2018

HERRAMIENTAS DE AUTOMATIZACIÓN PARA PRUEBAS DE SOFTWARE

FERNÁNDEZ VERA, SEBASTIÁN NICOLÁS

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

“HERRAMIENTAS DE AUTOMATIZACIÓN PARA PRUEBAS DE SOFTWARE”

SEBASTIÁN NICOLÁS FERNÁNDEZ VERA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Profesor Guía: Marcello Visconti Z.
Profesor Correferente: Cecilia Reyes C.

NOVIEMBRE - 2018

DEDICATORIA

A mi familia, mis padres, mis hermanos y mis abuelos, gracias por estar siempre a mi lado.

RESUMEN

En este trabajo se presenta el diseño, elaboración y construcción de pruebas de regresión automatizadas mediante distintas herramientas *open source* tales como Selenium y SoapUI, con el objetivo de realizar una análisis evaluativo y comparativo de las diferencias de esfuerzo entre la realización de pruebas manuales contra la realización de pruebas automatizadas.

Mediante el uso de las herramientas provistas por Selenium Webdriver, TestNG y SoapUI se construyeron pruebas automatizadas a partir de requerimientos básicos de las plataformas escogidas. A continuación se ejecutaron las pruebas construidas y se registraron los resultados tanto para pruebas manuales como automáticas, medidos en tiempo y esfuerzo. Estos resultados fueron comparados frente a frente, donde las pruebas automatizadas entregaron tiempos de ejecución hasta 20 veces más rápidos. Además, fue posible determinar el número de iteraciones en que la ejecución automática es más eficiente que la ejecución manual.

Palabras Clave— *Automatización, Regresión, Open Source, Calidad de Software*

ABSTRACT

This work shows the process of design, elaboration and construction of automated regression tests through the use of diverse open source tools like Selenium and SoapUI, with the objective of obtaining an evaluative and comparative analysis of the differences between executing manual tests versus executing automated tests.

Automated test were built based on fundamental requirements of the platforms chosen, with the help of the tools provided by Selenium Webdriver, TestNG and SoapUI. The constructed test were executed and the results for the automated and manual test were recorded and measured in time and effort. These results where compared face to face, where the automated test delivered execution times up to 20 times faster than the manual tests. It was also possible to determine the number of iterations required in which the automatic execution is more efficient than manual execution.

Keywords— *Automation, Regression, Open Source, Software Quality*

GLOSARIO

API: Una *Application Programming Interface* es un conjunto de métodos definidos que permiten la comunicación entre varios sistemas o componentes.

AVI: Administrador Virtual Inteligente, es una plataforma web que permite la administración de servicios de usuarios finales de una operadora de telecomunicaciones.

CA: Centroamérica

CL: Chile

OpenSource: Código Abierto, es un modelo de desarrollo de *software* basado en la colaboración abierta, que permite el acceso al código fuente.

REST: *Representational state transfer* es un estilo de arquitectura web basado en el estándar HTTP.

SOAP: *Simple Object Access Protocol* es un protocolo que define mediante el intercambio de datos XML la comunicación entre dos objetos.

XML: *eXtensible Markup Language*, es un metalenguaje de etiquetas que permite estructurar datos para almacenarlos de forma legible.

Driver: Programa que permite controlar un dispositivo instalado en una computadora.

ROI: *Return on Investment,* corresponde a un indicador que permite evaluar la eficiencia de una inversión.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Hoy en día es común la producción de cientos de nuevos desarrollos de software, los cuales pueden abarcar distintos mercados y diversas plataformas. El objetivo de cada uno de estos productos es satisfacer al cliente hacia el cual va destinado, y por ende, que este último lo utilice. Para ello debe cumplir con las funcionalidades propuestas y atravesar por distintas pruebas de certificación. Sin embargo es común la salida al mercado de software defectuoso debido a la deficiente o nula certificación. Las pruebas de certificación requieren de una planificación y estrategia definida, esto se traduce en esfuerzo y tiempo de desarrollo el cual generalmente es limitado; más aún cuando las pruebas son ejecutadas de forma manual, por esta razón suele ser relegada a segundo plano lo cual repercute en la correcta funcionalidad del producto desarrollado.

En este punto aparecen distintas estrategias las cuales nos permiten agilizar el proceso de certificación y ejecución de pruebas. Dentro de estas destaca la automatización de pruebas debido a su rapidez de ejecución, en contraste con la ejecución manual; pero a un costo mayor en su construcción.

En este documento se presenta una alternativa para la realización de pruebas de software mediante la automatización, en particular las pruebas de regresión pues son realizadas un gran número de veces. El proceso de automatización permite agilizar la ejecución de pruebas, previamente diseñadas a partir de un requerimiento y con sus respectivos pasos de ejecución. Para llevar a cabo este propósito se seleccionaron ciertas herramientas de software enfocadas en la construcción de pruebas automatizadas, en especial sobre desarrollos web y *webservices*. Con el diseño y posterior ejecución de las pruebas construidas se elaboró un análisis comparativo entre la ejecución de pruebas manuales versus la construcción y ejecución de pruebas automatizadas, logrando obtener mejoras considerables en los tiempos de ejecución de las mismas. Además de una solución escalable, pues fácilmente se pueden seguir añadiendo pruebas a los distintos conjuntos ya definidos.

El documento presentado ha sido estructurado en los siguientes capítulos:

Capítulo 1 Definición del Problema Presenta el problema a abordar, el contexto bajo el cual se desarrolla el trabajo y los objetivos que se buscan abarcar.

Capítulo 2 Marco Conceptual Introduce los principales conceptos a utilizar y las herramientas a emplear para lograr los objetivos previamente planteados.

Capítulo 3 Propuesta de Solución Expone las bases a utilizar, los pasos realizados para el diseño, construcción e implementación de la solución y los resultados obtenidos para cada uno de estos.

Capítulo 4 Validación de la Solución Entrega los resultados obtenidos a partir de la ejecución de la solución implementada, se evalúan, comparan y analizan los resultados de las ejecuciones.

