimiento que se tiene actualmente de que es necesario el uso de dos herramientas distintas para la automatización de pruebas para cada plataforma, se sabe que muchas compañías de desarrollo de software solo automatizan pruebas para una plataforma. Se desea saber un porcentaje aproximado de las compañías que toman esta medida, las razones específicas, tecnologías más frecuentes de uso para poder implementar una propuesta que solucione los problemas identificados.

Por otro lado, se desea medir la cantidad de esfuerzo necesario para desarrollar pruebas de software usando cada herramienta para cada plataforma, por lo cual se puede realizar mediciones de líneas de código.

Por estas razones el método numérico nos ayudara a tener valores cuantitativos acerca la problemática, y ciertos aspectos técnicos.

El método de la observación se aplicará para el análisis de las herramientas de automatización de pruebas de plataformas web y móvil para encontrar sus diferencias y similitudes.

1.2 Diagnostico de la automatización de pruebas de software

Para la evaluación de la problemática actual en el aseguramiento de calidad automatizado de compañías de software las cuales cuentan versiones de los productos que desarrollan tanto para web y dispositivos móviles es necesario conocer los factores con los cuales se lidian.

Inicialmente se conoce de manera general que generar pruebas automatizadas para cubrir pruebas de calidad de software para plataformas web y móviles, es requerido el uso de 2 herramientas distintas, los cuales tienen ciertas similitudes y diferencias.

Debido al uso de dos herramientas distintas para cada plataforma, es evidente que se necesita más esfuerzo en la hora de desarrollar las pruebas para cubrir ambas plataformas, se sabe también de manera general que gran parte de las compañías que desarrollan aplicaciones con soporte para ambas plataformas, dejan de lado la cobertura de pruebas automatizadas para alguna de las dos plataformas, inicialmente se puede pensar porque esto representa más esfuerzo, por lo tanto más recursos especializados en la automatización de pruebas para cada plataforma y más tiempo en tareas de control de calidad.

1.2.1 Estado actual de la automatización de pruebas en proyectos de software con soporte multiplataforma

Para analizar el estado actual y los problemas con los que lidian los proyectos de desarrollo de aplicaciones con soporte multiplataforma a la hora de realizar pruebas automatizadas de software se realizó una encuesta a 500 personas, las cuales trabajan actualmente en proyectos de desarrollo de aplicaciones con soporte multiplataforma (web y móvil).

Se pudo observar que el 93,6% de los encuestados aplican automatización de pruebas de software en sus proyectos y un 6.4% solo realizan pruebas de software de forma manual.

Esto nos muestra que la automatización de pruebas de software es muy valorada dentro del ciclo de desarrollo de software, y una gran mayoría de los proyectos de desarrollo de software lo adoptan actualmente.

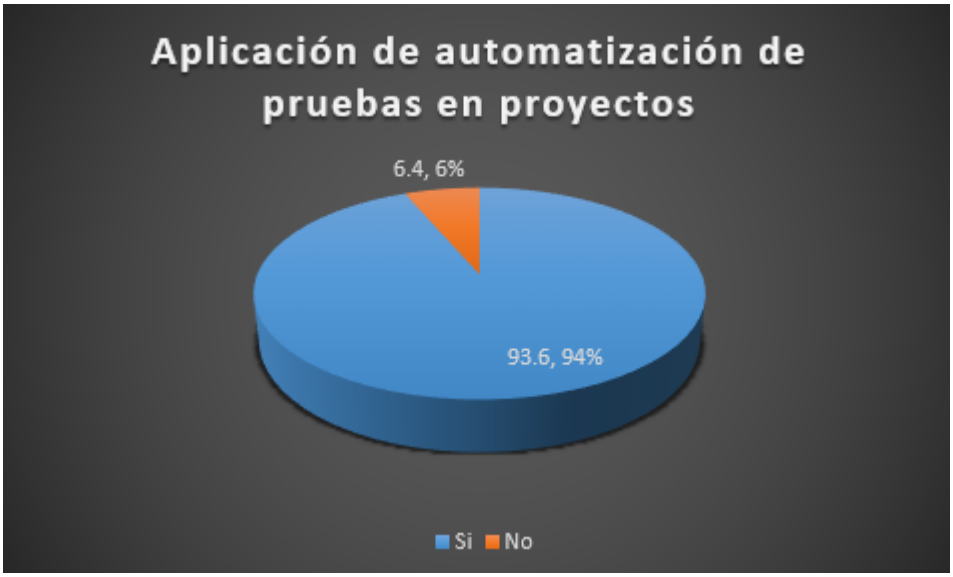

Figura 1.1 Aplicación de automatización de pruebas en proyectos (Elaboración propia)

El 83,20% de los encuestados manifestaron que las pruebas de software automatizadas solo se desarrollan para una de las plataformas ya sea para web o para móvil, mientras tanto un 10,40% de los encuestados manifestaron que en sus proyectos se desarrollan pruebas de software con soporte para web y móvil.

El otro 6,40% son los que manifestó que solo se realizan pruebas de software manuales en sus respectivos proyectos.

Se puede observar que en un gran porcentaje solo se prefiere realizar pruebas automatizadas para una plataforma, realizándose en la otra prueba de software de forma manual.

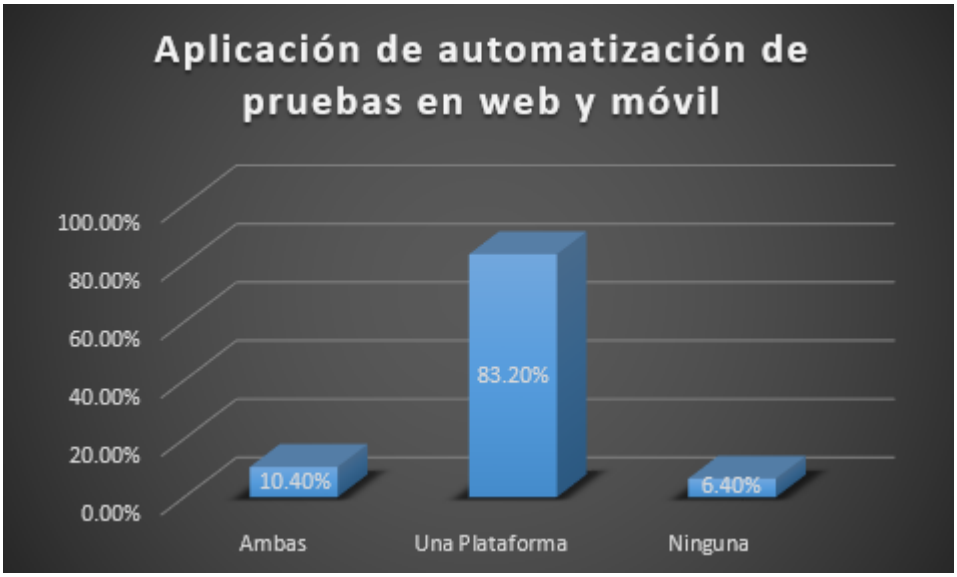

Figura 1.2 Aplicación de automatización de pruebas web y móvil (Elaboración propia)

El 78% de los encuestados manifestaron que las pruebas automatizadas que desarrollan en sus proyectos solo son para web, el 5,20% solo desarrolla las pruebas automatizadas para móvil, mientras tanto solo el 10,40% tiene pruebas automatizadas que cubran las plataformas web y móvil.

Este fenómeno puede deberse a que la mayoría de los usuarios prefieren realizar ciertos tipos de tareas desde un ordenador antes que móviles, se puede apreciar que por el momento la mayoría de los proyectos de desarrollo de software dan más prioridad a la cobertura de pruebas automatizadas en sus aplicaciones para sus versiones web, solo un número reducido considera dar prioridad a las versiones móviles o ambas.

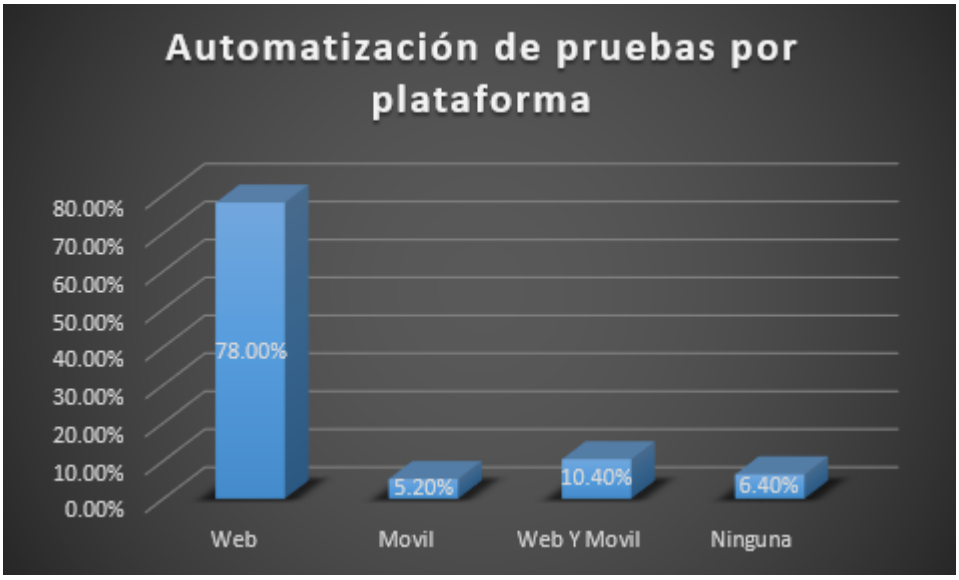

Figura 1.3 Automatización de pruebas por plataforma (Elaboración propia)

Del 83,20% de los encuestados que manifestaron que en sus proyectos solo se realiza pruebas automatizadas para una plataforma se observó:

- 79,74% de los encuestados manifestaron que solo se automatiza pruebas para una sola plataforma debido a que esto implica mucho más esfuerzo.
- 36,67% de los encuestados manifestaron que una de las plataformas las cuales no se aplican pruebas automatizadas es menos usada que la otra, por lo tanto, tiene menos prioridad a la hora de la automatización de pruebas.
- El 31,79% manifestó que el proceso de automatizar pruebas para ambas plataformas implica más tiempo, por lo cual se da más prioridad solo a una plataforma.
- El 14,36% manifestó que para automatizar pruebas para ambas plataformas se requiere más recursos humanos si es que no se desea tener un impacto en los tiempos del proyecto.
- El 4,10% manifestó que es necesario tener ingenieros especializados para la automatización de pruebas para cada área.
- Solo un 0,77% manifestó que no realizan pruebas automatizadas para ambas plataformas debido a limitaciones en sus recursos de hardware que disponen.

- Un 0,51% manifestó que por otras razones no realizaban las pruebas automatizadas para ambas plataformas.

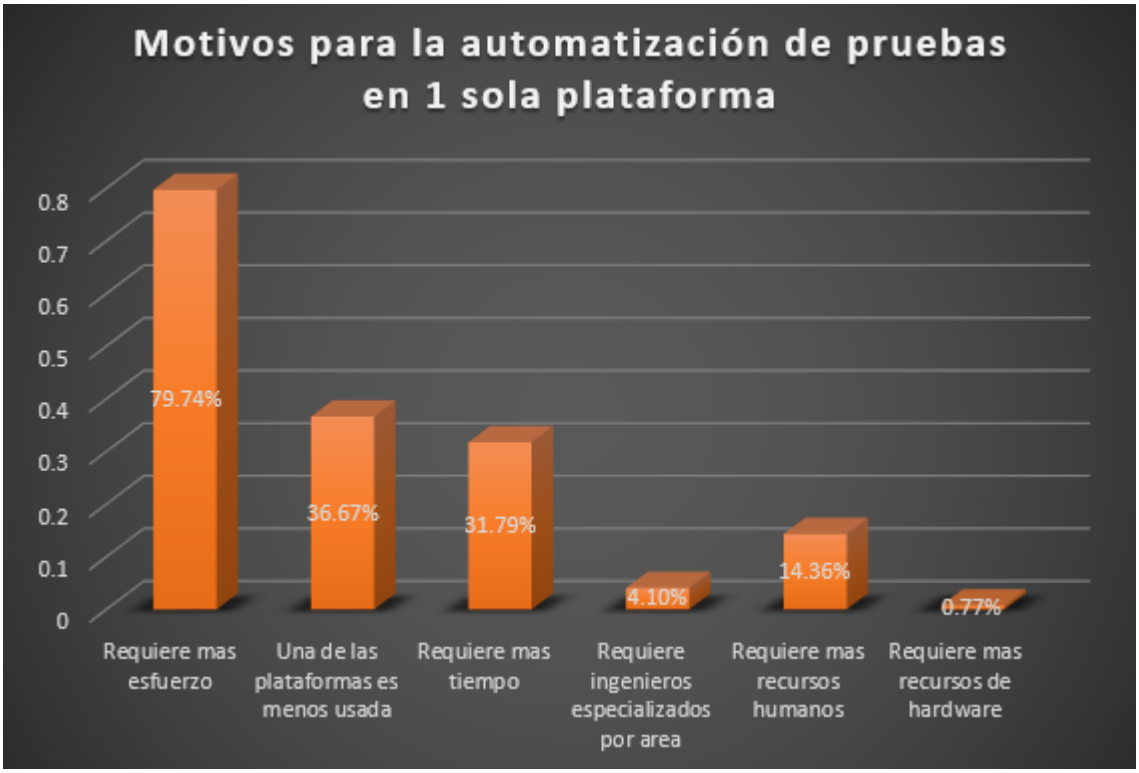

Figura 1.4 Automatización de pruebas por plataforma (Elaboración propia)

Para la implementación de una propuesta es necesario conocer también a fondo las tecnologías más usadas para la automatización de pruebas en ambas plataformas, de esta manera basar la solución en estas tecnologías.

Por esta razón aparte de estar interesados en los problemas actuales en la automatización de pruebas para múltiples plataformas se incluyó preguntas para la recolección de datos técnicos.

Un 69% de los encuestados que realizan pruebas automatizadas para web señalo que usa la herramienta Selenium en su desarrollo.

Un 14,72% de los encuestados usa Protactor como herramienta de desarrollo de pruebas automatizadas.

El otro 2% - 3% usa herramientas como ser Sikuli, Test Complete, Watir, Ranorex y otros.

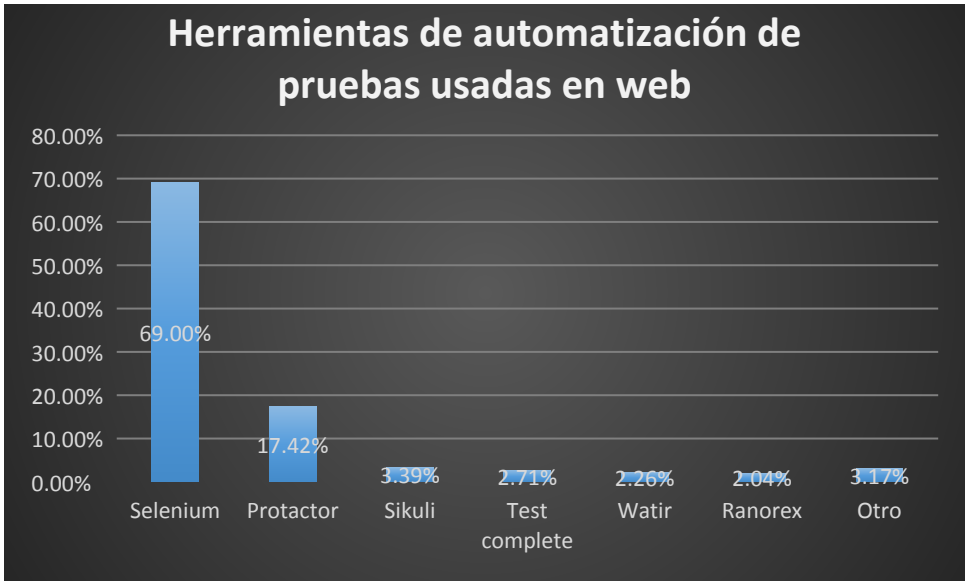

Figura 1.5 Herramientas de automatización de pruebas usadas en web (Elaboración propia)

Con respecto a las herramientas de automatización de pruebas usadas para aplicaciones móviles se tiene que:

Un 62,82% de los encuestados aplica Appium como herramienta de automatización de pruebas para aplicaciones.

Entre un 10% - 13% de los encuestados usan Calabash o Esspreso para la automatización de pruebas orientadas a aplicaciones móviles.

El otro 2% - 6% de los encuestados manifestaron usar Robotium, Xamarin u otra herramienta.

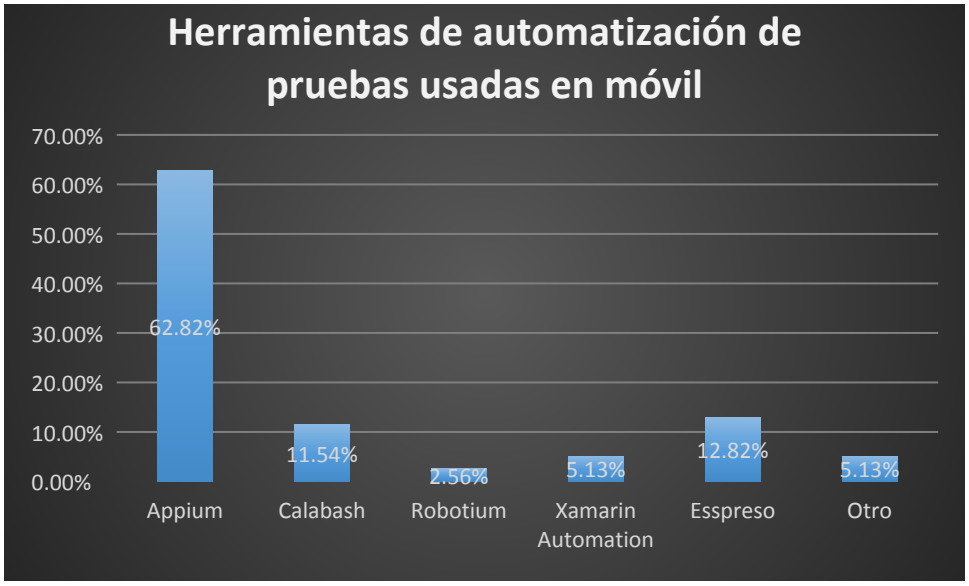

Figura 1.6 Herramientas de automatización de pruebas usadas en móvil (Elaboración propia)

La aplicación de BDD en pruebas de software fue ganando popularidad den los últimos años, por esta razón se incluyó en la investigación para conocer cuáles son las herramientas más usadas.

El 69,02% de los encuestados manifestó usar Cucumber como herramienta BDD en las pruebas automatizadas de software.

Un 11,97% de los encuestados manifestó usar JBehave como herramienta BDD en las pruebas automatizadas de software.

El otro 1% - 7% manifestó usar JDave, Concordion, otra herramienta BDD o ninguna en la automatización de pruebas de software.