Capítulo 5 Conclusiones Finalmente, se entregan las conclusiones derivadas a partir del trabajo realizado.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

La implementación de procesos de *QA*[1] ha sido un tópico recurrente en los diversos mercados, más aún en los tecnológicos, en donde el término *SQA*[2] ha tomado especial fuerza en los últimos años. Esto en respuesta a la necesidad de cumplir con las planificaciones y normativas que aseguren la entrega de un producto de calidad. Dentro del proceso propio de *SQA* destaca el proceso de *testing*[3] pues es una de las fases más críticas al momento de asegurar el correcto lineamiento y funcionamiento de un producto.

1.1 Contexto

Sixbell[4] es una empresa dedicada al desarrollo e integración de soluciones tecnológicas de comunicaciones. Con una trayectoria de 28 años, Sixbell ha brindado numerosas soluciones a distintos cliente, tanto del ámbito público como privado, a lo largo de Chile y América.

En la actualidad, el departamento de Innovación y Desarrollo de Sixbell Chile se encuentra impulsando la implementación de nuevos procesos de aseguramiento de calidad de software sobre sus distintas soluciones. Dentro de este contexto la empresa ha actualizado y mejorado sus políticas de desarrollo, de la mano con la incorporación de los nuevos procesos de QA.

Uno de los tópicos al cual más énfasis se ha otorgado es a la certificación funcional, considerando la envergadura de los proyectos y la siempre creciente implementación de nuevos *features*[5], lo cual repercute directamente en la mantenibilidad del software.

1.2 Descripción de la situación actual

En la actualidad existen un exorbitante número de nuevos desarrollos de *software*, los cuales abarcan diversos mercados y plataformas. Gran parte de estos nuevos desarrollos presentan numerosos defectos al momento de ser liberados debido, principalmente, a la necesidad de cumplir con plazos establecidos. Si bien en los últimos años se han incorporado mejoras en los procesos de aseguramiento de calidad y certificación, el factor tiempo sigue siendo un común denominador a la hora de apuntar a un culpable por la presencia de defectos en la liberación del *software*.

[1] *Quality Assurance,* Aseguramiento de Calidad.
[2] *Software Quality Assurance,* Aseguramiento de Calidad en Software.
[3] El proceso de *testing* corresponde a la implementación y realización de pruebas de software sobre el producto diseñado.
[4] http://www.sixbell.com/sixbell-corporativo
[5] Funcionalidades u opciones que son agregadas sobre un desarrollo base.

Es por ello que una de la soluciones para disminuir las tasas de defectos, y a la vez no impactar en demasía los plazos de desarrollo, es la incorporación de métodos de certificación automatizables.

En respuesta a esta problemática, diversas compañías han puesto a disposición de los desarrolladores herramientas que permiten agilizar los procesos de *testing*, y de esta manera alinearse a los procesos de desarrollo de software basados en metodologías agiles.

Ahora bien, este tipo de situaciones se dificulta aún más cuando es requerido liberar nuevas versiones que presentan ajustes, o se incorporan nuevas funcionalidades al producto base, los cuales son factores que dificultan la mantenibilidad del producto. Por ello uno de los focos que hoy existen en el ámbito del proceso de *testing* es la incorporación de *Suites* de pruebas[6] de regresión, con el fin de asegurar el correcto comportamiento de las funcionalidades base del producto al momento de incorporar nuevas.

1.3 Objetivos de la solución

Objetivo General

Automatizar, mediante el uso de herramientas, la ejecución de pruebas de regresión sobre nuevos desarrollos de software.

Objetivos Específicos

- Analizar y evaluar herramientas *Open Source* para crear y ejecutar portafolios de casos de pruebas automatizables y reutilizables.
- Elaborar casos de pruebas de regresión aplicables sobre desarrollos, apoyándose de las herramientas disponibles.
- Aplicar y ejecutar herramientas y casos de pruebas creados sobre proyectos en desarrollo.
- Medir y comparar los esfuerzos entre la ejecución de pruebas automatizadas y ejecución de pruebas manuales.

[6] Conjunto o colección de pruebas agrupadas para su ejecución.

CAPÍTULO 2: MARCO CONCEPTUAL

2.1 Calidad de Software

Calidad es un concepto complejo de explicar, pues tiene distintos significados según la persona y el contexto. Kitchenham y Pfleeger discuten cinco visiones de cómo se percibe la calidad (Naik & Tripathy, 2008):

1. Visión trascendental: considera la calidad como algo que es reconocible pero no definible.

2. Visión del usuario: la calidad depende de si el producto cumple las necesidades y expectativas del usuario.

3. Visión de fabricación: la calidad es determinada según el grado en que el producto cumple con las especificaciones y requerimientos.

4. Visión de producto: las características (calidad) internas del producto determinan su calidad externa.

5. Visión basada en el valor: la calidad esta determina según cuanto está dispuesto a pagar el consumidor por ella.

Dentro de estas visiones presentadas, hay dos que destacan sobre las demás al momento de aplicarlas sobre productos de *software,* estas son la visión del usuario y visión de fabricación. Ambas representan de mejor manera lo que uno espera como calidad en un producto de *software.* Es más, el IEEE[7] sugiere la siguiente definición (IEEE, 1991):

> "Calidad de Software es:
>
> 1. El grado en que un sistema, componente o proceso cumple los requerimientos especificados.
> 2. El grado en que un sistema, componente o proceso cumple con la necesidad o expectativas del consumidor o usuario."

Esta definición concuerda con las visiones previamente presentadas, y es el marco bajo el cual se rigen hoy en día numerosos desarrollos de software.

[7] Instituto de Ingeniería Eléctrica y Electrónica, https://www.ieee.org/index.html

Pero, ¿Qué características definen si un software es de calidad? Según la norma ISO/IEC 25010:2011 se definen 8 características que determinan la calidad de un producto de software (ISO, 2017):

- Funcionalidad
- Eficiencia
- Compatibilidad
- Usabilidad
- Confiabilidad
- Seguridad
- Mantenibilidad
- Portabilidad

Figura 1: Características de un Software de Calidad

Fuente: (Viraj, 2016)

Todas estas características son de vital importancia en un producto de software, pero la característica esencial es la funcionalidad, pues es el núcleo de un producto. Un producto

que no cumple con la funcionalidad esperada no sirve, independiente de la existencia de las otras características.

Cada característica de un producto posee diversas maneras de asegurar su calidad, por un parte el aspecto funcional generalmente se mide con pruebas funcionales, asegurando que estas cumplen con las especificaciones y requerimientos del cliente o usuario.

2.2 Pruebas Funcionales

Las pruebas funcionales poseen como objetivo validar que el producto de software cumple con las especificaciones requeridas. Para esto se diseñan modelos de prueba con el objetivo de evaluar cada una de las opciones que posee el producto.

La realización de una prueba consiste de al menos 3 pasos (Rätzmann & De Young, 2003):

1. Planificación: Se determina que será probado, se identifica el caso de prueba, las actividades de la prueba y que resultado se espera de esta prueba.
2. Ejecución: A su vez la ejecución también se divide en 3 pasos:
   2.1. Preparar el ambiente de prueba
   2.2. Completar la prueba
   2.3. Determinar resultados de la prueba
3. Evaluación: La fase de evaluación tiene como objetivo comparar el resultado obtenido con el resultado esperado (planificado).

Las pruebas funcionales se pueden clasificar en distintas categorías según el contexto/etapa en que se realizan, el orden lógico es el siguiente:

- Unitarias
- Componentes
- Integración
- Sistema
- Humo
- Alfa
- Beta
- Aceptación
- Regresión

2.2.1 Pruebas de Regresión

La idea principal tras las pruebas de regresión es verificar que no se presentan nuevos errores en secciones del software que no han sido modificadas luego de introducir una nueva funcionalidad, mejora o corrección en otra sección (Naik & Tripathy, 2008).
El hecho de que una nueva inclusión en el desarrollo de software de un producto pueda, potencialmente, introducir un defecto en una sección que previamente funcionaba correctamente hace tentadora la inclusión, en esta etapa, de todas las pruebas que fueron ejecutadas en etapas previas. Por lo tanto, las pruebas de regresión consisten, principalmente, de pruebas sobre funcionalidades consideradas críticas para el correcto funcionamiento de la aplicación. A medida que incrementa el volumen de funciones también lo hace el volumen de pruebas a seleccionar, por ello una de las opciones para enfrentar esta situación es la automatización, pero no se debe subestimar el esfuerzo y costo que, a la vez, posee este proceso.

2.3 Automatización

La automatización de pruebas tiene como propósito ejecutar pruebas funcionales los más rápido y eficientemente posible(Garg, 2014). Es por esto que numerosos mercados han adoptado dentro de sus líneas productivas diversos procesos de automatización, y el desarrollo de software no es ajeno a esta realidad. Ahora bien, hay que tener en cuenta que si bien la automatización de pruebas entrega variados beneficios como repetitividad, reusabilidad y rapidez, la implementación también posee sus costos. Según Kaner et al(Kaner, Bach, & Pettichord, 2002) el costo estimado en crear una prueba automatizada es equivalente a diez veces el costo de realizar la prueba manualmente.

Para realizar esta implementación diversas compañías han puesto a disposición de los desarrolladores herramientas que facilitan y agilizan la implementación de pruebas automáticas.

2.4 Herramientas y sus alcances

2.4.1 Selenium Webdriver

Selenium Webdriver es parte de un conjunto de herramientas desarrolladas en el marco del Selenium Project la cual posee como novedad la inclusión de la API Webdriver, que permite interacción directa con diversos navegadores. Esta herramienta tiene como foco la automatización de pruebas sobre aplicaciones web (Selenium Project, 2018).

Es una herramienta flexible, permitiendo desarrollar casos de prueba en diversos lenguajes de programación y ejecutarlos en diversos navegadores, además de soportar *frameworks* de prueba tales como JUnit o TestNG.

Su utilización se basa en el uso de *drivers*[8] de navegadores, los cuales permiten el acceso a funcionalidades propias del navegador en cuestión. Para esto las librerías de Selenium utilizan el método *driver,* con el cual es posible la manipulación de inicio, termino, modificación, entre otras funcionalidades, sobre las ventanas del navegador que se desee utilizar.

2.4.2 TestNG

TestNG es un *testing framework*[9] basado en JUnit[10], diseñado por Cédric Beust (Beust, TestNG, 2017) para suplir ciertas deficiencias que presenta JUnit hasta el día de hoy.

Esta versátil herramienta fue diseñada para simplificar una amplia gama de tipos de pruebas, desde pruebas unitarias hasta pruebas de integración, permitiendo incluso probar *frameworks* externos, como servidores de aplicaciones.

Dentro de la gama de funcionalidad que hereda TestNG de JUnit se agrega soporte para anotaciones, parámetros, pruebas de código sobre *multithreading*[11], entre otras funcionalidades.

Una de las funcionalidades más destacables de TestNG es el uso de anotaciones, pues con ellas es posible identificar conjuntos de métodos a ejecutar en distintas fases de la ejecución, así como entregar parámetros extras a los métodos relacionados.

2.4.3 Soap UI

SoapUI es una herramienta *OpenSource* que permite realizar pruebas funcionales sobre SOAP (*Simple Object Access Protocol)* y REST (*Representational State Transfer).* Al poseer una interfaz gráfica de fácil uso, SoapUI permite crear y ejecutar pruebas funcionales, de regresión y de carga fácilmente (SmartBear Software, 2018).

[8] Controladores de dispositivos que permiten al sistema operativo, interactuar con un dispositivo o periférico en particular.
[9] Es un entorno de trabajo que sirve como base para el desarrollo de software. Estos contienen y generan una estructura base que incluye soporte de programas, bibliotecas y un lenguaje interpretado para facilitar el desarrollo.
[10] https://junit.org/junit5/
[11] Permite ejecutar múltiples procesos o aplicaciones paralelamente, donde cada *hilo (thread)* se encarga de la ejecución de una o un subconjunto de rutinas.

Una de las ventajas de esta herramienta es la capacidad de automatizar y administrar las pruebas creadas, permitiendo agruparlas en colecciones de pruebas para su posterior ejecución en la etapa de pruebas de regresión.