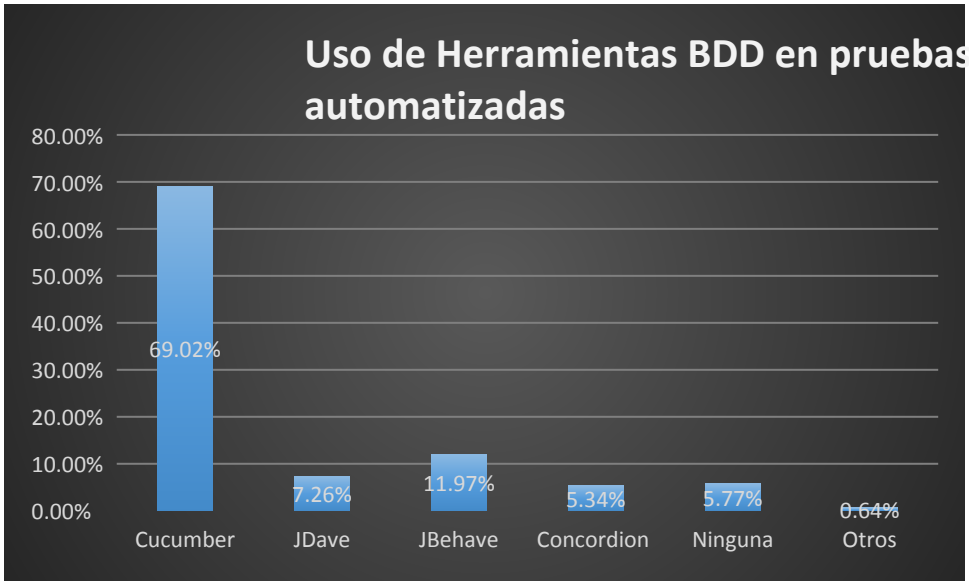

Figura 1.7 Uso de herramientas BDD en pruebas automatizadas (Elaboración propia)

También fue importante conocer acerca de los lenguajes de programación más usados para la automatización de pruebas de software y se obtuvo que:

Un 36,75% de los encuestados usa Java como lenguaje de programación para pruebas automatizadas.

El 19, 87% de los encuestados usa C# como lenguaje de programación para pruebas automatizadas.

El 19,44% de los encuestados usa Ruby, el 12,87% Phyton, el 8,97% JavaScript, y el 2,14% otro lenguaje de programación.

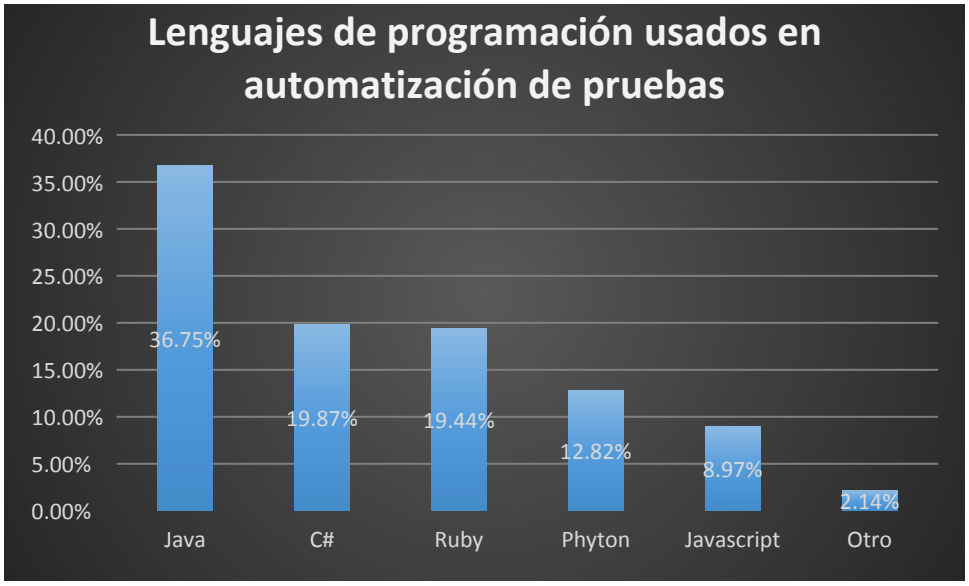

Figura 1.8 Lenguajes de programación usados en automatización de pruebas (Elaboración propia)

1.2.2 Implementación de herramientas de automatización en proyectos de software con soporte multiplataforma

Analizando los resultados mostrados por la encuesta se evidencia que gran parte de los proyectos de software solo cuentan con pruebas automatizados para una sola plataforma, esto debido a que automatizar pruebas para ambas requiere mucho más esfuerzo, impacto en tiempos de desarrollo y más personal.

Teniendo en cuenta que las herramientas usadas para cada plataforma son distintas, se observó y analizo las características de las herramientas más usadas, Appium para móviles y Selenium para Web.

Para tener un punto de referencia del esfuerzo que implica tener un grupo de pruebas automatizadas que puedan ser ejecutadas en ambas plataformas midiendo las líneas de código implementadas, se desarrolló 4 pruebas de software.

Las pruebas de software desarrolladas usan las tecnologías más usadas para web (Selenium) y móvil (Appium).

Debido a que nuestra propuesta es la implementación de un framework capaz de ejecutar las mismas pruebas en ambas plataformas, reduciendo el esfuerzo de implementación, conocer cuantas líneas de código nos tomaría implementar esas pruebas en las tecnologías actuales nos será de ayuda para validar que cumplimos con nuestros objetivos y tenemos una solución al problema.

Las pruebas automatizadas son para la página web: https://todo.ly/ que también puede ser usada en dispositivos móviles, la cual es una página de gestión de tareas libre que solo requiere un registro con correo electrónico para empezar a utilizarla.

Se implementó las siguientes pruebas:

Funcionalidad: Login

Escenario 1: Login con credenciales validas en la página https://todo.ly/

Pasos:

1. Ingresar a https://todo.ly/
2. Realizar clic en el botón Login
3. Ingresar un email valido
4. Ingresar un password valido
5. Realizar clic en el botón Login

Resultado esperado: La página home de todo.ly debe ser mostrada.

Escenario 2: Login con credenciales invalidas en la página https://todo.ly/

Pasos:

1. Ingresar a https://todo.ly/
2. Realizar clic en el botón Login
3. Ingresar un email invalido
4. Ingresar un password invalido
5. Realizar clic en el botón Login

Resultado esperado: Un mensaje de error de credenciales invalidas debe ser mostrado

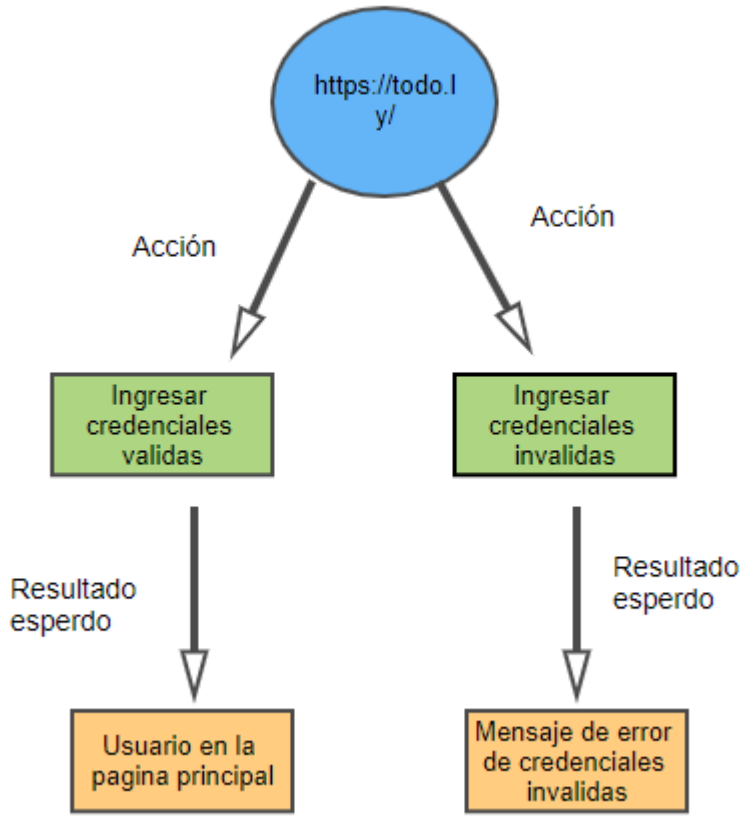

Figura 1.9 Escenarios implementados para la funcionalidad de Login (Elaboración propia)

Funcionalidad: Proyectos

Escenario 1: Crear proyecto usando valores cadena en el nombre

Pasos:

1. Ingresar a https://todo.ly/
2. Realizar clic en el botón Login
3. Ingresar un email invalido
4. Ingresar un password invalido

5. Realizar clic en el botón Login
6. Realizar clic en botón de agregar nuevo proyecto
7. Ingresar un nombre que contenga valores cadena, ejemplo “MyProyecto”
8. Realizar clic en el botón agregar

Resultado esperado: El nuevo proyecto creado debe ser mostrado en la lista de proyectos.

Escenario: Crear proyecto usando valores numéricos en el nombre

Pasos:

1. Ingresar a https://todo.ly/
2. Realizar clic en el botón Login
3. Ingresar un email invalido
4. Ingresar un password invalido
5. Realizar clic en el botón Login
6. Realizar clic en botón de agregar nuevo proyecto
7. Ingresar un nombre que contenga valores cadena, ejemplo “12345”
8. Realizar clic en el botón agregar

Resultado esperado: El nuevo proyecto creado debe ser mostrado en la lista de proyectos.

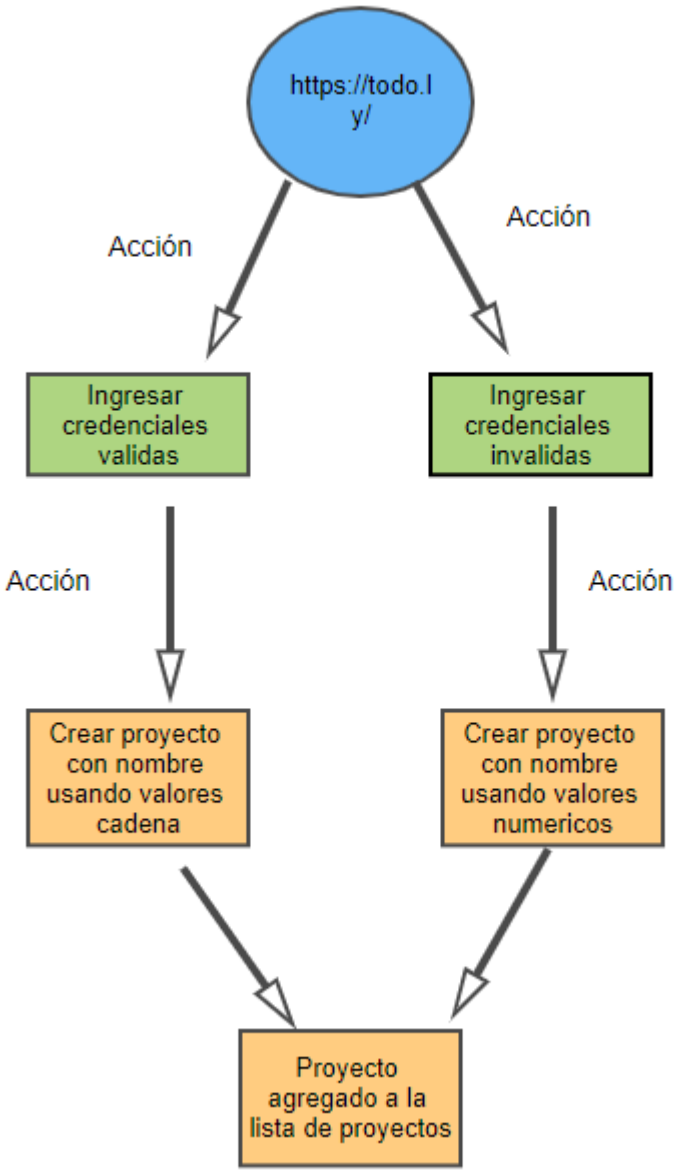

Figura 1.10 Escenarios implementados para la funcionalidad de Proyectos (Elaboración propia)

Para la creación de pruebas automatizadas de forma simple se requieren algunos componentes básicos:

- Clases que manejen el driver para la automatización.
- Clases contenedoras de los localizadores de los elementos de la interfaz de usuario.

- Clases con acciones y verificaciones en la interfaz de usuario.
- Pruebas escritas en lenguaje natural.

Para el análisis del total de líneas de código implementadas para automatizar las pruebas para la plataforma web se usó Java + Selenium +Cucumber, debido a que se encontró mediante las encuestas que estas eran las herramientas más usadas.

Las clases que se encargan de manejar el driver para la automatización de pruebas alcanzaron las 90 líneas de código.

Las clases que se encargan de manejar las páginas de interfaz de usuario en las pruebas y sus acciones/verificaciones necesitaron la implementación de 110 líneas de código.

Lo utilitarios usados durante la automatización requirieron 201 líneas de código.

Las clases de definición de pasos de las pruebas requirieron 77 líneas de código para establecer una relación entre pasos representados en lenguaje natural con código java.

Las pruebas escritas en lenguaje natural requirieron 44 líneas de texto plano que describen las pruebas a ser ejecutadas.

En total se tiene que para implementar las 4 pruebas descritas para la plataforma web se requirió 522 líneas de código java/texto plano.

Para el análisis del total de líneas de código implementadas para automatizar las pruebas para la plataforma móvil se usó Appium + Java + Cucumber ya que se encontró mediante las encuestas que estas tecnologías son las más usadas en las pruebas automatizadas para dispositivos móviles.

Las clases que se encargan de manejar el driver para la automatización de pruebas alcanzaron las 231 líneas de código.

Las clases que se encargan de manejar las páginas de interfaz de usuario en las pruebas y sus acciones/verificaciones necesitaron la implementación de 117 líneas de código.

Lo utilitarios usados durante la automatización requirieron 115 líneas de código.

Las clases de definición de pasos de las pruebas requirieron 77 líneas de código para establecer una relación entre pasos representados en lenguaje natural con código java.

Las pruebas escritas en lenguaje natural requirieron 44 líneas de texto plano que describen las pruebas a ser ejecutadas.

En total se tiene que para implementar las 4 pruebas descritas para la plataforma web se requirió 584 líneas de código java/texto plano.

Se pudo evidenciar que para tener las 4 pruebas automatizadas que sean capaces de correr en ambas plataformas es necesario escribir 1106 líneas de código.

No hay mucha diferencia en las líneas de código que se requiere para automatizar las 4 pruebas en cada plataforma, son alrededor de 500, pero se puede evidenciar que prácticamente tener las mismas 4 pruebas en ambas plataformas requiere casi un doble esfuerzo.

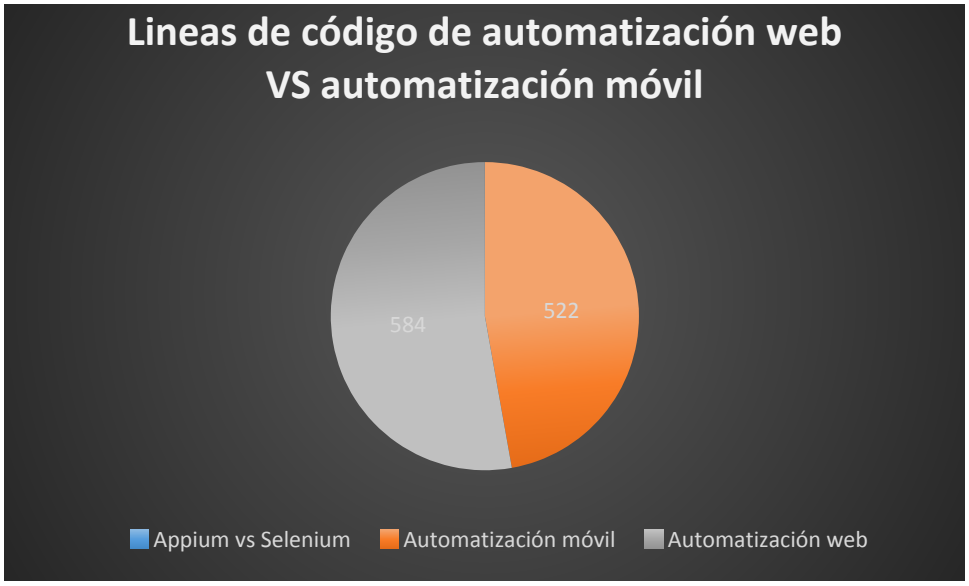

Figura 1.11 Líneas de código de automatización web vs automatización móvil (Elaboración propia)

1.2.3 Appium Vs Selenium

Appium es un framework para la automatización de pruebas para tecnologías móviles, Selenium es un framework para la automatización de pruebas para tecnologías web.

Hay una estrecha relación entre Appium y Selenium en términos de su arquitectura, Appium usa Selenium para las pruebas web en dispositivos móviles, actualmente existe una relación entre ambas herramientas, ya que Appium usa librerías de Selenium como dependencias.

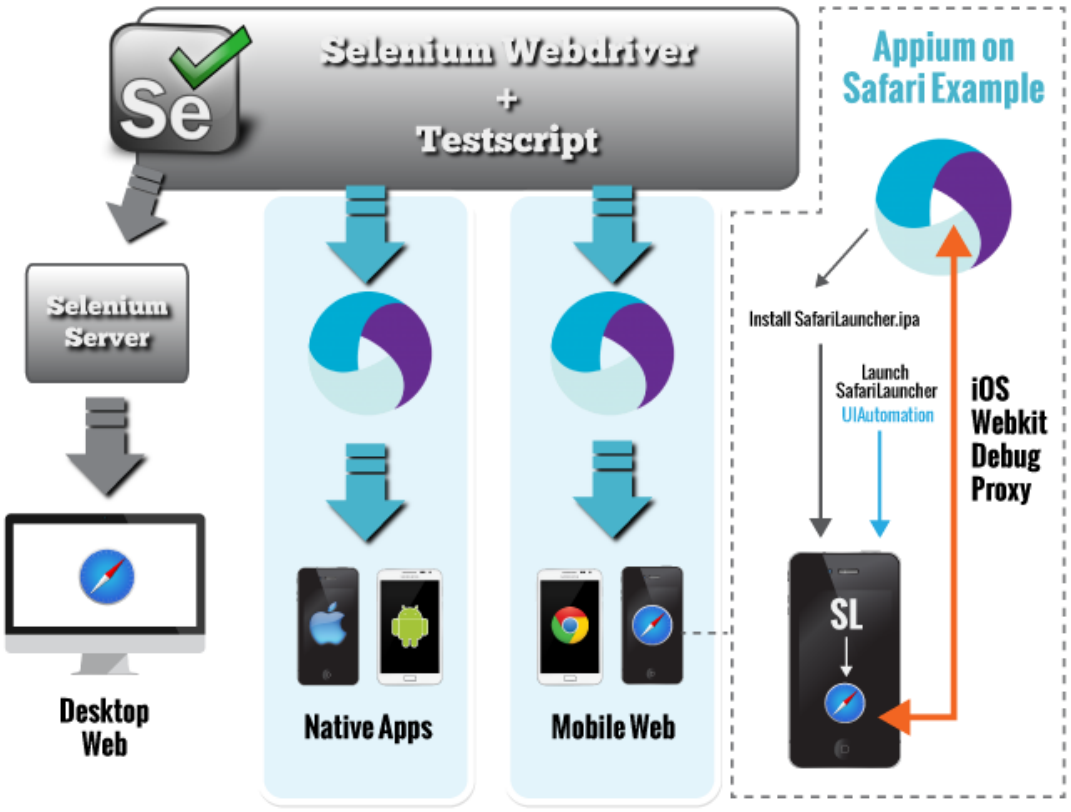

Figura 1.12 Relación entre Appium y Selenium (Elaboración propia)

Esta relación de dependencia que existe entre ambas plataformas hace que exista ciertas similitudes y diferencias en el código implementado en el proyecto: todo.ly

La principal diferencia es a la hora de configurar el driver, pues Selenium solo necesita especificar el tipo de driver a usar según el navegador, y tener disponible el driver.

Para la parte móvil Appium necesita también al igual que Selenium el driver para el navegador deseado, pero también se debe configurar ciertos parámetros como el dispositivo móvil donde se correrá las pruebas (físico o real), sistema operativo, servidor Appium y otros.

```
2
3  import java.net.MalformedURLException;
4  import java.net.URL;
5
6  import org.openqa.selenium.remote.DesiredCapabilities;
7  import io.appium.java_client.AppiumDriver;
8  import io.appium.java_client.MobileElement;
9  import io.appium.java_client.android.AndroidDriver;
10
11 public class ChromeTest {
12
13     public static void main(String[] args) {
14
15         //Set the Desired Capabilities
16         DesiredCapabilities caps = new DesiredCapabilities();
17         caps.setCapability("deviceName", "My Phone");
18         caps.setCapability("udid", "ENUL6303030010"); //Give Device ID of your mobile phone
19         caps.setCapability("platformName", "Android");
20         caps.setCapability("platformVersion", "6.0");
21         caps.setCapability("browserName", "Chrome");
22         caps.setCapability("noReset", true);
23
24         //Set ChromeDriver location
25         System.setProperty("webdriver.chrome.driver","C:\\selenium_drivers\\chromedriver.exe");
26
27         //Instantiate Appium Driver
28         AppiumDriver<MobileElement> driver = null;
29         try {
30             driver = new AndroidDriver<MobileElement>(new URL("http://0.0.0.0:4723/wd/hub"), cap
31
32         } catch (MalformedURLException e) {
33             System.out.println(e.getMessage());
34         }
35
36         //Open URL in Chrome Browser
37         driver.get("http://www.google.com");
38     }
39 }
```

Figura 1.13 Inicialización del driver en Appium (http://www.automationtestinghub.com/launch-chrome-browser-on-mobile-device/)

```
1  package demotc;
2
3  import org.openqa.selenium.WebDriver;
4  import org.openqa.selenium.chrome.ChromeDriver;
5  import org.testng.annotations.Test;
6  public class Openchrome {
7
8  @Test
9  public void test12() throws Exception{
10
11 // Initialize browser
12 WebDriver driver=new ChromeDriver();
13
14 // Open Google
15 driver.get("http://www.google.com");
16
17 // Close browser
18 driver.close();
19 }
20
21 }
```

Figura 1.14 Inicialización del driver en Selenium (http://learn-automation.com/launch-chrome-browser-using-selenium-webdriver/)

Las acciones que se usan para interactuar con la interfaz de usuario tienen ciertas similitudes y diferencias, algunas acciones como el clic, llenar campos de texto y verificaciones son las mismas para ambas, en el caso de Appium incluye algunas acciones más específicas para dispositivos móviles, como el tap, press and hold y otros.

La forma de localizar elementos de aplicaciones multiplataforma es prácticamente la misma, se usan localizadores de elementos web: xpath, css, id, link text, classs name y otros. Solo en algunos casos es necesario ajustar localizadores específicos para cada plataforma, debido a que algunas veces el mismo localizador de un componente de interfaz de usuario no funciona para ambas plataformas.

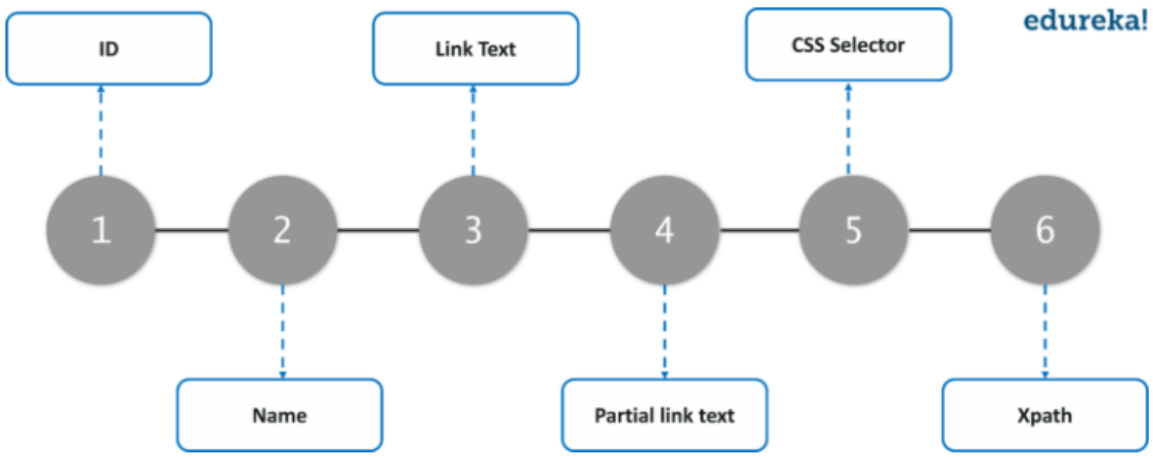

Figura 1.15 Tipos de localizadores usados en Appium y Selenium para elementos web (https://www.edureka.co/blog/locators-in-selenium/)

1.3 Conclusiones

De acuerdo a los estudios realizados se evidencio que muchos proyectos de desarrollo de software dejan de lado las pruebas automatizadas de una de las plataformas, debido a que este proceso requiere más esfuerzo, más tiempo, y recursos humanos.

Con la implementación de 4 pruebas se pudo evidenciar que prácticamente el esfuerzo es el doble para cubrir los mismos escenarios de pruebas en ambas plataformas, las líneas de código necesarias son prácticamente el doble.

Se estudió y se concluyó que las herramientas más usadas en automatización de pruebas web (Selenium) y móviles (Appium) tienen algunas diferencias y muchas cosas en común, debido a que Appium usa Selenium como dependencia, esto nos ayudó a entender que se puede aprovechar esas similitudes para combinar esas herramientas y obtener una solución que reduzca el esfuerzo y líneas de código a la hora de desarrollar pruebas automatizadas que cubran ambas plataformas.

Se pudo obtener información relevante sobre las tecnologías más usadas en la automatización de pruebas para ambas plataformas, de esta manera enfocar la solución en ellas.

CAPITULO II

MARCO TEORICO

En el presente capitulo se abordará conceptos importantes referentes al aseguramiento de calidad de software, automatización de pruebas de software, herramientas en automatización de software que tienen relevancia en nuestro estudio.

2.1 Calidad de software

2.1.1 Definición

El aseguramiento de calidad de software es una parte importante dentro del ciclo de desarrollo de software, un concepto importante es la de calidad de software.

En sentidos generales la calidad de software se define como:

“Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan” (Pressman, 2010, pp. 340)

Esta definición es útil para enfatizar tres puntos importantes:

1. Un proceso eficaz de software establece la infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad. Los aspectos de administración del proceso generan las verificaciones y equilibrios que ayudan a evitar que el proyecto caiga en el caos, contribuyente clave de la mala calidad. Las prácticas de ingeniería de software permiten al desarrollador analizar el problema y diseñar una solución sólida, ambas actividades críticas de la construcción de software de alta calidad. Por último, las actividades sombrilla, tales como

administración del cambio y revisiones técnicas, tienen tanto que ver con la calidad como cualquier otra parte de la práctica de la ingeniería de software.

2. Un producto útil entrega contenido, funciones y características que el usuario final desea; sin embargo, de igual importancia es que entrega estos activos en forma confiable y libre de errores. Un producto útil siempre satisface los requerimientos establecidos en forma explícita por los participantes. Además, satisface el conjunto de requerimientos (por ejemplo, la facilidad de uso) con los que se espera que cuente el software de alta calidad.

3. Al agregar valor para el productor y para el usuario de un producto, el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales. La organización que elabora el software obtiene valor agregado porque el software de alta calidad requiere un menor esfuerzo de mantenimiento, menos errores que corregir y poca asistencia al cliente. Esto permite que los ingenieros de software dediquen más tiempo a crear nuevas aplicaciones y menos a repetir trabajos mal hechos. La comunidad de usuarios obtiene valor agregado porque la aplicación provee una capacidad útil en forma tal que agiliza algún proceso de negocios. El resultado final es 1) mayores utilidades por el producto de software, 2) más rentabilidad cuando una aplicación apoya al proceso de negocios y 3) mejor disponibilidad de información, que es crucial para el negocio. (Pressman, 2010, pp. 340 - 341)

2.1.2 Atributos clave de calidad de software

El estándar ISO 9126 se desarrolló con la intención de identificar los atributos clave del software de cómputo. Este sistema identifica seis atributos clave de la calidad:

Funcionalidad: Grado en el que el software satisface las necesidades planteadas según las establecen los atributos siguientes: adaptabilidad, exactitud, interoperabilidad, cumplimiento y seguridad.

Confiabilidad: Cantidad de tiempo que el software se encuentra disponible para su uso, según lo indican los siguientes atributos: madurez, tolerancia a fallas y

recuperación. Usabilidad. Grado en el que el software es fácil de usar, según lo indican los siguientes subatributos: entendible, aprendible y operable.

Eficiencia: Grado en el que el software emplea óptimamente los recursos del sistema, según lo indican los subatributos siguientes: comportamiento del tiempo y de los recursos.

Facilidad de recibir mantenimiento: Facilidad con la que pueden efectuarse reparaciones al software, según lo indican los atributos que siguen: analizable, cambiable, estable, susceptible de someterse a pruebas.

Portabilidad: Facilidad con la que el software puede llevarse de un ambiente a otro según lo indican los siguientes atributos: adaptable, instalable, conformidad y sustituible. (Pressman, 2010, pp. 343)

2.1.3 Aseguramiento de calidad de software

Según la IEE el aseguramiento de calidad de software se define como:

“Un modelo sistematizado y planificado de todas las acciones necesarias para proporcionar la adecuada confianza que el proyecto precisa para los requerimientos técnicos establecidos”.

Dentro del proceso de aseguramiento de calidad de software existen varias tareas que se ejecutan:

1. Asegurar un nivel aceptable de confianza de que el software cumplirá a requisitos técnicos funcionales.
2. Asegurar un nivel aceptable de confianza de que el software cumplirá a los requisitos presupuestarios y gerenciales.
3. Iniciación y manejo de las actividades para mejorar y maximizar la eficiencia del desarrollo de software.
4. Asegurar un nivel aceptable de confianza de que la actividad de mantenimiento de software cumplirá los requerimientos técnicos funcionales.

5. Asegurar un nivel aceptable de confianza de que las actividades de mantenimiento de software cumplirán los requisitos presupuestarios y gerenciales.
6. Iniciación y manejo de las actividades para mejorar y maximizar la eficiencia del mantenimiento de software. (Galin, 2004, pp. 29)

2.1.4 Errores de software

Una persona puede cometer un error, que a su vez puede producir un defecto en el código del programa o en un documento. Si se ejecuta un defecto en el código, el sistema puede no hacer lo que debiera (o hacer algo que no debiera), lo que provocaría un fallo.

Algunos defectos de software, sistemas o documentos pueden dar lugar a fallos, pero no todos los defectos lo hacen.

Los defectos suceden por que las personas somos falibles y por qué existen factores como presión, códigos complejos, infraestructuras complejas, tecnologías cambiantes y/o muchas interacciones del sistema.

Los fallos también pueden venir provocados por condiciones medioambientales. Así por ejemplo la radiación, el magnetismo, los campos electrónicos y la contaminación pueden provocar faltas en sistemas o afectar a la ejecución de un software al modificar las condiciones del hardware. (ISQTB, 2010, pp. 13)

En resumen, un error de software es conocido como “bug”, y se produce cuando el software no realiza lo que se supone que debe realizar, desencadena resultados no deseados y por ende tienen impacto directo en la calidad de software.

2.2 Pruebas de software

Las pruebas de software pueden ser aplicadas a una amplia gama de proyectos de desarrollo con el propósito de reducir riesgos de gastos en desarrollo no panificados o riesgos de fracaso de los proyectos.

Por esta razón las pruebas de software están contempladas dentro del ciclo de vida del desarrollo de software.

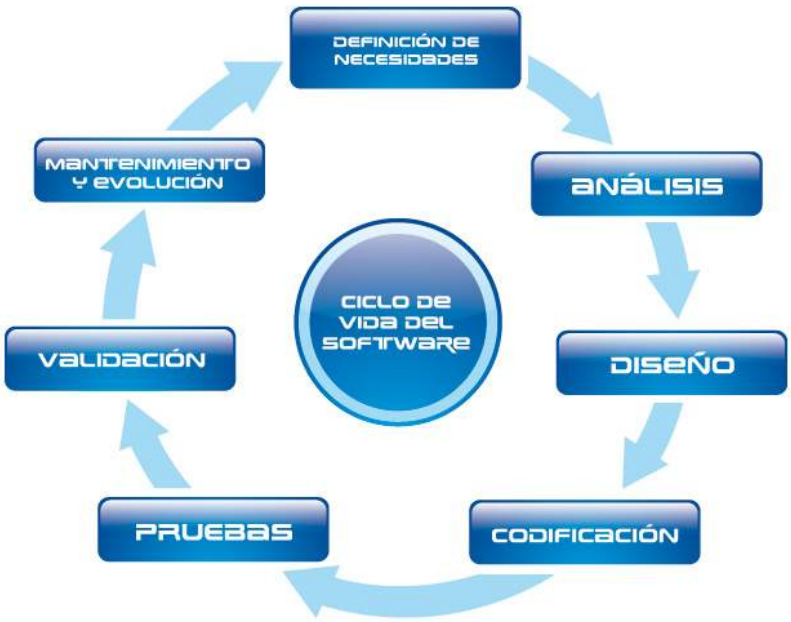

Figura 2.1 Ciclo de vida del desarrollo de software
(https://www.ecured.cu/Ciclo_de_vida_del_software)

2.2.1 Niveles de pruebas

Las pruebas son derivadas de los requerimientos y especificaciones, diseño de artefactos, o del código fuente. Un nivel diferente de pruebas acompaña las diferentes actividades del desarrollo.

Pruebas de componentes: tiene como objetivo localizar defectos y comprobar el funcionamiento de módulos de software, programas, objetos, clases.

Pruebas de integración: se ocupan de probar las interfaces entre componentes, las interacciones de distintas partes del mismo sistema.

Pruebas de sistema: Se refieren al comportamiento de todo un sistema/producto.

Pruebas de aceptación: Su objetivo es crear confianza en el sistema, partes del sistema o características no funcionales del sistema. (ISQTB, 2010, pp. 31)

A estos niveles de prueba se los denomina la pirámide de pruebas.

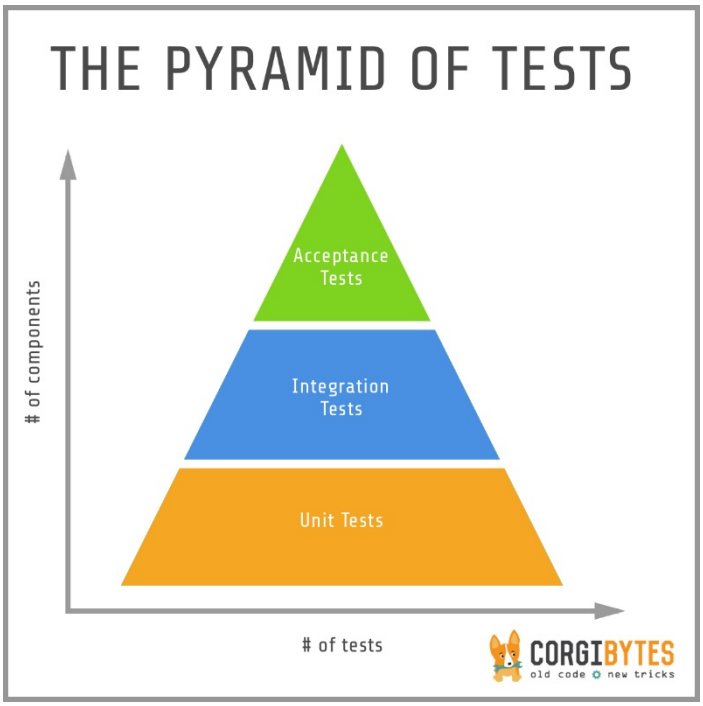

Figura 2.2 La pirámide de pruebas (https://oldcodenewtricks.com/the-pyramid-of-tests/)

2.2.2 Tipos de pruebas

Las pruebas pueden aplicarse a varios niveles, esas pruebas pueden ser de los siguientes tipos:

Pruebas funcionales: se basa en funciones y prestaciones descritas en documentos o entendidas por los probadores, pueden llevarse a cabo en todos los niveles.

Pruebas no funcionales: Incluyen pruebas de rendimiento, pruebas de carga, estrés, usabilidad, mantenibilidad. (ISQTB, 2010, pp. 31)

2.3 Automatización de pruebas de software

Las pruebas de software son caras y requieren mucho trabajo. Las pruebas de software requieren hasta 50% de los costos de desarrollo de software, y aún más para aplicaciones críticas.

Uno de los objetivos de las pruebas de software es automatizar tanto como sea posible, reduciendo significativamente su costo, minimizando el error humano y haciendo pruebas de regresión más fácil.

La automatización de tareas especiales sirve al ingeniero de pruebas de muchas maneras. Primero, eliminar las tareas especiales, elimina el trabajo pesado, lo que hace que el trabajo de los ingenieros de prueba sea más satisfactorio.

Segundo, la automatización libera tiempo para concentrarse en las partes divertidas y desafiantes de las pruebas.

Tercero, la automatización puede ayudar a eliminar errores de omisión.

La automatización elimina parte de la variación en la calidad de la prueba causada por diferencias en habilidades individuales.