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN

3.1 Plataformas

Cada proyecto presenta un gran número de plataformas que proveen distintas funcionalidades, para este trabajo en particular el enfoque se dará en las dos plataformas descritas a continuación.

3.1.1 AVI: Administrador Virtual Inteligente

El administrador virtual inteligente es una plataforma web que permite la administración y configuración de distintos servicios sobre los usuarios finales de una operadora telefónica. Se divide en distintos módulos, los cuales permiten administrar las reglas de negocio correspondientes a cada operadora.

Se presentan los módulos de tarificación, donde se administran las configuraciones necesarias para asignar las tarifas a cada uso de tráfico según las características de este. El módulo plataforma, donde se configuran los parámetros requeridos para el uso de tráfico, entre los cuales se encuentran los destinos. Rangos numéricos, tipos de llamadas, entre otros. El módulo de clientes, donde se crean y configuran las características de los usuarios de la operadora, tales como el plan tarifario, servicios y productos. El módulo usuarios, donde se administran los usuarios de la plataforma web y sus roles dentro de la plataforma. Finalmente el módulo reportes permite obtener datos acerca del uso de tráfico y transacciones según distintos criterios.

Estas plataformas están desarrollada y adaptadas para diversos clientes de Sixbell. En este trabajo se optó por las plataformas desarrolladas para un cliente de Centroamérica [CA] y otro en Chile [CL], pues ambas se encuentran en un estado óptimo de operación. De la misma manera ambas plataformas son objeto de nuevas funcionalidades y ajustes requeridos por los clientes, lo cual las hace sujeto ideal para la implementación de pruebas de regresión.

3.1.2 Webservice

Servicios que permiten el intercambio y envío de datos mediante distintos protocolos, en este caso están elaborados en XML, utilizando como protocolo SOAP[12]. Los servicios se comunican con la plataforma de negocios encargada de la orquestación, en donde los datos enviados se aprovisionan en los esquemas y tablas correspondientes en la base de datos.

[12] *Simple Object Access Protocol* define como dos objetos en diferentes procesos pueden comunicarse mediante intercambio de datos *XML.*

3.2 Definición de Requerimientos

Para la obtención de los requerimientos se observaron las plataformas utilizadas para cada una de las regiones, identificando y seleccionando aquellos requerimientos funcionales que poseen mayor criticidad, pues estas funcionalidades corresponden a los pilares fundamentales que soportan la plataforma. La gran mayoría de los requerimientos funcionales se presentan en ambas plataformas por lo cual, en los casos que aplique, se mencionaran sus diferencias en su descripción.

Ingreso de usuarios [CA, CL]

El usuario ingresa a la plataforma mediante el uso de su nombre de usuario web y contraseña, y se le dirige a la interfaz del menú principal de la plataforma correspondiente.

Crear usuario [CA, CL]

El usuario administrador crea un nuevo usuario ingresando sus datos personales (nombre, apellido, dirección, teléfono, email, localidad), sus datos web (nombre de usuario y contraseña) y su rol en la plataforma. La plataforma retornará un mensaje de éxito cuando le creación del nuevo usuario sea satisfactoria.

[CA] Se selecciona la operadora de la cual forma parte el usuario.

Modificar usuario [CL]

El usuario administrador busca un usuario ingresando su nombre de usuario web, siendo posible la modificación de los campos a excepción del nombre de usuario.

Crear rol [CA, CL]

El usuario administrador crea un nuevo rol, ingresando el nombre/descripción del nuevo rol y asignando las funciones y permisos para realizar acciones dentro de la plataforma web.

Crear periodo [CA, CL]

El usuario administrador crea un nuevo período, ingresando la descripción de este. Este período puede ser utilizado posteriormente para la creación de calendarios de períodos.

Crear calendario periodo [CA, CL]

El usuario administrador crea un calendario ingresando, en primera instancia, la descripción de este, luego los días de vigencia del calendario, el período, la hora de inicio y hora de fin.

El calendario período tiene por finalidad determinar a qué período corresponde el tráfico realizado en ciertas combinaciones horarias (día y hora).

Crear tarifa [CA, CL]

El usuario administrador crea una nueva tarifa, ingresando la descripción de esta, las fechas en que esta tarifa será vigente, el costo aire (tiempo que tarda la conexión en realizarse) y el costo según el bloque de tiempo en el cual se encuentra la llamada.

[CA] Se selecciona el tipo de moneda asociado a la tarifa.

Crear bloque de tiempo [CA, CL]

El usuario administrador determina la estructura de los bloques de tiempo ingresando una descripción, las fechas en que se mantendrá en vigencia, el tiempo de gracia (tiempo inicial de la llamada que no es tarificada), el número de bloques inicial, intermedio y final, y la duración (en segundos) de cada uno de estos bloques. Posteriormente los bloques de tiempo serán asignados al plan tarifario en conjunto con un tipo de llamada y la tarifa asociada a este.

Crear estructura tajadas [CA]

El usuario administrador, o el operador, ingresa la descripción de la estructura de tajadas y el número de bloques que entregará cada tajada. La estructura de tajadas se utiliza en la creación del Plan Tarifario, asociado junto a un tipo de llamada, una tarifa y un bloque de tiempo. Se encarga de entregar, secuencialmente y *on-demand*, bloques tarificables necesarios para el tráfico de las llamadas.

Crear plan [CA, CL]

El operador ingresa el nombre del nuevo Plan, el límite de minutos asignado al plan, el tipo de llamada que permitirá y el tramo horario en que será utilizable. Estos planes son asignables a los clientes de la operadora.

[CA] Se selecciona la operadora donde será utilizable el Plan.

Crear producto [CL]

El operador ingresa el nombre del producto, luego tiene la opción de asignar al producto un plan de minutos pre configurado o una cuenta controlada, seleccionando el tipo de bolsa y el límite de consumo en dinero. Estos productos quedan configurados a nivel de plataforma y es posible asignarlos a los clientes de la operadora.

Crear destino [CA, CL]

El administrador, o el operador, ingresa el nombre/descripción del nuevo destino y selecciona la categoría de este (internacional, nacional fijo o nacional móvil). Luego selecciona la zona de red del destino e ingresa la máscara numérica[13] del destino.