Muchas tareas de prueba que desafiaron la automatización en el pasado ahora son candidatos para tal tratamiento debido a los avances tecnológicos. Por ejemplo, generar casos de prueba que satisfaga ciertos requisitos de prueba dados suele ser un problema difícil que requiere intervención del ingeniero de pruebas. Sin embargo, existen herramientas, tanto de investigación como comerciales, que automatizan esta tarea en diversos grados. (Ammann & Offutt, 2008, pp 10)

2.3.1 Que son las pruebas automatizadas de software

Pruebas automatizadas de software significa utilizar una herramienta de automatización para ejecutar su conjunto de casos de prueba. Por el contrario, la prueba manual es realizada por un humano sentado frente a una computadora que ejecuta cuidadosamente los pasos de la prueba.

El software de automatización también puede ingresar datos de prueba en el Sistema bajo prueba, comparar resultados esperados y reales y generar informes de prueba detallados

Los ciclos de desarrollo sucesivos requerirán la ejecución del mismo conjunto de pruebas repetidamente. Con una herramienta de automatización de prueba, es posible grabar este conjunto de pruebas y reproducirlo según sea necesario. Una vez que el conjunto de pruebas está automatizado, no se requiere intervención humana. El objetivo de la automatización es reducir el número de casos de prueba que se ejecutarán manualmente y no eliminar por completo las pruebas manuales.

2.3.2 Framework de automatización de pruebas

Conjunto de herramientas que se establece para proporcionar un entorno de ejecución para los scripts de prueba de automatización. El framework proporciona al usuario varios beneficios que lo ayudan a desarrollar, ejecutar e informar los scripts de prueba de automatización de manera eficiente. Es más, como un sistema que se ha creado específicamente para automatizar nuestras pruebas.

En un lenguaje muy simple, podemos decir que un framework es una combinación constructiva de varias pautas, estándares de codificación, conceptos, procesos, prácticas, jerarquías de proyectos, modularidad, mecanismo de informes, inyecciones de datos de prueba, etc. para pruebas de automatización de pruebas. Por lo tanto, el usuario puede seguir estas pautas mientras automatiza la aplicación para aprovechar los diversos resultados productivos.

Las ventajas pueden estar en diferentes formas, como la facilidad de scripting, escalabilidad, modularidad, comprensibilidad, definición del proceso, reutilización, costo, mantenimiento, etc.

2.3.3 Arquitectura general de framework de automatización de pruebas

En la automatización de pruebas es necesario tener ciertos mecanismos para crear pruebas, ejecutarlas automáticamente, configurar los datos de entrada, visualización de reportes y otros.

Es por esto que existen ciertos componentes generales que son parte de la arquitectura de un framework de automatización de pruebas.

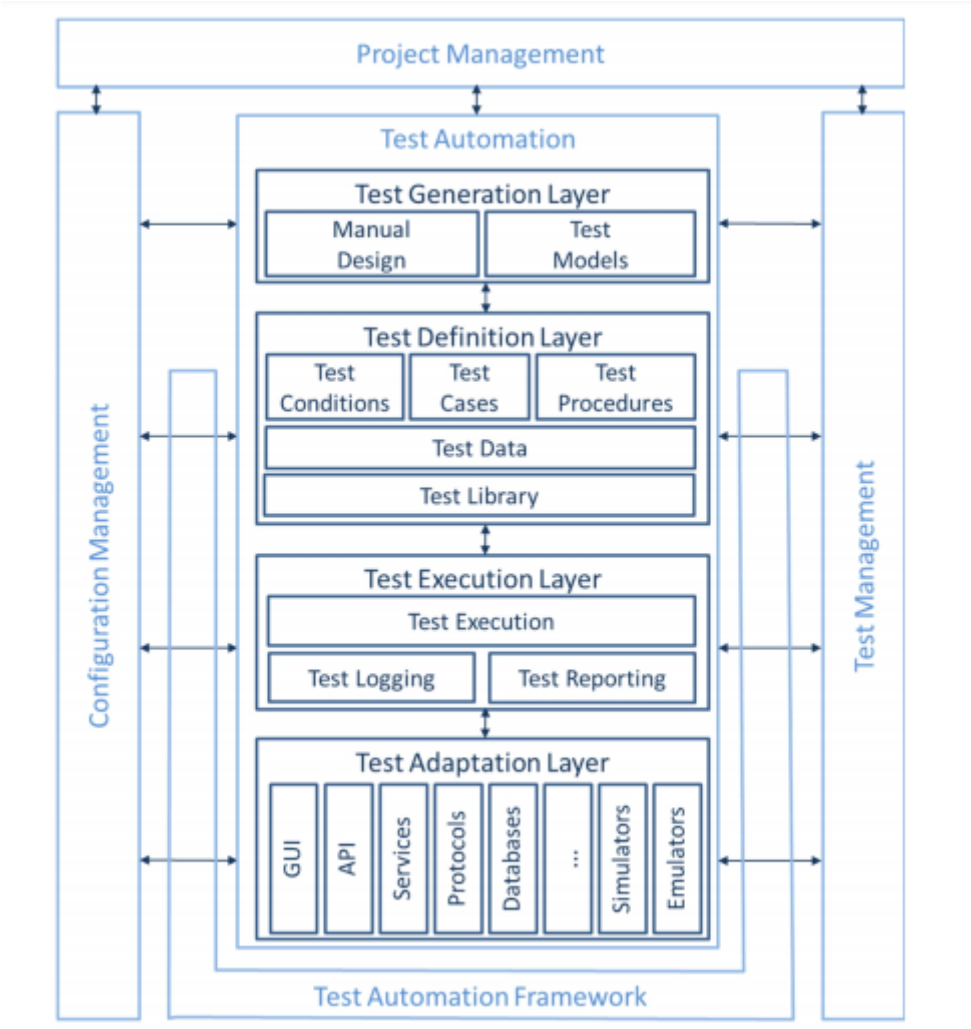

Figura 2.3 Arquitectura genérica de un framework de automatización (ISQTB, 2016, pp. 25)

Capa de generación de pruebas: La capa de generación de prueba consiste en el soporte de herramientas para lo siguiente:

- Diseño manual de casos de prueba.
- Desarrollar, capturar o derivar datos de prueba.
- Generar automáticamente casos de prueba a partir de modelos que definen el SUT y / o su entorno (es decir, pruebas automatizadas basadas en modelos)

Los componentes de esta capa se utilizan para:

- Editar y navegar por las estructuras del conjunto de pruebas
- Relacionar casos de prueba con objetivos de prueba o requisitos SUT
- Documentar el diseño de la prueba.

Para la generación automatizada de pruebas, también se pueden incluir las siguientes capacidades:

- Capacidad para modelar el SUT, su entorno y / o el sistema de prueba.
- Capacidad para definir directivas de prueba y para configurar / parametrizar algoritmos de generación de prueba.
- Capacidad para rastrear las pruebas generadas hasta el modelo (elementos).

Capa de generación de pruebas: La capa de definición de prueba consiste en el soporte de herramientas para lo siguiente:

- Especificar casos de prueba (en un nivel alto y / o bajo).
- Definir datos de prueba para casos de prueba de bajo nivel.
- Especificar procedimientos de prueba para un caso de prueba o un conjunto de casos de prueba.
- Definición de scripts de prueba para la ejecución de los casos de prueba.

- Proporcionar acceso a las bibliotecas de prueba según sea necesario (por ejemplo, en enfoques basados en palabras clave).

Los componentes de esta capa se utilizan para:

- Particionar / restringir, parametrizar o instanciar datos de prueba.
- Especificar secuencias de prueba o comportamientos de prueba completos (incluidas las declaraciones de control y expresiones), para parametrizar y / o agruparlos.
- Documentar los datos de prueba, casos de prueba y / o procedimientos de prueba.

Capa de ejecución de pruebas: La capa de ejecución de prueba consiste en el soporte de herramientas para lo siguiente:

- Ejecución de casos de prueba automáticamente.
- Registro de las ejecuciones de casos de prueba.
- Informar los resultados de la prueba.

La capa de ejecución de prueba puede constar de componentes que proporcionan las siguientes capacidades:

- Configurar y desplegar el SUT para la ejecución de la prueba.
- Configurar y lanzar conjuntos de pruebas (es decir, conjunto de casos de prueba, incluidos los datos de prueba).
- Configurar y parametrizar la ejecución de pruebas.
- Interpretar los datos de prueba y los casos de prueba y transformarlos en scripts ejecutables.
- Configurar el sistema de prueba y / o el SUT para el registro (filtrado) de la ejecución de las pruebas falladas.
- Analizar las respuestas SUT durante la ejecución de la prueba para dirigir las ejecuciones de prueba posteriores.

- Validar las respuestas SUT (comparación de resultados esperados y reales) para caso de prueba automatizado.

Capa de adaptación de pruebas: La capa de adaptación de prueba consiste en el soporte de herramientas para lo siguiente:

- Control del arnés de prueba.
- Interactuar con el SUT.
- Monitorear del SUT.
- Simular o emular el entorno SUT.

La capa de adaptación de prueba proporciona la siguiente funcionalidad:

- Mediación entre las definiciones de prueba tecnológicamente neutrales y los requisitos tecnológicos específicos del SUT y los dispositivos de prueba.
- Aplicar diferentes adaptadores específicos de tecnología para interactuar con el SUT.
- Distribuir la ejecución de la prueba en múltiples dispositivos de prueba / interfaces de prueba o ejecutar pruebas localmente. (ISQTB, 2016, pp. 26 - 27)

CAPITULO III

MODELO TEORICO

El modelo teórico, como idealización que hace el hombre del objeto de investigación, para el esclarecimiento de la situación polémica, que se resuelve en el proceso de la investigación científica, se convierte en el instrumento fundamental para la optimización de sus actividades científicas.

El modelo es una representación ideal del objeto a investigar, donde el sujeto abstrae todos aquellos elementos y relaciones que él considera esenciales y los sistematiza, en el objeto modelado. (Álvarez, 2016, pp. 142)

3.1 Enfoque para el modelado teórico

El enfoque usado para el modelado teórico es el estructural funcional, para hacer énfasis al objeto de estudio, identificar sus elementos y relacionarlos.

El enfoque estructural funcional a tiende a la estructura del objeto, dado por el conjunto ordenado de elementos y relaciones del mismo, encaminado a lograr su

funcionamiento; estableciéndose vínculos estructura-propiedad, estructura-funcionamiento. (Álvarez, 2016, pp. 146)

3.2 Definición de componentes

El objeto de estudio tiene varios elementos y relaciones con otros campos, se describirá los elementos y relaciones esenciales que sean de utilidad para llegar a la solución del problema.

A continuación, se presentan los principales elementos propios del objeto de estudio (calidad de software).

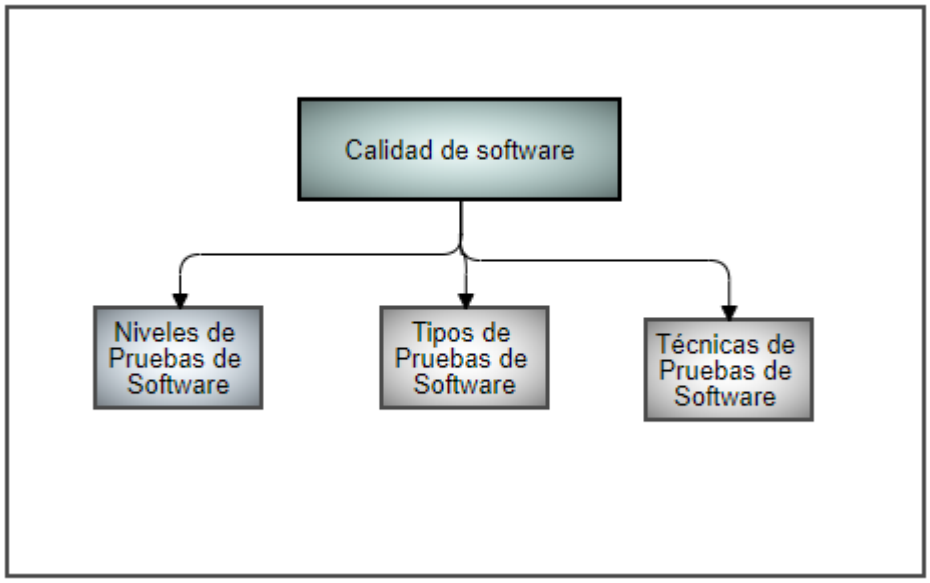

Figura 3.1 Elementos del objeto de estudio (Elaboración propia)

3.2.1 Calidad de software

La calidad de software se define como:

“Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan”.

3.2.2 Niveles de prueba

Los niveles de pruebas son conjunto de pruebas con diferentes destinos u/objetivos, en fases o niveles diferentes que se aplican en diferentes etapas del desarrollo, las cuales de unidad, integración, sistema y aceptación.

3.2.3 Tipos de prueba

Los tipos de prueba se dividen funcionales y no funcionales, estos pueden aplicarse a diferentes niveles de prueba, los funcionales están relacionados a que debe realizar el sistema, mientras los funcionales a como, es decir relacionados a rendimiento, usabilidad y otros.

3.2.4 Técnicas de pruebas

Son un conjunto de técnicas útiles a la hora de diseñar pruebas, dentro de estas existe las técnicas de caja blanca, en los cuales se debe contar con el código fuente, y las técnicas de caja blanca, donde no se tiene acceso al código.

A continuación, se presentan los principales elementos propios del campo de acción (automatización de pruebas web multiplataforma).

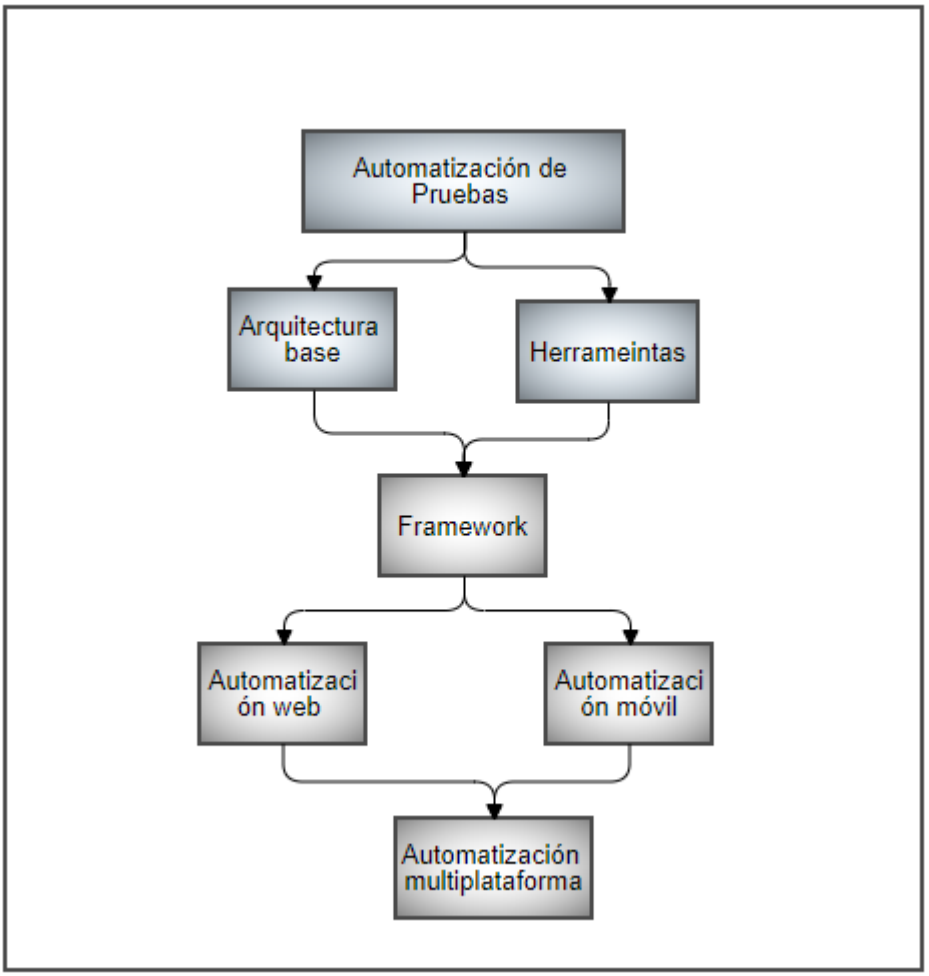

Figura 3.2 Elementos del campo de acción (Elaboración propia)

3.2.5 Automatización de pruebas

La automatización de pruebas reduce el trabajo manual en proyectos de software, consiste en automatizar los procedimientos que un ingeniero de calidad de software realiza, tareas repetitivas, complejas, de esta manera su ejecución no requiere de intervención humana.

3.2.6 Arquitectura

Es el conjunto de elementos relaciones que componen un framework de automatización de pruebas, una arquitectura base genérica provee los mecanismos de desarrollo, ejecución, monitoreo y reportes de pruebas.

3.2.7 Herramientas

Son aplicaciones, librerías, plugins y otros elementos necesarios para el desarrollo e implementación de un framework de automatización de pruebas de software.

3.2.8 Automatización web

La automatización de pruebas web está enfocadas a navegadores de escritorio como ser Chrome, Firefox y otros, su objetivo es automatizar pruebas de software para aplicaciones web.

3.2.9 Automatización móvil

La automatización móvil está enfocada a dispositivos móviles y sus sistemas operativos como Android/iOS, su objetivo es automatizar pruebas de software para aplicaciones web móviles o nativas.

3.2.10 Automatización multiplataforma

La automatización multiplataforma está enfocada a cubrir pruebas de software en distintas plataformas, es decir plataformas web y plataformas móviles.