Crear tipo de llamada [CA, CL]

El administrador, o el operador, ingresa la descripción del tipo de llamada y el alcance de este (nacional o internacional). Posteriormente el tipo de llamada será asociado, en conjunto con un bloque de tiempo y tarifa, a un Plan Tarifario. Ayuda a determinar el costo del tráfico de llamadas en función de las condiciones de estas.

Crear grupo de tipo de llamada [CA, CL]

El administrador, o el operador, ingresan el nombre del grupo de tipo de llamadas y seleccionan los tipos de llamada que se desee formen parte de este grupo. El propósito de esta agrupación es facilitar la asignación de servicios a los clientes de la operadora.

Crear plan tarifario [CA, CL]

El operador ingresa la descripción/nombre del nuevo plan tarifario e ingresa la fecha en que comenzará su vigencia.

Posteriormente tiene la opción de agregar transacciones con costo, para ello presiona el botón agregar y se presentará una interfaz donde debe ingresar la fecha de inicio de vigencia de la transacción, el costo, número de intentos sin costo, número máximo de intentos fallidos, período de gracia y el tipo de transacción.

Luego se debe seleccionar los calendarios de periodos deseados. Presionando el botón agregar se desplegará una interfaz donde se debe ingresar la fecha de inicio de vigencia del calendario, el tipo de llamada y calendario período.

Finalmente se debe seleccionar el calendario de tarifas deseado. Al presionar el botón agregar se desplegará una interfaz en donde se debe ingresar el tipo de llamada, período, sentido del tráfico (*IN* si el cliente recibe la llamada, *OUT* si el cliente origina la llamada), el

[13] Corresponde al formato del número de destino, por ejemplo 99955NN o 99955%, en donde el símbolo N representa un valor reemplazable de un número con largo fijo, y el símbolo % representa valores reemplazables de un número con largo variable. (Ej. 99955NN -> 9995512, 99955% -> 9995512345)

alcance (discrimina entre qué tipo de clientes se está cursando la llamada) y la tarifa asociada.

[CA] Se debe ingresar el código del plan (utilizan un identificador único) el tipo de moneda y el tipo de cliente al cual se le podrá asignar.

[CA] En la sección para asociar calendarios de período se debe seleccionar el bloque de estructura de tiempo y el tipo de tajadas a utilizar.

[CL] En la sección para asociar calendarios de tarifas se debe seleccionar la estructura de bloque de tiempo a utilizar.

Crear cliente vía WEB [CA]

El operador selecciona el tipo de cliente a crear y la operadora que mantendrá el contrato del cliente.

Posteriormente se completa el formulario con los datos personales del cliente, datos del usuario web que tendrá acceso al portal web, los datos del contrato (tales como Plan Tarifario, día de facturación, entre otros), los controles que se aplicarán sobre el cliente y los servicios que poseerá.

Crear línea vía WEB [CA]

Luego de seleccionar un cliente, el operador podrá asignarle una nueva línea. Para ello se debe ingresar el número de la línea a asignar, el modelo del teléfono (Móvil o Fijo) y su tipo (IP, GSM[14], etc.), en caso que corresponde a un teléfono móvil se debe ingresar su IMSI[15] y ESN[16]. Luego se debe seleccionar el perfil de servicios a utilizar en la línea, la localidad del cliente de la línea y si se agregará una cuenta anexa. En caso de corresponder a un teléfono móvil se pueden seleccionar servicios adicionales tales como destinos frecuentes.

Ver información cliente vía WEB [CA, CL]

El operador puede verificar la información personal de cada uno de los clientes bajo contrato.

El cliente puede verificar su propia información personal.

[14] *Global System for Mobile communication* corresponde a un un sistema de telefonía móvil digital

[15] *International Mobile Subscriber Identity* es un código de identificación único para cada dispositivo de telefonía móvil, el cual está integrado en la tarjeta SIM y permite la identificación en distintas redes.

[16] *Equipment Serial Number* corresponde al identificador único del equipo móvil el cual se encuentra integrado electrónicamente.

Dentro de esta interfaz es posible verificar datos personales, contractuales, líneas asignadas y servicios habilitados.

Crear cliente vía Webservice [CA, CL]

Similar a **crear cliente vía Web** el operador envía los datos del cliente requeridos para su creación los cuales son procesados mediante el webservice, generando las consultas e insertando los registros correspondientes en la Base de Datos.

[CL] Solo es posible la creación de cliente mediante webservice.

Modificar cliente vía Webservice [CA, CL]

El operador envía los datos que desea modificar del cliente, referenciándolo mediante el identificador tributario. Estos datos son procesados por el webservice, y si son válidos los registros son modificados.

Crear línea vía Webservice [CA, CL]

Similar a **crear línea vía Web** el operador envía los datos de la línea requeridos para su creación los cuales son procesados mediante el webservice, generando las consultas e insertando los registros correspondientes en la Base de Datos.

Activar servicio vía Webservice [CL]

El operador envía los datos de los servicios que desee habilitar sobre un cliente en particular, referenciándolo mediante el identificador tributario. El webservice procesa estos datos y si son válidos, modifica e ingresa los registros necesarios.

Modificar línea vía Webservice [CA, CL]

El operador envía los datos que desea modificar de la línea, referenciándola mediante el identificador tributario del cliente y el número de la línea. Estos datos son procesados por el webservice, y si son válidos los registros son modificados.

Eliminar línea vía Webservice [CA, CL]

El operador envía los datos requeridos para la eliminación física de la línea, estos datos corresponden al identificador del cliente que posee la línea, el número de la línea y la operadora que posee el contrato del cliente. El webservice procesa estos datos, y si son válidos elimina el registro.

Eliminar cliente vía Webservice [CA, CL]

El operador envía los datos requeridos para la eliminación lógica del cliente, estos datos corresponden al identificador del cliente y la operadora que posee el contrato del cliente. El webservice procesa estos datos, y si son válidos elimina lógicamente el registro.