3.3 Representación del modelo teórico

Para lograr la calidad de software y dentro de esta área se usan muchas técnicas, como también herramientas, la relación que existe entre el objeto de estudio (calidad de software) y el campo de acción (automatización de pruebas de software) se hace evidente al establecer las relaciones de componentes detallados en el siguiente diagrama:

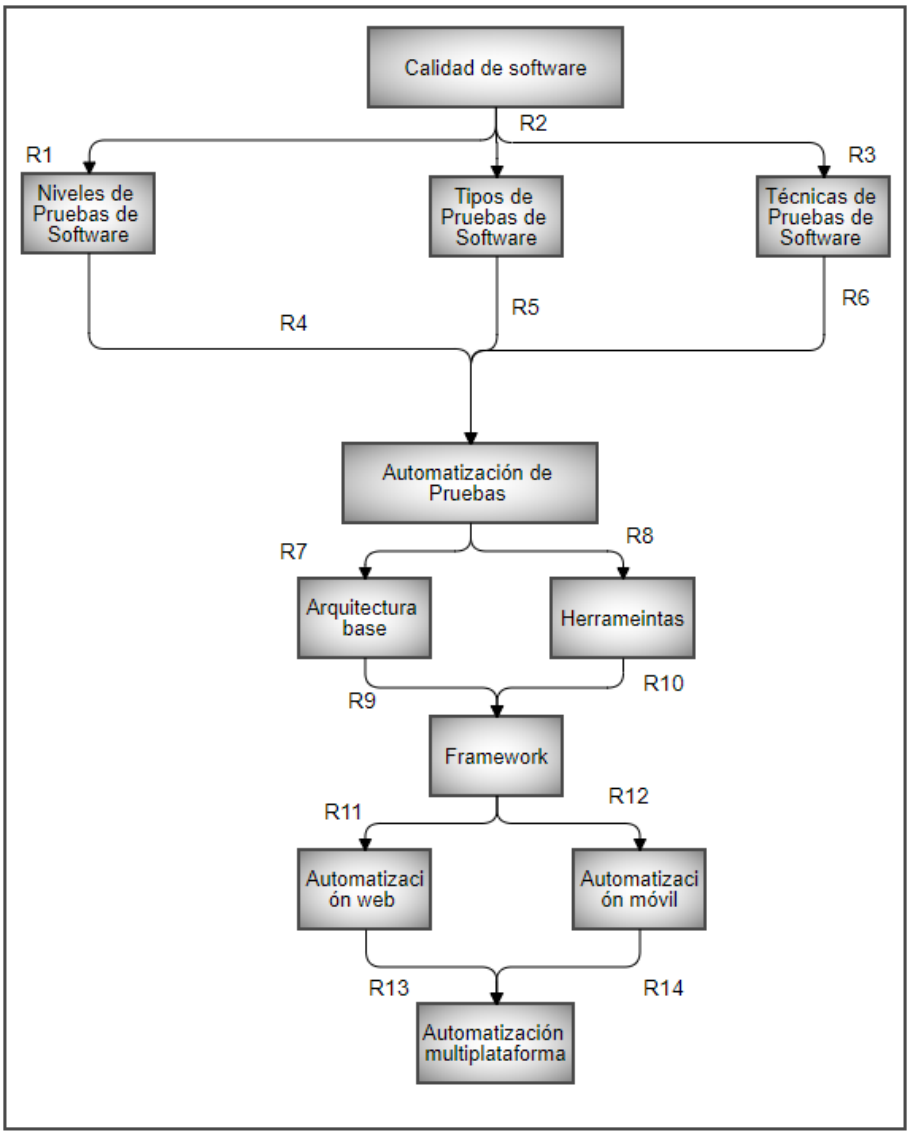

Figura 3.3 Representación gráfica del modelo teórico (Elaboración propia)

En resumen, se puede decir que dentro de calidad de software se aplican muchas técnicas y tipos de pruebas en varios niveles, que pueden ser cubiertos por la automatización de pruebas, que requiere contar con framework que es construido a partir de una arquitectura base y herramientas que están orientados a ciertos tipos de aplicaciones, como ser web y móviles, al cubrir pruebas para ambas preformas se habla de automatización multiplataforma.

3.4 Relación entre componentes

En el modelo teórico se puede apreciar todos los elementos y relaciones entre ellos que son necesarias para bordar el estudio y llegar a la solución,

Se describirán todas las relaciones que existen entre los principales elementos del objeto de estudio y sus relaciones con el campo de acción.

3.4.1 Relación R1: Calidad de software – niveles de pruebas de software

Dentro del área de calidad de software y para lograr la calidad de software se ejecutan pruebas a en varios niveles y etapas del desarrollo de software, para garantizar la calidad de software es necesario ejecutar pruebas en cada nivel: unidad, integración, sistema, aceptación.

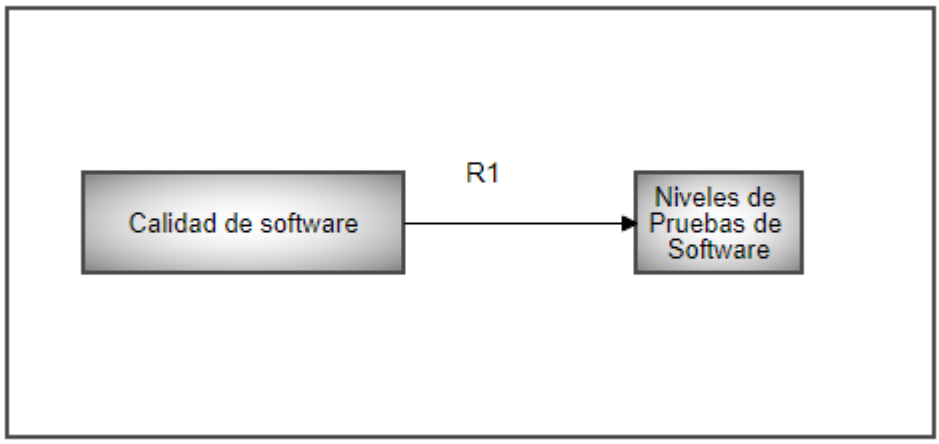

Figura 3.4 Relación 1 (Elaboración propia)

3.4.2 Relación R2: Calidad de software – tipos de pruebas de software

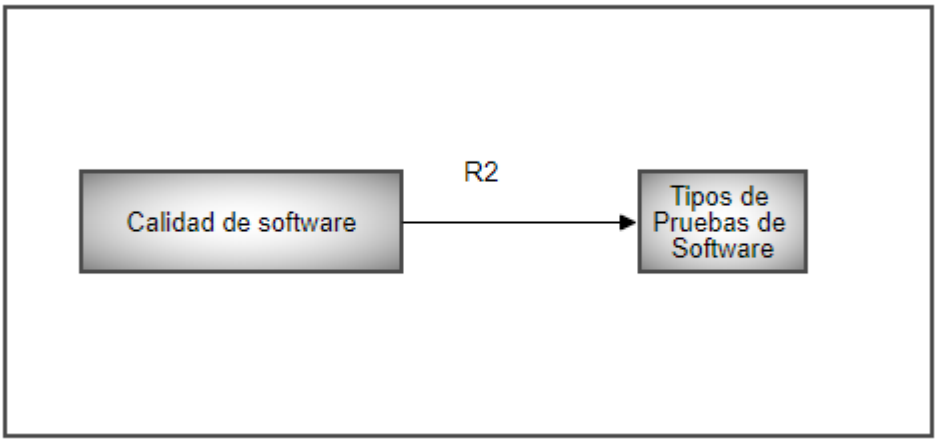

Figura 3.5 Relación 2 (Elaboración propia)

Para lograr cierta confianza acerca la calidad de software se ejecutan 2 tipos de pruebas, funcionales y no funcionales, normalmente estas pruebas se obtienen de los requerimientos de usuario, las pruebas funcionales van enfocadas a lo que el software hace, y las pruebas no funcionales a como lo hace.

3.4.3 Relación R3: Calidad de software – técnicas de pruebas de software

Según la cantidad y la eficacia de las pruebas ejecutadas, se puede obtener cierto grado de confianza acerca la calidad de software, esas pruebas necesitan ser diseñadas con alguna técnica, en función también del propósito de las pruebas, existen 2 grupos de técnicas las de caja negra y las de caja blanca.

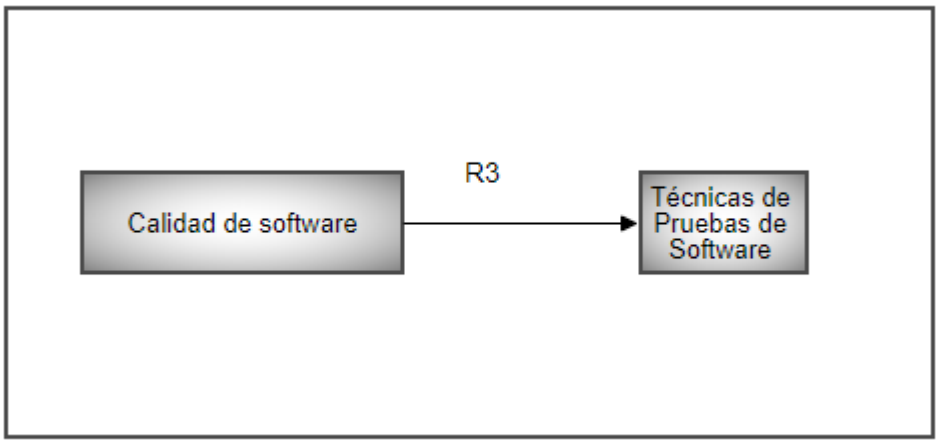

Figura 3.6 Relación 3 (Elaboración propia)

3.4.4 Relación R4-R5-R6: Automatización de pruebas – niveles, tipos, técnicas de pruebas de software.

La automatización de pruebas puede ser aplicadas en distintos niveles de pruebas de software, también en diferentes tipos, se puede tener pruebas automatizadas funcionales y no funcionales.

Las pruebas automatizadas pueden ser aplicadas a pruebas aplicando técnicas de caja blanca o técnicas de caja negra.

En síntesis, la automatización de pruebas puede ser aplicada a todos los niveles, tipos y técnicas de pruebas de software.

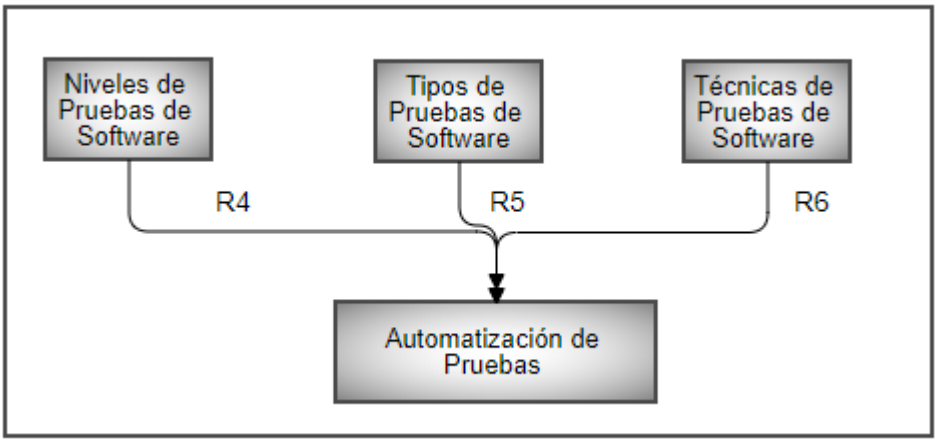

Figura 3.7 Relación 4-5-6 (Elaboración propia)

3.4.5 Relación R7-R8: Automatización de pruebas – arquitectura base, herramientas.

Cuando se desea implementar la automatización de pruebas, se debe realizar un análisis y diseño base que responda a las necesidades de automatización del proyecto como también las herramientas necesarias para lograr implementar pruebas de software automatizadas.

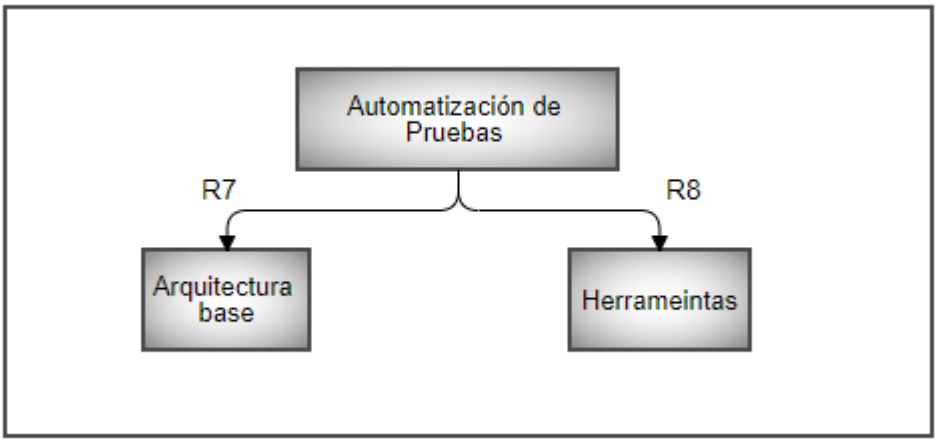

Figura 3.8 Relación 7-8 (Elaboración propia)

3.4.6 Relación R9-R10: Framework – arquitectura base, herramientas

Para tener un framework de automatización de pruebas de software, se debe tener una arquitectura base y construir el framework basado en la arquitectura planteada haciendo uso de herramientas que faciliten el desarrollo o que aporten nuevas funcionalidades al framework.

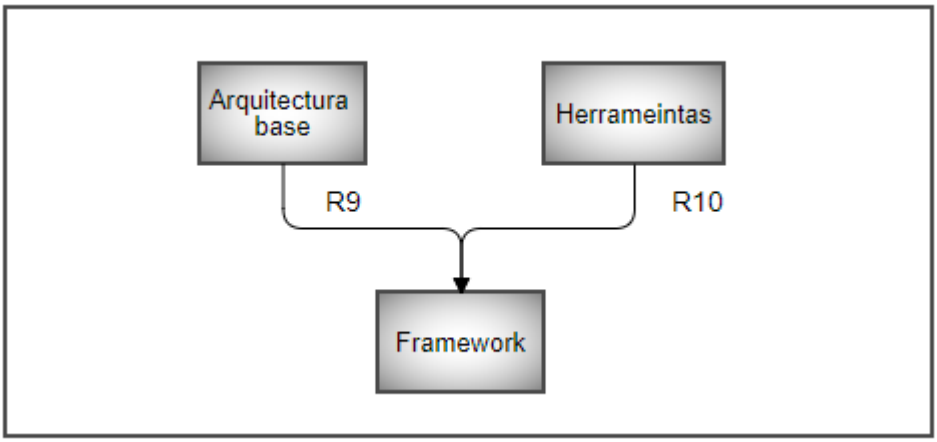

Figura 3.9 Relación 9-10 (Elaboración propia)

3.4.7 Relación R11-R12: Framework – automatización web, automatización móvil

Un framework tiene el propósito de brindar da capacidad de implementar y ejecutar pruebas automatizadas de software, estas pueden tener como objetivo ordenadores de escritorio o dispositivos móviles, un framework puede estar diseñado para ejecutar pruebas a aplicaciones web en escritorio o en dispositivos móviles.

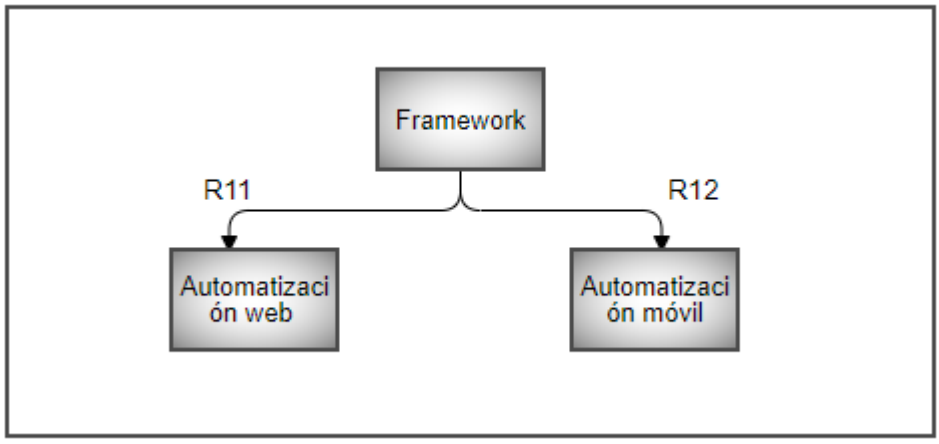

Figura 3.10 Relación 11-12 (Elaboración propia)

3.4.8 Relación R13-R14: Automatización multiplataforma – automatización web, automatización móvil

Al referirnos que un solo framework es capaz de cubrir pruebas de software automatizadas para web en ordenadores de escritorio, y para dispositivos móviles, podemos decir que hablamos de un framework de automatización multiplataforma.

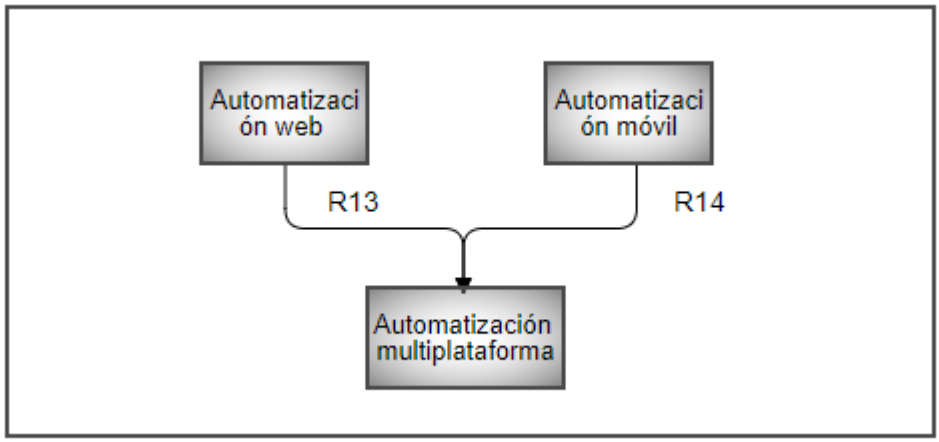

Figura 3.11 Relación 13-14 (Elaboración propia)

3.5 Conclusiones

A partir de todos los elementos identificados y sus relaciones en el modelo teórico se obtiene un aporte teórico en cuanto a las pruebas automatizadas en el objeto de estudio.

Se pudo mostrar la relación de las pruebas automatizadas en calidad de software que ayudó a generar nuevo conocimiento en torno al objeto de estudio.

CAPITULO IV

PROPUESTA

En este capítulo se detalla la implementación de una propuesta para solucionar los problemas identificados relacionados a la automatización de pruebas web multiplataforma.

En términos generales se identificó que el principal problema es que crear y mantener pruebas automatizadas para plataformas web de escritorio y móviles requiere mucho más esfuerzo y en algunos casos conocimiento, por esta razón muchos proyectos que tienen suporte multiplataforma solo implementan pruebas automatizadas para una de las plataformas.

El esfuerzo requerido tiende a ser alto debido a que usan 2 herramientas distintas y se deben implementar 2 frameworks, la propuesta se enfoca en proveer un “modelo de framework para la automatización de aplicaciones web multiplataforma” que permite implementar pruebas automatizadas para ambas plataformas, reduciendo el número de líneas de código necesarias y por lo tanto el esfuerzo.

4.1 Objetivos de la propuesta

Como objetivos de la propuesta se tiene:

- Generar e implementar un “framework” para la automatización de pruebas a aplicaciones web multiplataforma, combinando tecnologías de escritorio y móviles.

- A partir de la implementación del framework, se busca reducir el esfuerzo y líneas de código en desarrollo de pruebas.
- Proveer las capacidades estándar que los frameworks de automatización de pruebas de software proveen.

4.2 Metodología

La metodología usada para el desarrollo de la solución fue la metodología cascada que está compuesta por las siguientes fases:

- **Requisitos:** Se hace el análisis de las necesidades para determinar las características del software a desarrollar.
- **Diseño:** Se describe la estructura interna del software, y las relaciones entre las entidades que lo componen.
- Implementación: En esta fase se programan los requisitos especificados.
- **Verificación:** Como su propio nombre lo indica, una vez terminada la fase de implementación, se verifica que todos los componentes del sistema funcionen correctamente y cumplen con los requisitos.
- **Mantenimiento:** Una vez terminado el desarrollo, las actividades posteriores están enfocadas a tareas de mantenimiento.

Figura 4.1 Metodología de desarrollo de software cascada (https://aspgems.com/metodologia-de-desarrollo-de-software-i-modelo-en-cascada/)

4.3 Análisis

El análisis tiene como objetivo capturar y realizar una especificación precisa de los requerimientos del framework y representarlos de manera adecuada para.

Para especificar estos requerimientos de manera precisa y sin ambigüedad se los representa con diagramas de casos de uso del Lenguaje de Modelamiento Unificado (UML). (Jacobson, Booch, & Rumbaugh, 2000, pp. 165)

4.3.1 Requerimientos funcionales

Los requerimientos funcionales de un sistema describen la funcionalidad o los servicios que se espera que éste provea. Estos dependen del tipo de software y del sistema que se desarrolle y de los posibles usuarios del software. El desarrollador recolecta todos los requerimientos del usuario, y de manera funcional los implementa para así construir el sistema. (Jacobson, Booch, & Rumbaugh, 2000, pp. 105)

A continuación, se detalla los requerimientos funcionales para el framework de automatización de pruebas web multiplataforma.

RF1: El framework debe ser capaz de ejecutar pruebas automatizadas web en navegadores (Firefox, Chrome) en ordenadores de escritorio.

RF2: El framework debe ser capaz de ejecutar pruebas automatizadas web en el navegador Chrome para Android.

RF3: El framework debe tener integración con la herramienta con Cucumber, debe permitir usar esta herramienta para representar en texto plano las pruebas de software.

RF4: El framework debe manejar definiciones de páginas de las interfaces de usuario independientes para cada plataforma (web y móvil).

RF5: El framework debe manejar definiciones de objetos de la interfaz de usuario de forma independiente para cada plataforma (web y móvil).

RF6: El framework debe permitir configurar el navegador a ser usado desde un archivo de propiedades para el caso de web en ordenadores.

RF7: El framework debe permitir configurar los parámetros necesarios para correr pruebas en Android (package, activity, apk, deviceName) en un archivo de propiedades.

RF8: El framework debe permitir configurar la dirección del servidor Appium desde un archivo de propiedades.

RF9: El framework debe tener un ejecutor de pruebas.

RF10: El framework debe permitir manejar localizadores de elementos de interfaz de usuario independientes a la plataforma.

RF11: El framework debe permitir proveer los datos de prueba desde archivos YAML.

RF12: El framework debe generar reportes de pruebas ejecutadas.

RF13: Los métodos que exponen las API de Selenium deben estar disponibles para ser usadas en el caso de automatización de pruebas para ordenadores de escritorio.

RF14: Los métodos que exponen las API de Appium deben estar disponibles para ser usadas en el caso de automatización de pruebas para dispositivos móviles.

4.3.2 Requerimientos no funcionales

Estos, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. (Jacobson, Booch, & Rumbaugh, 2000, pp. 110)

RNF1: El framework debe permitir crear pruebas usando el patrón página – objeto.

RNF2: El framework debe tener tiempos de respuesta similares a los que tiene Selenium en el caso de automatización de pruebas para ordenadores de escritorio.

RNF3: El framework debe tener tiempos de respuesta similares a los que tiene Appium en el caso de automatización de pruebas para dispositivos móviles.

4.3.3 Especificación de casos de uso

Los diagramas de Casos de Uso muestran las relaciones estructurales entre los actores y los casos de uso del sistema, es decir, representan la funcionalidad del sistema. Un modelo de casos de uso es un modelo del sistema que contiene: Actores, Casos de Uso, y sus relaciones. (Jacobson, Booch, & Rumbaugh, 2000, pp. 125)

4.3.3.1 Actor

Los actores son entidades externas que interaccionan con el sistema participando en los casos de uso. Representan los papeles que distintos usuarios pueden jugar, cada uno de los usuarios se representa por uno o más actores. (Jacobson, Booch, & Rumbaugh, 2000, pp. 128)

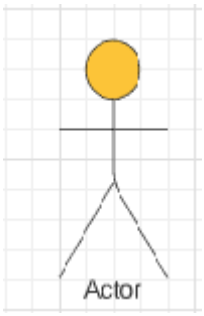

Figura 4.2 Actor en UML (Elaboración propia)

4.3.3.2 Casos de uso

Cada forma en que los actores usan el sistema se representa con un caso de uso, entonces los “Caso de Uso” son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. (Jacobson, Booch, & Rumbaugh, 2000, pp. 129)

Figura 4.3 Caso de en UML (Elaboración propia)

4.3.3.2.1 Caso de uso: Crear pruebas

El framework provee la funcionalidad para crear pruebas automatizadas que requieren crear definiciones de páginas, definiciones objetos de interfaz de usuario y datos de prueba.

La crear los objetos de interfaz de usuario es necesario determinar localizadores para poder identificarlos.

Al crear páginas y determinar localizadores, se debe determinar una plataforma destino, sea web de ordenador o móvil.

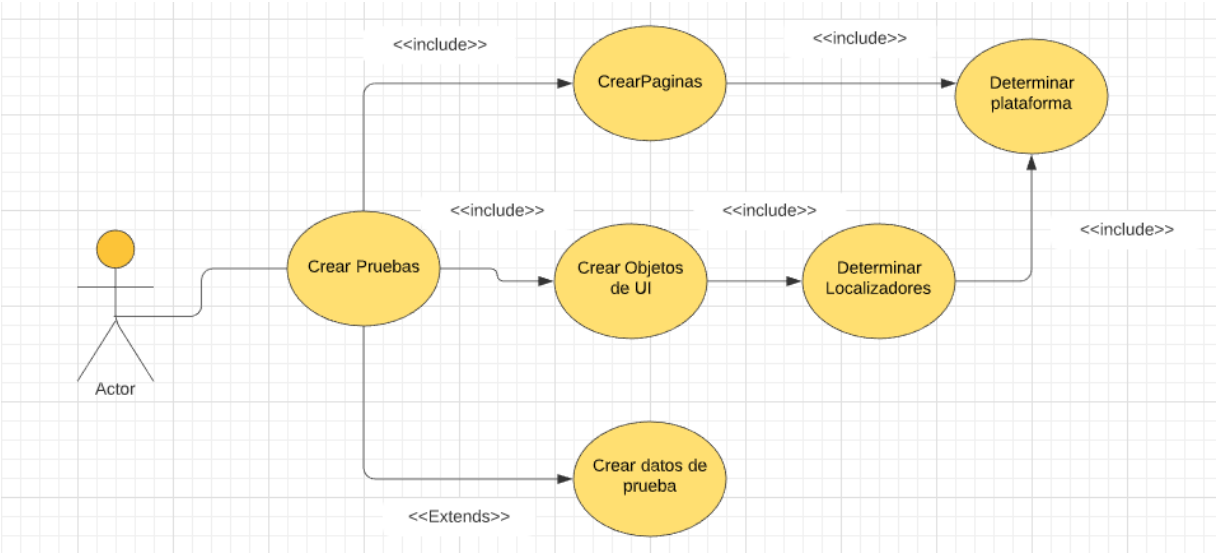

Figura 4.4 Caso de uso – crear pruebas (Elaboración propia)

4.3.3.2.1 Caso de uso: Correr pruebas

Para correr las pruebas creadas se debe tener configurado las propiedades del entorno donde se enjutarán las pruebas, un reporte puede ser enviado como parte de la ejecución de pruebas.

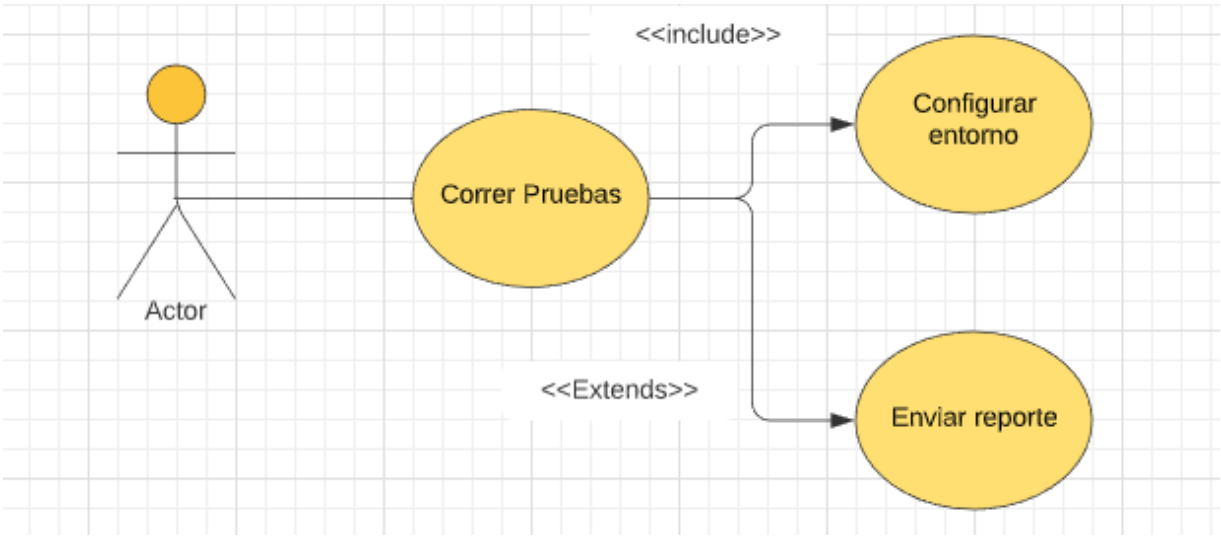

Figura 4.5 Caso de uso – crear pruebas (Elaboración propia)

4.3.4 Paquetes de análisis

El propósito es identificar los paquetes de análisis, que sirve para organizar el modelo de análisis en piezas más pequeñas y manejables. Para identificar los paquetes de análisis, nos basaremos en los requerimientos funcionales y el dominio del problema. Los paquetes son el resultado de asignar la mayor parte de los casos de uso, desde el punto de vista de cada uno de los actores del sistema. (Jacobson, Booch, & Rumbaugh, 2000, pp. 181)

A continuación de se detallan los paquetes identificados y sus relaciones:

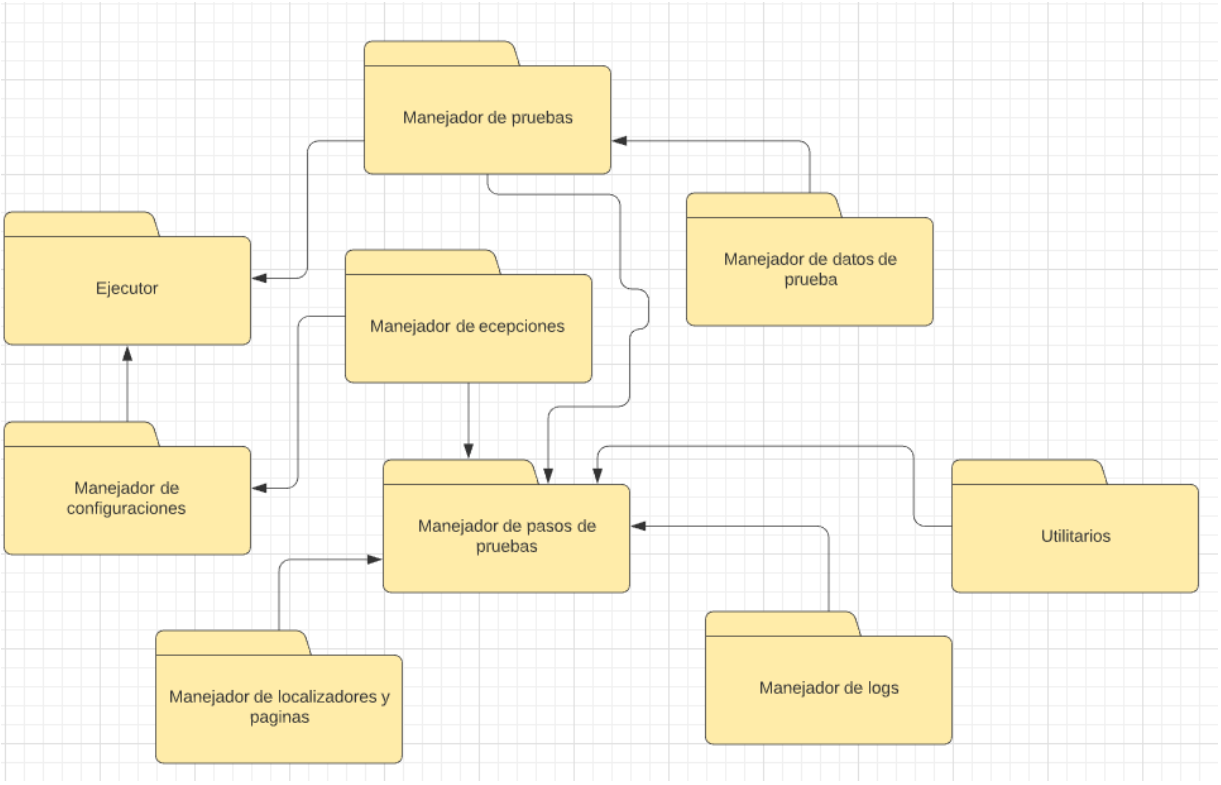

Figura 4.6 Paquetes – framework de automatización (Elaboración propia)

4.4 Diseño

El propósito del diseño de sistema es la transformación de la especificación de requerimientos y análisis para adquirir una comprensión en profundidad de los requerimientos funcionales, no funcionales y las restricciones de lenguajes de programación, Sistemas Operativos, etc. También esta etapa es importante ya que nos sirve para modelar y encontrar su la forma del sistema, lo cual nos ayudara a visualizar la etapa de implementación. Una entrada esencial en el diseño es el resultado del análisis. Esto es, el modelo de análisis Al igual como en el caso del análisis, se utilizará herramientas del lenguaje UML. (Jacobson, Booch, & Rumbaugh, 2000, pp. 205)

4.4.1 Clases de diseño

Una clase de diseño es una abstracción sin costuras de una clase que corresponde a una construcción similar en la implementación del sistema, en otras palabras, el lenguaje utilizado para especificar una clase de diseño es lo mismo que el lenguaje

de programación. En consecuencia, las operaciones, parámetros, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido. (Jacobson, Booch, & Rumbaugh, 2000, pp. 209)

4.4.2 Diagrama de secuencia

El diagrama de secuencia tiene el objetivo de describir el comportamiento dinámico del sistema haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos. A continuación, se muestra los diagramas de secuencia para los casos de uso.

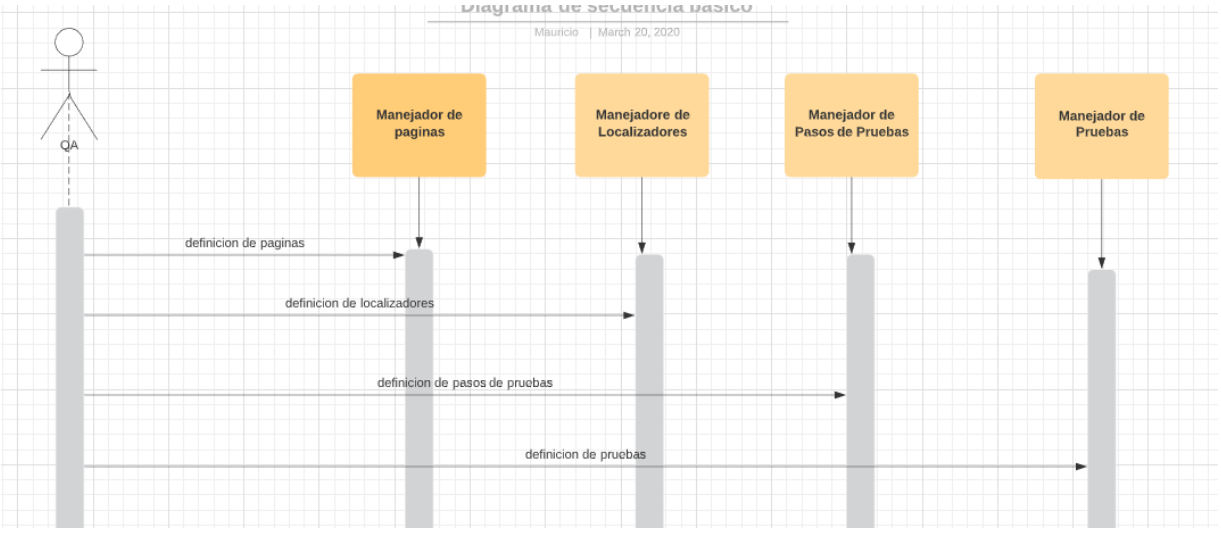

Figura 4.7 Diagrama de secuencia – crear pruebas (Elaboración propia)

Al crear una prueba automatizada de software, es necesario que el ingeniero de calidad de software provea las definiciones de páginas, localizadores, pasos de las pruebas, y las pruebas a ser creadas dentro del framework.

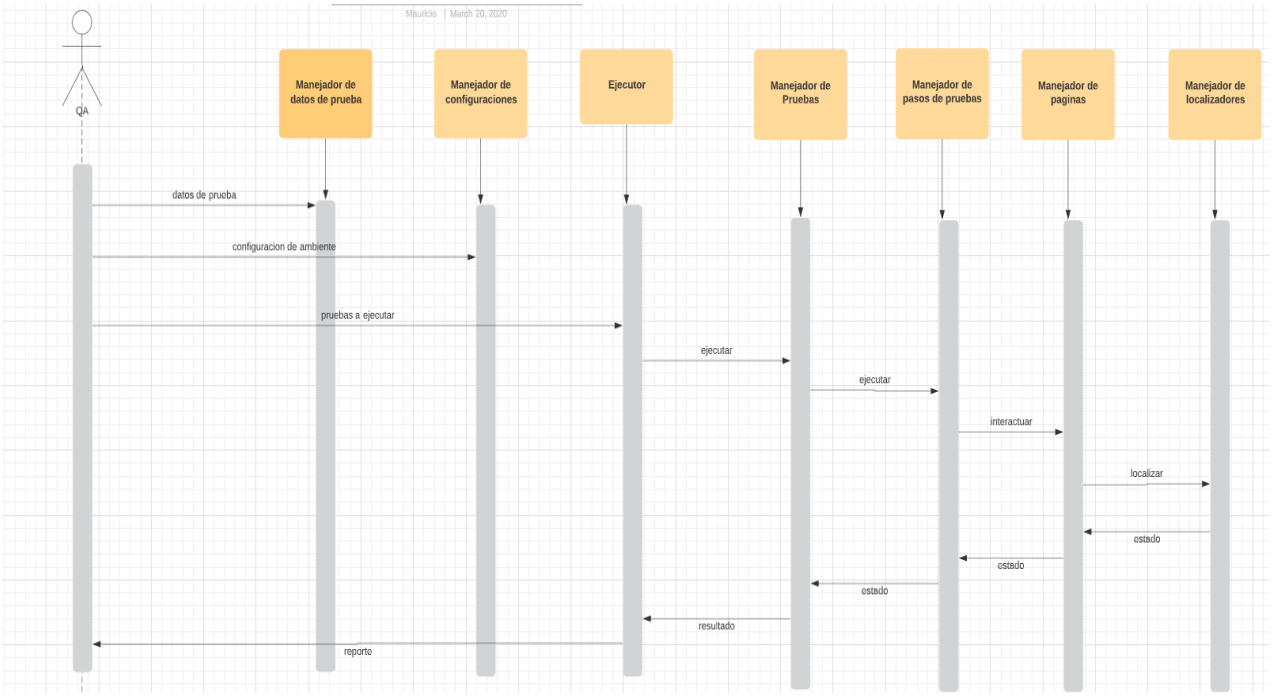

Figura 4.8 Diagrama de secuencia – ejecutar pruebas (Elaboración propia)

Para ejecutar pruebas automatizadas el ingeniero debe proveer los datos de prueba, configuraciones y el conjunto de pruebas as ser ejecutadas, el ejecutor llama a la ejecución de pruebas que se van pasando datos entre componentes del framework y retornando el estado que se ve reflejado en el reporte final de la ejecución.