3.3 Diseño de pruebas

Para abarcar la fase de planificación, se diseñará una prueba funcional, a partir de los requerimientos previamente presentados, que abarque el curso normal de eventos (caso exitoso) por cada requerimiento. Para la construcción de las pruebas se utilizara la siguiente estructura:

Tabla 1: Estructura de pruebas funcionales

Proyecto	Requerimiento	Plataforma	Herramienta de Automatización	Instrucciones	Resultado Esperado	Resultado Obtenido	Estado	Condición

- **Proyecto:** Sobre qué proyecto se está diseñando las pruebas funcionales.
- **Requerimiento:** Corresponden a los requerimientos presentados en el subcapítulo Definición de Requerimientos.
- **Plataforma:** Sobre la cual se desarrollan los requerimientos, ya sea página web o *webservice.*
- **Herramienta de Automatización:** La herramienta a utilizar al momento de ejecutar la prueba funcional.
- **Instrucciones:** Los *inputs*[17] necesarios para llevar a cabo la prueba.
- **Resultado Esperado:** Lo que se espera obtener como resultado de la ejecución de la prueba.
- **Resultado Obtenido:** Resultado obtenido luego de la ejecución de la prueba.
- **Estado:** Estado de ejecución de la prueba.
- **Condición:** Resolución de la prueba luego de ser ejecutada. Depende de si coincide o no el resultado esperado con el obtenido.

[17] Información enviada hacia un sistema para su procesamiento.

A continuación se presenta un ejemplo de diseño de una prueba funcional, desestimando los campos **Resultado Obtenido, Estado y Condición** pues estos solo serán añadidos luego de la ejecución de la prueba.

Tabla 2: Ejemplo de prueba Funcional diseñada

Proyecto	Requerimiento	Plataforma	Herramienta de Automatización	Instrucciones	Resultado Esperado
CHILE	Ingreso de Usuario	AVI	Selenium WebDriver	1. Se ingresa el nombre de usuario en el campo USUARIO 2. Se ingresa la contraseña del usuario en el campo CONTRASEÑA 3. Se presiona el botón ENTRAR	El Sistema autentifica al usuario. El sistema permite el ingreso al menú principal

La base del diseño de las pruebas funcionales está en una primera ejecución de forma manual, de tal manera de registrar los pasos necesarios que han de ser automatizados para la correcta ejecución de la prueba. Para ello el campo **Instrucciones** dicta los pasos que han de ser incluidos en el código de la automatización.

3.4 Implementación de la automatización

La implementación de la automatización consta de dos frentes. Por una parte las pruebas que han de ser ejecutadas sobre la plataforma web **AVI**, serán construidas en el lenguaje de programación Java[18], valiéndose de Eclipse como IDE[19], y utilizando las librerías requeridas por Selenium Webdriver. Y por otra parte las pruebas a ser ejecutadas sobre la plataforma de webservice, estas pruebas serán construidas y almacenadas sobre la herramienta SoapUI. Para comprobar los resultados obtenidos por las pruebas será utilizada la funcionalidad *Assertion*[20] presente en ambas herramientas.

[18] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)
[19] *Interactive Development Environment,* Aplicación que consolida números herramientas (editores, *debugging*, etc) consolidadas en un entorno para facilitar el desarrollo de software.
[20] Afirmación, en el caso de ambas herramientas permite comprobar y verificar campos o resultados esperados por las pruebas.

3.4.1 Automatización con Selenium Webdriver

La construcción de las pruebas automatizadas fue diseñada separando cada prueba según el módulo al cual corresponde dentro de la plataforma **AVI**, de esta manera se pueden generar conjuntos de prueba que apuntan a cada módulo en particular.

Estas pruebas fueron diseñadas en base al patrón *PageObjects*[21], de tal manera de modularizar y abstraerse del comportamiento de la página. Un ejemplo del empleo de este método se puede ver en la *Figura 2* y *Figura 3*.

```
public static void personalData(WebDriver driver, String clientName,String tributaryId,String codCount) {
    driver.findElement(By.name("clientName")).sendKeys(clientName);
    driver.findElement(By.name("clientTributaryId")).sendKeys(tributaryId);
    driver.findElement(By.name("codCount")).sendKeys(codCount);
}
```

Figura 2: Diseño de método basado en patrón PageObjects

```
Client.personalData(driver, "SeleniumTesting", "919897969594", "919897969594");
```

Figura 3: Invocación de método basado en patrón PageObjects

A continuación se presenta cada *PageObject* diseñado por módulo, los cuales son posteriormente utilizados por los métodos que responden a cada requerimiento. Por defecto cada método tiene como parámetro de entrada el campo *driver*, el cual apunta al navegador bajo el cual se ejecutarán las pruebas.

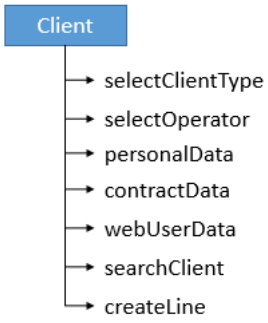


Figura 4: PageObjects módulo Client

[21] https://github.com/SeleniumHQ/selenium/wiki/PageObjects

Client

selectClientType

Descripción: Selecciona el tipo de cliente disponible en una *dropdown list.*

Parámetros: Sin parámetros de entrada.

selectOperator

Descripción: Selecciona la operadora disponible en una *dropdown list.*

Parámetros: Sin parámetros de entrada.

personalData

Descripción: Completa el formulario con los datos básicos del cliente.

Parámetros: *clientName, tributaryId, codCount.* Se entrega el nombre del cliente, su número de identificación personal y un código de identificación de la plataforma.

contractData

Descripción: Completa el formulario con los datos del plan a contratar.

Parámetros: *billingCycle, ratePlan.* Requiere del día de facturación y el identificador numérico del Plan a contratar por el cliente.

webUserData

Descripción: Completa el formulario con los datos de usuario web a asignar al cliente.

Parámetros: *userName, firstName, lastName, phone.* Se requiere el nombre de usuario a asignar al cliente, su nombre y apellido, y un teléfono de contacto.

searchClient

Descripción: Busca el cliente entregado según el parámetro indicado.

Parámetros: *searchName, platform.* Se requiere el nombre de usuario el cual buscar y la plataforma, que responde al proyecto (CL o CA) en cual buscar.

createLine

Descripción: Crea una línea para el cliente actualmente ingresado a la plataforma.

Parámetros: *phoneNumber, model, phoneType, imsi, esn, coverZone.* Requiere el número de teléfono a asociar, el modelo; ya sea fijo o móvil; el tipo de modelo, el imsi y esn si es móvil, y la zona de cobertura.

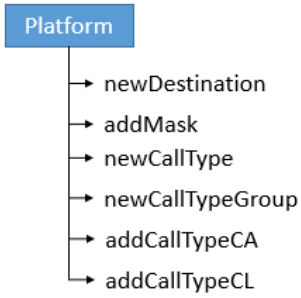


Figura 5: PageObjects módulo Platform

Platform

newDestination

Descripción: Crea un nuevo destino en la plataforma.

Parámetros: *destinationName, destinationCategory, platform.* Se requiere un nombre de destino y el tipo de destino al cual corresponde. Además de seleccionar en que proyecto se ingresará el nuevo destino.

addMask

Descripción: Asocia un valor numérico al destino creado, de tal manera de que sea visible en la red.

Parámetros: *destinationMask, networkZone, platform.* Se asigna una máscara numérica al destino, y la zona de red a la cual pertenece. Nuevamente se selecciona en que proyecto se ingresará el nuevo destino.

newCallType

Descripción: Crea un nuevo tipo de llamada.

Parámetros: *callTypeName, platform.* Se asigna el nombre a asociar al nuevo tipo de llamada y el proyecto en el cual se creará el nuevo tipo de llamada.

newCallTypeGroup

Descripción: Crea un nuevo grupo de tipo de llamada al cual asociar tipos de llamada.

Parámetros: *callTypeGroupName, callTypeGroupLevel, platform.* Se requiere un nombre para el grupo y si corresponde a un grupo padre (al cual asociar grupos hijos) o un grupo hijo. Se selecciona el proyecto sobre el cual crear el nuevo grupo.

addCallTypeCA

Descripción: Asigna tipos de llamada disponibles a un grupo tipo de llamada para el proyecto CA.

Parámetros: Sin parámetros de entrada.

addCallTypeCL

Descripción: Asigna tipos de llamada disponibles a un grupo tipo de llamada para el proyecto CL.

Parámetros: Sin parámetros de entrada.

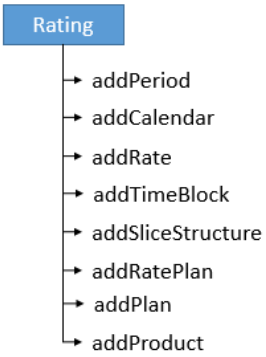


Figura 6: PageObjects módulo Rating

Rating

addPeriod

Descripción: Crea un nuevo período asignable posteriormente a un calendario.

Parámetros: *periodName, platform.* Se requiere un nombre a asignar y el proyecto en el cual se creará el nuevo periodo.

addCalendar

Descripción: Crea un nuevo calendario, designando días y horas disponibles. Este puede ser utilizado por una línea para definir horarios hábiles.

Parámetros: *calendarName, period, startTime, endTime, platform.* Requiere un nombre, un período y horas de inicio y termino para el conjunto de días.

addRate

Descripción: Crea una nueva tarifa asignable posteriormente a un plan tarifario.

Parámetros: *rateName, currency, startDate, fixedAirCost, initialBlockAirCost, firstBlockAirCost, secondBlockAirCost, fixedAdditionalCost, initialBlockAdditionalCost, firstBlockAdditionalCost, secondBlockAdditionalCost, platform.* Se requiere un nombre de tarifa, la moneda sobre la cual cobrar, fecha de inicio de la tarifa, y los costos asociados.

addTimeBlock

Descripción: Crea un nuevo bloque de tiempo, asignable en un plan tarifario.

Parámetros: *timeBlockDesc, startDate, freeTime, initialBlocks, intermediateBlocks, finalBlocks, intermediateBlockQty, finalBlockQty, platform.* Requiere un descripción, fecha de inicio, tiempo de gracia de la llamada, cantidades de bloques tarificables y los sub-bloques de cada uno.

addSliceStructure

Descripción: Crea una estructura de tajadas para las llamadas, este se asigna en el plan tarifario.

Parámetros: *sliceStructureDesc, blocks.* Se requiere una descripción y el número de bloques a entregar por tajada.

addRatePlan

Descripción: Crea un nuevo plan tarifario.

Parámetros: *ratePlanName, ratePlanOperator, startDate, currency, clientType, platform.* Requiere un nombre y el tipo de cliente asociable al plan. Si corresponde al proyecto CA se requiere también la operadora sobre la cual aplicara el plan, la fecha de inicio y la moneda sobre la cual tarificar

addPlan

Descripción: Crea un nuevo plan de minutos, asignable a un cliente.

Parámetros: *planName, planLimit, callType, periodDesc, platform.* Es necesario asignar un nombre, el límite de minutos, el tipo de llamada que cubrirá y el período en caso de corresponder al proyecto CA.

addProduct

Descripción: Crea un nuevo producto con planes de minutos por defecto.

Parámetros: *productName.* Se requiere un nombre descriptivo del producto.

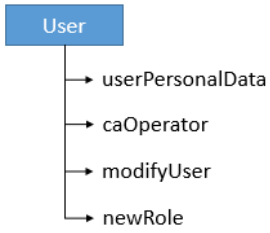


Figura 7: PageObjects módulo User

User

userPersonalData

Descripción: Completa el formulario para la creación de un nuevo usuario web.

Parámetros: *loginName, userPass, userName, userLastName, userAddress, userPhone, userEmail, homeZone.* Se necesita el nombre y contraseña de ingreso, nombre y apellido del usuario, dirección, teléfono, correo y ciudad.

caOperator

Descripción: Selecciona el operador por defecto en el formulario de creación de usuario para el proyecto CA.

Parámetros: Sin parámetros de entrada.

modifyUser

Descripción: Modifica el usuario según los parámetros ingresados.