4.5 Implementación

En la etapa de implementación se construyó el framework para la automatización de pruebas web multiplataforma basado en las especificaciones funcionales y no funcionales para dar solución al problema planteado.

4.5.1 Herramientas de implementación

Para desarrollar el framework de automatización de pruebas web multiplataforma se usó varias herramientas para propósitos específicos.

4.5.1.1 Selenium

Selenium es un proyecto que alberga un abanico de herramientas y librerías que permiten y apoyan la automatización de navegadores web.

Proporciona extensiones que permiten emular las interacciones que realizan los usuarios con los navegadores, un servidor que permite distribuir la asignación de navegadores de forma escalable, y la infraestructura necesaria para las implementaciones de la especificación del WebDriver del W3C, el cual permite escribir código intercambiable para los navegadores web más usados.

Este proyecto es posible gracias a los colaboradores voluntarios, los cuales han dedicado miles de horas de su propio tiempo haciendo así que el código fuente esté disponible de manera gratuita para que cualquiera pueda usarlo, disfrutarlo y mejorarlo.

El corazón de Selenium es el WebDriver, una interfaz que permite escribir conjuntos de instrucciones que se pueden ejecutar de manera indistinta en muchos navegadores.

4.5.1.1.1 Arquitectura de Selenium

Existen 4 componentes básicos de la arquitectura de Selenium:

- Librerías cliente de Selenium
- Protocolo JSON Wire
- Drivers de los navegadores
- Navegadores reales

Librerías cliente de Selenium: Selenium admite varias bibliotecas, como Java, Ruby, Python, etc., los desarrolladores de Selenium han desarrollado enlaces de idiomas para permitir que Selenium admita varios idiomas.

Por ejemplo, si desea usar el driver del navegador en java, use los enlaces de java. Todos los enlaces de idiomas admitidos se pueden descargar desde el sitio web oficial (https://www.seleniumhq.org/download/#client-drivers) de Selenium.

Protocolos JSON Wire: JSON (JavaScript Object Notation) es un estándar abierto para el intercambio de datos en la web. Es compatible con estructuras de datos como objetos y matriz. Por lo tanto, es fácil escribir y leer datos de JSON. Para obtener más información sobre JSON, visite https://www.javatpoint.com/json-tutorial

JSON Wire Protocol es una API REST que transfiere la información entre el servidor HTTP. Cada BrowserDriver (como FirefoxDriver, ChromeDriver, etc.) tiene su propio servidor HTTP.

JSON Wire Protocol proporciona un mecanismo de transporte para transferir datos entre un servidor y un cliente. JSON Wire Protocol sirve como un estándar de la industria para varios servicios web REST.

Driver del navegador: Selenium utiliza drivers/controladores, específicos para cada navegador para establecer una conexión segura con el navegador sin revelar la lógica interna de la funcionalidad del navegador. El controlador del navegador también es específico del idioma utilizado para la automatización, como Java, C #, etc.

Cuando ejecutamos un script de prueba con WebDriver, las siguientes operaciones se realizan internamente.

- La solicitud HTTP se genera y se envía al controlador del navegador para cada comando de Selenium.
- El controlador recibe la solicitud HTTP a través del servidor HTTP.
- El servidor HTTP decide todos los pasos para realizar las instrucciones que se ejecutan en el navegador.

- El estado de ejecución se envía de vuelta al servidor HTTP, que posteriormente se envía al script de automatización.

Navegadores: Los navegadores soportados por Selenium WebDriver son Internet explorer, Mozilla Firefox, Google Chrome y Safari.

Funcionamiento: Se escribe un código en algún IDE de desarrollo (por ejemplo, Eclipse IDE) utilizando cualquiera de las librerías de cliente que nos ofrece Selenium (por ejemplo, Java).

Una vez que esté listo el script, se lo ejecuta. cada declaración en el script se convertirá como una URL con la ayuda de JSON Wire Protocol sobre HTTP. Las URL se pasarán a los controladores/drivers del navegador.

Cada driver del navegador utiliza un servidor HTTP para recibir solicitudes HTTP. Una vez que la URL llegue al driver del navegador, el driver del navegador pasará esa solicitud al navegador real a través de HTTP. Luego los comandos de su script de Selenium se ejecutarán en el navegador.

Si la solicitud es POST, habrá una acción en el navegador.

Si la solicitud es una solicitud GET, la respuesta correspondiente se generará en el extremo del navegador y se enviará a través de HTTP al controlador del navegador y al controlador del navegador a través del protocolo JSON Wire y se enviará a la interfaz de usuario (Eclipse IDE).

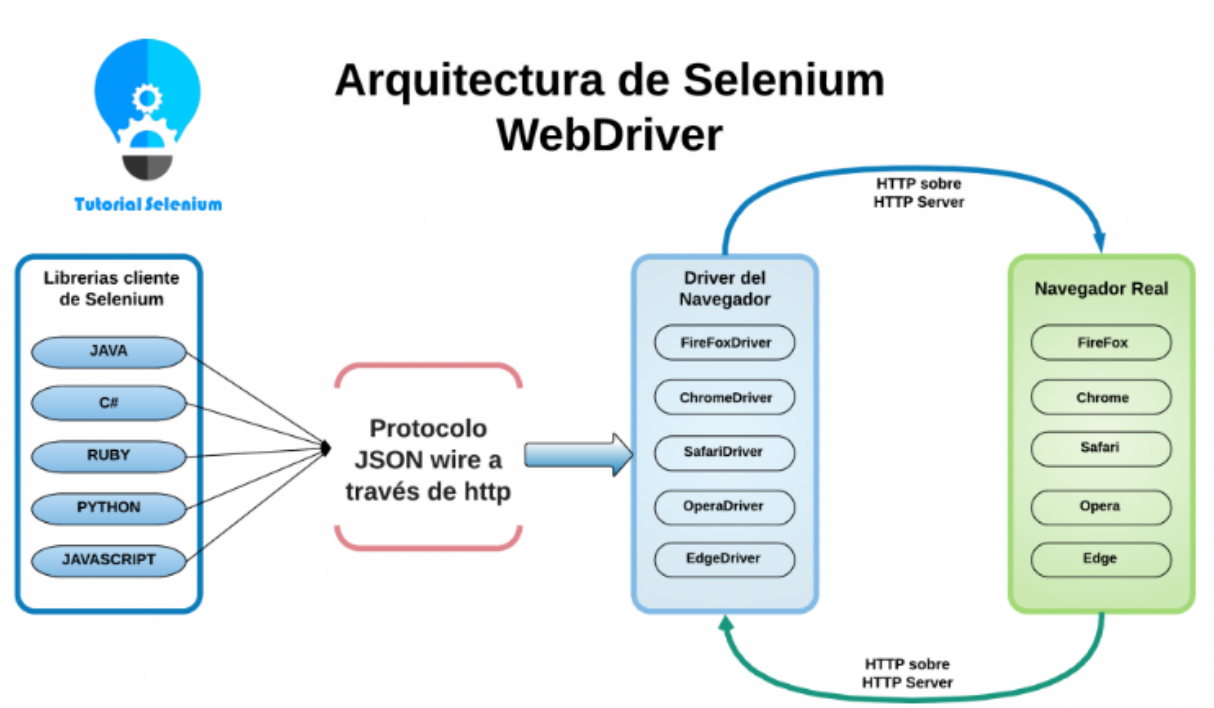

Figura 4.8 Arquitectura de Selenium (https://www.tutorialselenium.com/2018/11/21/arquitectura-selenium-webdriver/)

4.5.1.1.2 Localización de elementos web e interacciones

Básicamente para automatizar una prueba se debe abrir una página web, localizar los elementos, realizar acciones sobre los elementos y verificar condiciones esperadas, esto la estructura básica de una prueba automatizada, Selenium provee métodos para estas acciones.

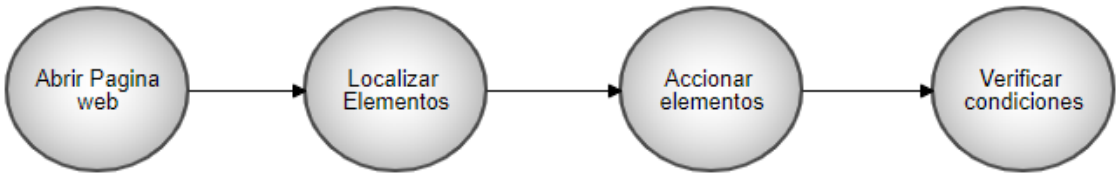

Figura 4.9 Flujo básico de automatización de pruebas (Elaboración propia)

Los métodos básicos que actúan sobre el navegador son:

get(): Método que se encarga de acceder a la url que se le pasa como parámetro y cargar su contenido en el navegador.

getCurrentUrl(): Retorna la página actual donde se encuentra el navegador.

close(): Cierra el navegador utilizado para la prueba automatizada.

Los métodos básicos usados para localizar elementos en la interfaz de usuario son:

findElement(): Devuelve el primer elemento encontrado que coincida con el criterio de búsqueda.

findElements(): Devuelve una lista de elementos encontrados que coincidan con el criterio de búsqueda.

Los criterios de búsqueda en Selenium pueden ser:

- ByClassName
- ByCssSelector
- ById
- ByLinkText
- ByName
- ByPartialLinkText
- ByTagName
- ByXpath

Los métodos básicos para interactuar con los elementos de una página web son:

sendKeys(): Simula la introducción de texto por parte del usuario en una caja de texto.

clear(): Simula el eliminado de texto en una caja de texto.

isSelected(): Permite conocer si un elemento de tipo checkbox esta seleccionado.

isDisplayed(): Método para conocer si un elemento está siendo mostrado en el navegador.

getText(): Obtiene el texto que está siendo mostrado en un determinado elemento web.

isEnabled(): Método para conocer si un elemento web está habilitado.

click(): Método que simula el hacer clic en un elemento web.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

public class TestFindWithGoogle {


    public static void main(String[] args) {

        //WebDriver driver = new HtmlUnitDriver();
        WebDriver driver = new InternetExplorerDriver();
        driver.get("http://www.yahoo.es");
        WebElement element = driver.findElement(By.name("p"));
        //WebElement element = driver.findElement(By.xpath("//input[@name='q']"));
        element.sendKeys("Forza Atleti!");
        element.submit();
        System.out.println("El titulo de mi página es: " + driver.getTitle());
    }

}
```

Figura 4.9 Ejemplo de prueba automatizada en Selenium (Elaboración propia)

4.5.1.2 Appium

Appium es un framework de automatización de pruebas de código abierto que impulsan aplicaciones nativas, móviles e híbridas tanto para iOS y Android utilizando el protocolo WebDriver, es decir, la API de Selenium. Es decir, Appium está basado en Selenium y se usa para probar aplicaciones móviles en lugar de aplicaciones web en navegadores de escritorio.

Al igual que Selenium, las filosofías de Appium incluyen que no debe tener que aprender una lenguaje de programación específico o un framework para poder

escribir o ejecutar pruebas, y las herramientas de automatización deben ser gratuitas.

Appium es importante porque al igual que los diferentes navegadores responden de manera diferente al diseño web, los sistemas operativos móviles presentan las aplicaciones de manera diferente. Además, las aplicaciones nativas, móviles e híbridas funcionan por separado y tienen diferentes propósitos, por lo que requieren diferentes procesos para el diseño, desarrollo y pruebas.

4.5.1.2.1 Arquitectura de Appium

Appium es en esencia un servidor web que expone una API REST. Recibe conexiones de un cliente, escucha los comandos, ejecuta esos comandos en un dispositivo móvil y responde con una respuesta HTTP que representa el resultado de la ejecución del comando. El hecho de que tengamos una arquitectura de cliente / servidor abre muchas posibilidades: podemos escribir nuestro código de prueba en cualquier idioma que tenga una API de cliente http, pero es más fácil usar una de las bibliotecas de cliente de Appium. Podemos poner el servidor en una máquina diferente a la que se ejecutan nuestras pruebas. Podemos escribir un código de prueba y confiar en un servicio en la nube como Sauce Labs para recibir e interpretar los comandos.

Los clientes inician una sesión con un servidor de formas específicas para cada biblioteca, pero todos terminan enviando una solicitud POST /session al servidor, con un objeto JSON llamado el objeto ‘capacidades deseadas’. En este punto, el servidor iniciará la sesión de automatización y responderá con una ID de sesión que se utiliza para enviar más comandos.

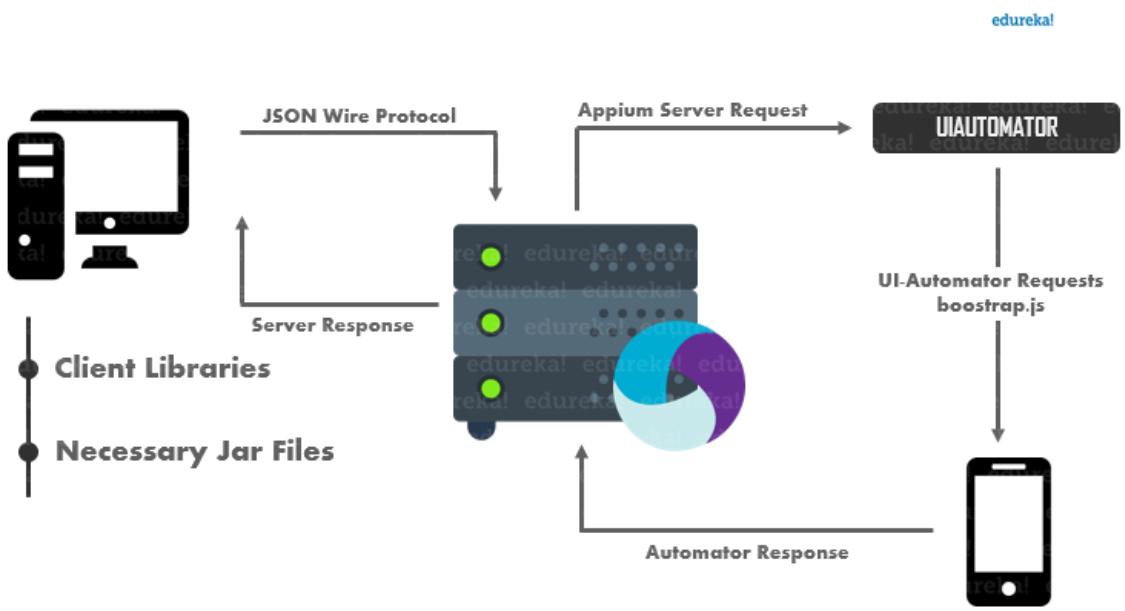

Figura 4.10 Arquitectura de Appium (https://www.edureka.co/blog/appium-architecture/)

4.5.1.2.2 Localización de elementos web e interacciones

Debido a que Appium puede automatizar aplicaciones nativas y web para dispositivos móviles en los localizadores existen estrategias distintas para aplicaciones nativas dependiendo el sistema operativo, debido a que el trabajo investigación está enfocado a aplicaciones web multiplataforma, los localizadores, acciones e interacciones expuestas en Selenium, son válidas para Appium.

4.5.1.3 Cucumber

BDD es uno de los términos de moda en el desarrollo de software en los últimos años. A pesar de ser un término muy utilizado, no todo el mundo sabe exactamente qué es eso de BDD, más allá del significado de esas siglas, Desarrollo Dirigido por Comportamiento (Behaviour Driver Development), ni cómo puede BDD ayudarnos en nuestro trabajo diario como desarrolladores.

Cucumber es una de las herramientas que podemos utilizar para automatizar nuestras pruebas en BDD. Cucumber nos va permitir ejecutar descripciones funcionales en texto plano como pruebas de software automatizadas.

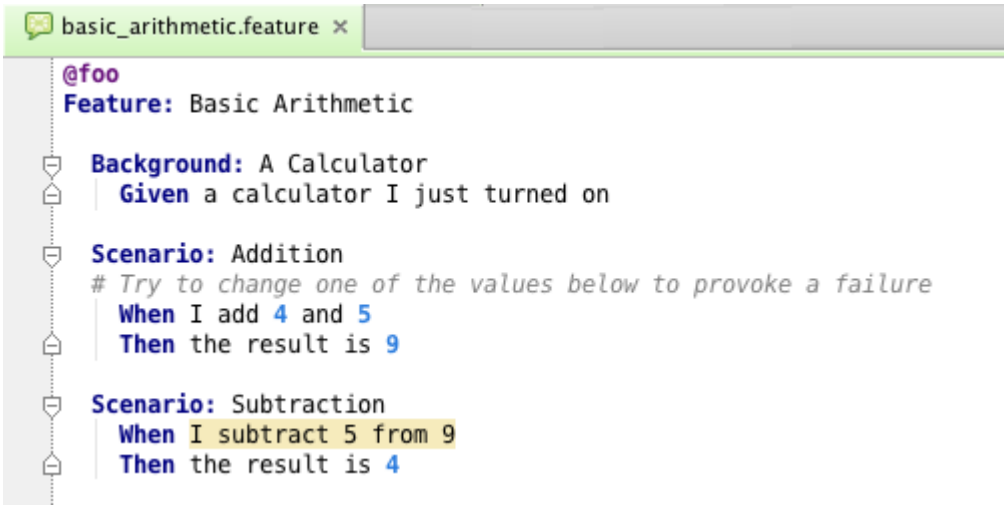

Figura 4.11 Prueba usando Cucumber (Elaboración propia)

4.5.1.4 JUnit

JUnit esun Framework Open Source para la automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

4.5.1.5 Selenide

Selenide es una librería basada en Selenium cuyo objetivo es simplificar el desarrollo de pruebas ofreciendo métodos que permitan resolver de forma efectiva los problemas más comunes que se encuentran durante el desarrollo de un test automatizado, así como también ayudar en el proceso de debug teniendo en cuenta la performance.

En general, lo cual podría ser la mayor distinción, es que Selenium es una librería de bajo nivel (low-level library) para poder interactuar con el browser, Selenide es una librería de alto nivel (high-level library) para desarrollar pruebas automatizadas.

4.5.1.6 AssertJ

AssertJ es un proyecto de código abierto que ha surgido a partir del desaparecido Fest Assert. Es compatible con otras librerías como Guava, Joda Time y Neo4J. También, dispone de un generador automático de comprobaciones para los atributos de las clases, lo que añade más semántica a nuestras clases de prueba.

4.5.2 Framework de automatización de pruebas web multiplataforma

El framework de automatización de pruebas web multiplataforma fue implementado en función de los requerimientos especificados y agregando características básicas que cualquier framework de automatización debe tener.

Los principales elementos que se deben tener para automatizar pruebas son las paginas, los pasos y las pruebas en texto plano.

4.5.2.1 Pruebas en texto plano

Gracias a la implementación de Cucumber, el framework permite escribir las pruebas en texto plano, esto facilita de que cualquier persona que no posea conocimientos técnicos del área pueda entender lo que se está probando.

Sirve de guía para desarrolladores, gerentes y todo el equipo en general para conocer que escenarios se están cubriendo de forma automatizada y como poder ejecutarlos manualmente.

```
@Web
Scenario: Login with valid creadentials
 Given I go to http://myWebPage.com
 When I enter Lucia.Marin as user name
 And I enter Password292 as password
 And I click on login button
 Then I shoould be redirected to the home page

@Web
Scenario: Login with invalid credentials
 Given I go to http://myWebPage.com
 When I enter invalid.user as user name
 And I enter invalid.pass as password
 And I click on login button
 Then I should see an error message that says Invalid Credentials
```

Figura 4.12 Definición de pruebas en texto plano (Elaboración propia)

4.5.2.2 Definición de pasos de pruebas

La definición de pasos de pruebas son clases en las cuales se define el código a ser ejecutado para cada oración de los archivos de pruebas en texto plano.

En otras palabras, es la asociación de instrucciones presentes en los archivos de pruebas con métodos a ser ejecutados.

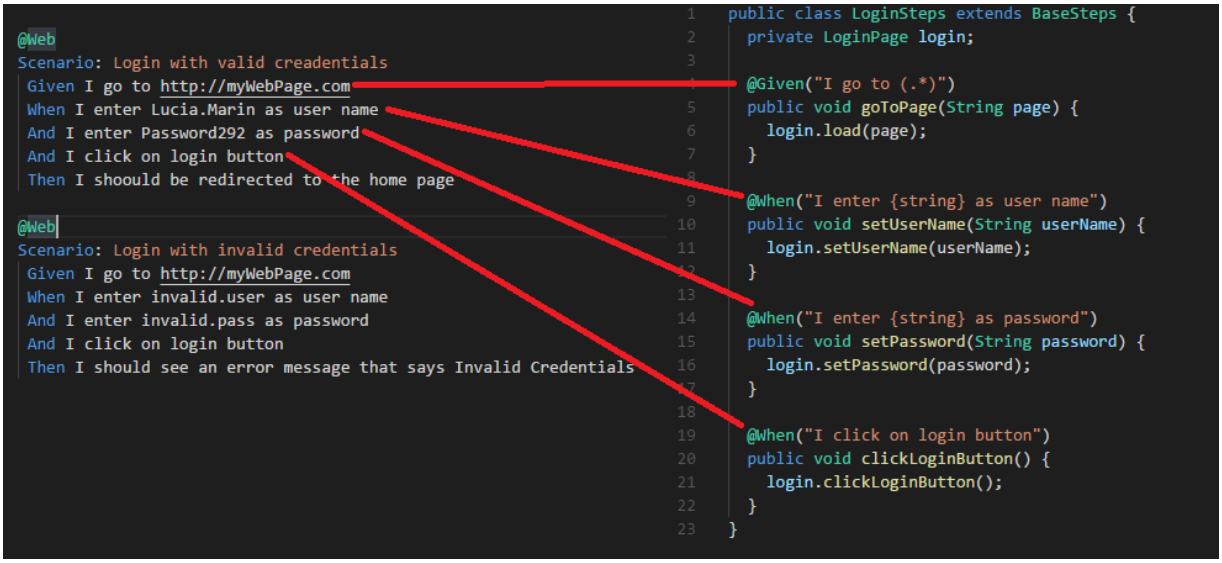

Figura 4.13 Definición de pasos de pruebas (Elaboración propia)

4.5.2.3 Definición de paginas

Se definen las representaciones de elementos de la interfaz de usuario como una página, sus componentes y acciones.

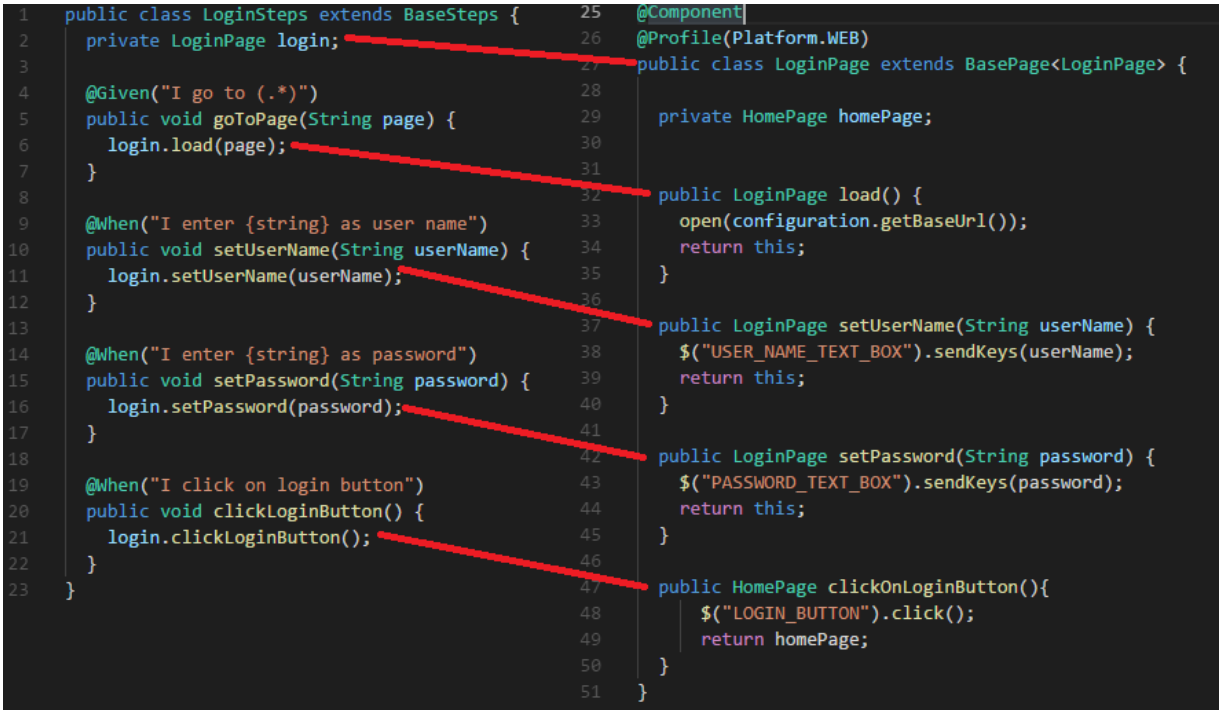

Figura 4.14 Definición de páginas (Elaboración propia)

Con el tag @Profile podemos determinar a qué tipo de plataforma pertenece la página, la misma página puede aplicarse para una o más plataformas, idealmente si solo se diferencian los localizadores en las plataformas, una sola página puede ser usada para todas las plataformas, caso contrario se puede crear paginas específicas para cada plataforma.

```
@Component
@Profile({ANDROID, WEB})
public class LoginPage extends BasePage<LoginPage> {
```

Figura 4.15 Definición plataforma en páginas (Elaboración propia)

4.5.2.4 Configuraciones de ejecución

Las configuraciones de ejecución se las especifica en el archivo “test.properties”.

```
# FRAMEWORK config
platform=web
pages.package=qa.mvr.google.pages
steps.package=qa.mvr.google.setepDefinitions
features.directory=src/test/resources/features/stackoverflow
cucumber.report.directory=target/report/cucumber

# WEB config
web.baseUrl=https://www.google.com
web.browser=chrome

# MOBILE config
mobile.appiumUrl=http://127.0.0.1:4723/wd/hub

# ANDROID config
android.appPackage=com.example.android
android.appActivity=com.example.android.MainActivity
android.appPath=/Users/mauricio/chrome.apk
android.deviceName=Google Nexus 5
```

Figura 4.16 Configuraciones de ejecución (Elaboración propia)

En este archivo se establecen configuraciones del framework como los paquetes donde están ubicadas las páginas y definición de pasos, configuraciones web donde se especifica el navegador a usar y la pagina base, configuraciones para dispositivos móviles como el Appium server, configuraciones para Android referentes a la aplicación a ejecutar y el dispositivo donde se ejecutará.

4.5.2.5 Ejecutor de pruebas

JUnit es utilizado para ejecutar las pruebas, solo es necesario extender la clase "TestRunner" para crear el ejecutor de pruebas.

```
public class Runner extendsTestRunner {}
```

Figura 4.17 Ejecutor de pruebas (Elaboración propia)

4.5.2.6 Localizadores de interfaz de usuario

Como vimos Selenium y Appium pueden localizar elementos de interfaz de usuario por id, css, xpath y otros, cada elemento debe tener un identificador único como (BOTON_LOGIN), el cual es mapeado a un localizador, estos son representados en archivos yaml, estos deben ser agrupados por páginas.

```
LOGIN_BUTTON:
  web:
    type: xpath
    value: "//div//button[text()='login']"
  android:
    type: id
    value: "btn_login"
USER_NAME_TEXT_BOX:
  web:
    type: id
    value: "user_Name_textbox"
  android:
    type: css
    value: "div>div.userName"
```

Figura 4.18 Localizadores de interfaz de usuario (Elaboración propia)

4.6 Pruebas

Se realizaron pruebas para verificar el correcto funcionamiento del framework, automatizando las 4 pruebas presentadas en el capítulo de diagnóstico, en este caso usando un solo framework de automatización, es decir la propuesta implementado.

Se observó que solo fueron necesarias alrededor de 400 líneas de código implementado, porque se pudo reducir las líneas de código aplicadas para automatizar pruebas web multiplataforma con las herramientas actuales en un 60%.

Se probaron y verificaron que los siguientes escenarios funciones correctamente:

E1: El framework ejecuta pruebas automatizadas web en navegadores en ordenadores de escritorio.

E2: El framework ejecuta pruebas automatizadas web en el navegador Chrome para Android.

E3: El framework está integrado con la herramienta con Cucumber, para representar en texto plano las pruebas de software.

E4: El framework maneja definiciones de páginas de las interfaces de usuario independientes para cada plataforma (web y móvil).

E5: El framework maneja definiciones de objetos de la interfaz de usuario de forma independiente para cada plataforma (web y móvil).

E6: El framework permite configurar el navegador a ser usado desde un archivo de propiedades para el caso de web en ordenadores.

E7: El framework permite configurar los parámetros necesarios para correr pruebas en Android (package, activity, apk, deviceName) en un archivo de propiedades.

E8: El framework permite configurar la dirección del servidor Appium desde un archivo de propiedades.

E9: El framework tiene un ejecutor de pruebas.

E10: El framework permite manejar localizadores de elementos de interfaz de usuario independientes a la plataforma.

E11: El framework permite proveer los datos de prueba desde archivos YAML.

E12: El framework genera reportes de pruebas ejecutadas.

E13: Los métodos que exponen las apis de Selenium están disponibles para ser usadas en el caso de automatización de pruebas para ordenadores de escritorio.

E14: Los métodos que exponen las apis de Appium están disponibles para ser usadas en el caso de automatización de pruebas para dispositivos móviles.

E15: El framework permite crear pruebas usando el patrón página – objeto.

E16: El framework tiene tiempos de respuesta similares a los que tiene Selenium en el caso de automatización de pruebas para ordenadores de escritorio.

E17: El framework tiene tiempos de respuesta similares a los que tiene Appium en el caso de automatización de pruebas para dispositivos móviles.

4.7 Mantenimiento

Las actividades posteriores serán enfocadas a manteniendo, debido a que las dependencias que usa el framework van actualizándose constantemente, es necesario en algunas ocasiones realizar tareas de mantenimiento, cuando haya cambios relevantes en las librerías como inclusión de nuevas características, o cambios que puedan afectar el buen desempeño del framework.

Por otro lado, constantemente tenemos a disponibilidad nuevas herramientas de automatización, y con mejores características, por lo tanto, en las tareas de mantenimiento se analizará inclusión de nuevas librerías para la mejora de la actual implementación.

4.8 Conclusiones

Se puede concluir que el framework de automatización multiplataforma implementada reduce en gran medida el trabajo de crear/mantener pruebas automatizadas para cada plataforma, teniendo toda la lógica de automatización en un solo framework.

El framework implementado posee las características que cualquier framework de automatización estándar debe poseer, creación de pruebas, logs, reportes, configuraciones de archivos.

Se pudo lograr una implementación que funcione para plataformas web y móvil, a partir de la implementación de localizadores y paginas independientes para cada plataforma que son usadas en los pasos, por lo cual los pasos ya no son re escritos para cada plataforma, como cuando se usa Appium y Selenium, en el mejor de los casos hasta los localizadores y paginas serán compartidas, facilitando mucho más la creación de pruebas.

CONCLUSIONES

En la investigación se pudo constatar que muchos de los proyectos de desarrollo de software en donde el producto es una aplicación web con soporte para ordenadores de escritorio y dispositivos móviles la automatización de pruebas de software se la realiza para solo una de las plataformas, unas pequeñas porciones tienen pruebas automatizadas para ambas plataformas.

Se pudo identificar que una de las causas era que esto implicaba mayor esfuerzo, incremento de recursos humanos, posibles impactos en tiempos de entrega, por lo cual se decidía dejar de lado la plataforma menos utilizada.

De manera práctica, se pudo evidenciar que el esfuerzo para automatizar pruebas para ambas plataformas es prácticamente el doble (doble de líneas de código) que solo cubrir una sola plataforma, esto de manera práctica aplicando herramientas de automatización para cada una de ellas.

Se analizó las herramientas más usadas (Appium y Selenium) y se identificó una dependencia entre ellas, muchas similitudes, lo que hacía posible una combinación de ambas tecnologías para generar un solo framework capaz de cubrir las pruebas en las plataformas web y móvil.

Se implementó un framework modelo de automatización de pruebas web multiplataforma, capaz de ejecutar pruebas web en dispositivos móviles y ordenadores de escritorio, con una reducción de esfuerzo del 60%, comprobadas debido a que gran parte del código es reusable, esto fue demostrado con contabilizando las líneas de código usadas para implementar un grupo de 4 pruebas, usando el framework propuesto (alrededor de 400 líneas de código) y usando independientemente Appium y Selenium (más de 1000 líneas de código.)

También se pudo demostrar la hipótesis inicial:

“La combinación de tecnologías de automatización de pruebas (móviles y de escritorio) podrían permitir el diseño e implementación de un “framework” capaz de

ejecutar las mismas pruebas en diferentes navegadores y tipos de dispositivos, ya que existen similitudes entra las herramientas usadas para ambas tecnologías.”

Esto debido a que el framework propuesto está basado en la combinación de tecnologías existentes (Appium y Selenium)

RECOMENDACIONES

- Para poner en práctica el framework en proyectos reales, re recomienda tener conocimiento en Appium y Selenium, ya que la propuesta esta basada en esas herramientas, y seguir las buenas practicas que sus desarrolladores sugieren.
- Las librerías base se actualizan constantemente, por lo cual se deberá dar mantenimiento al framework, futuras investigaciones pueden estar orientadas a la reducción de dependencias en los componentes más importantes como en los drivers.
- Futuras investigaciones pueden estar en agregar más características al framework, como adaptación para ejecuciones paralelas, ejecuciones en servicios en la nube y otros.

REFERENCIAS BIBLIOGRAFICAS

Álvarez, C., Sierra, V. (2016). Metodología de la investigación científica. Cochabamba, Bolivia.

Ammann, P., y Offutt J. (2008). Introduction to software testing. Nueva York, Estados Unidos.

Everett, G. y McLeod, R. (2007). Software Testing, testing across the entire software development cycle. New Jersey, Estados Unidos.

Galin, D., (2004). Software Quality Assurance from theory to implementation. Inglaterra.

Jacobson, I., Booch, G., y Rumbaugh, L. (2000). El Proceso Unificado de Desarrollo de Software. Madrid, España

Muñoz Razo, C. (2002). Auditoria de Sistemas Computacionales. México

Pressman, R., (2010). Ingeniería de Software un enfoque práctico. Mexico.

Toledo, F. (2014). Introducción a las pruebas de sistemas de la información. Montevideo, Uruguay.

ANEXOS

ANEXO 1. GLOSARIO

- API: Application Programming Interface
- JSON: Javascript Object Notation
- HTTP: Hipertext Transfer Protocol
- REST: Representational State Transfer
- IDE:Integrated Development Environment
- BDD:Behavior Driven Development
- URL: Uniform Resource Locator

ANEXO 2. ENCUESTA

Automatizacion de pruebas web y movil

*Obligatorio

En el proyecto actual en el cual usted trabaja, se aplica la automatizacion de pruebas? *

- ◯ Si
- ◯ No

La automatizacion de pruebas es aplicada a pruebas para web de escritorio y dispositivos moviles?

- ◯ Ambas
- ◯ Solo una plataforma

- ◯ En su proyecto, a que plataforma esta dirigida la automatizacion de pruebas?
- ◯ Web
- ◯ Movil
- ◯ Ambas
- ◯ Opción 5

Si solo se aplica automatizacion a una plataforma, cual piensa que es la razon?

- ◯ Requiere mas esfuerzo
- ◯ Una de las plataformas es menos usada
- ◯ Requiere mas tiempo
- ◯ Requiere mas recursos humanos
- ◯ Requiere mas recursos de hardware
- ◯ Requiere ingenieros especializados por area

Cual es la herramienta base de automatizacion de pruebas web en su proyecto?

- ◯ Selenium
- ◯ Protactor
- ◯ Test Complete
- ◯ Sikuli
- ◯ Watir
- ◯ Ranorex
- ◯ Otro

Cual es la herramienta base de automatizacion de pruebas para moviles en su proyecto?

- ◯ Appium
- ◯ Calabash
- ◯ Xamarin
- ◯ Espresso
- ◯ Robotium
- ◯ Otro

Cual es la herramienta BDD usada en su proyecto?

- ○ Cucumber
- ○ JBehave
- ○ JDave
- ○ Concordion
- ○ Ninguna
- ○ Otros

Cual es el lenguaje de programacion usado para la automatizacion de pruebas en su proyecto?

- ○ Java
- ○ C#
- ○ Phyton
- ○ Ruby
- ○ Javascript
- ○ Otro