Parámetros: *userLogin, newUserName, newUserLastName.* Busca el usuario según su nombre de ingreso y modifica el nombre de usuario y apellido si es requerido.

newRole

Descripción: Crea un nuevo rol asignable a usuarios, con permisos básicos por defecto.

Parámetros: *rolName, platform.* Requiere de una descripción del nuevo rol.

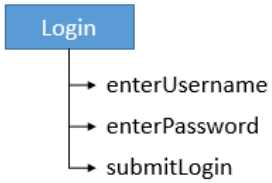


Figura 8: PageObjects sección Login

Login

enterUsername

Descripción: Ingresa el nombre de usuario administrador en el bloque ingreso.

Parámetros: Sin parámetros de entrada. Valores configurados como constantes.

enterPassword

Descripción: Ingresa la contraseña del usuario administrador en el bloque de ingreso.

Parámetros: Sin parámetros de entrada. Valores configurados como constantes.

submitLogin

Descripción: Envía las credenciales de acceso, buscando presionando el botón de ingreso.

Parámetros: Sin parámetros de entrada.

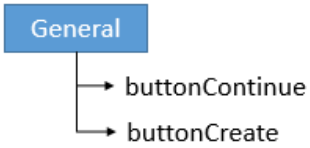


Figura 9: PageObjects de uso General

General

buttonContinue

Descripción: Busca y oprime el botón continuar cuando sea requerido.

Parámetros: Sin parámetros de entrada.

buttonCreate

Descripción: Busca y oprime el botón crear cuando sea requerido.

Parámetros: Sin parámetros de entrada.

Los pageObjects diseñados fueron utilizados según la estructura presentada en la figura 10.

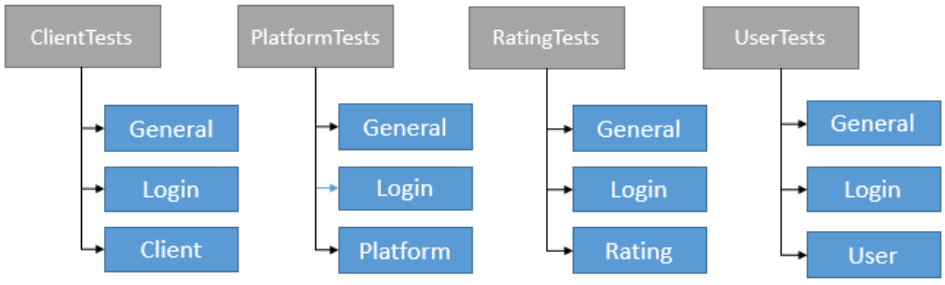


Figura 10: Estructura de los módulos a evaluar

Aprovechando las funcionalidades de *TestNG*, se utilizaron anotaciones para separar las secciones bajo las cuales se ejecutarían y evaluarían cada una de las pruebas.

Las anotaciones utilizadas son:

@BeforeClass: Define las operaciones a ser realizadas previa a la ejecución de las pruebas.

@Test(priority=x): Define las pruebas a ejecutar, la prioridad define el orden en que serán ejecutadas.

@AfterClass: Define las operaciones a ser realizadas posterior a la ejecución de las pruebas.

De esta manera cada una de los módulos a ser evaluados quedo estructurado como se presenta en la *Figura 11*.

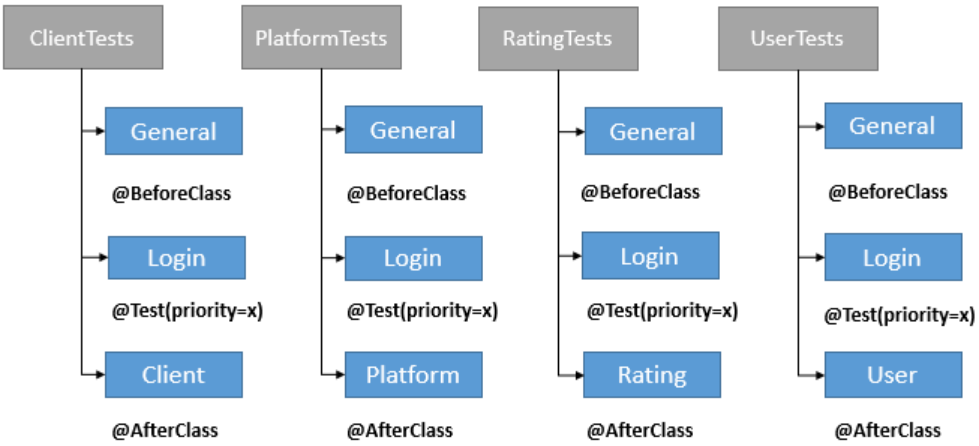


Figura 11: Uso de anotaciones sobre la estructura de módulos a evaluar

3.4.2 Automatización con SoapUI

Las pruebas sobre los webservice fueron diseñadas sobre SoapUI, utilizando como base la funcionalidad de creación de *TestSuites.*

A continuación se presentan de manera resumida los webservice utilizados para cada proyecto.

Proyecto CL

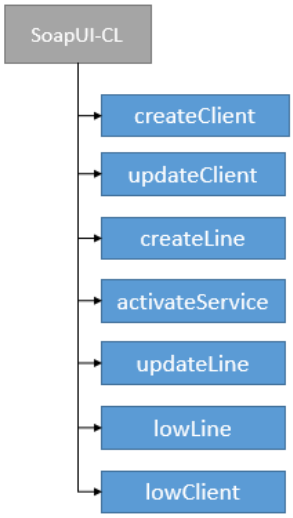


Figura 12: WebServices utilizados para proyecto Chile

createClient: Crea un nuevo cliente y lo registra en él sistema.

updateClient: Modifica y actualiza la información del cliente seleccionado.

createLine: Crea una nueva línea, asignándolo al cliente seleccionado.

activateService: Activa un servicio deseado sobre la línea seleccionada.

updateLine: Modifica y actualiza información de la línea seleccionada.

lowLine: Da de baja una línea, eliminando su registro del sistema.

lowClient: Da de baja un cliente, eliminando su registro del sistema.

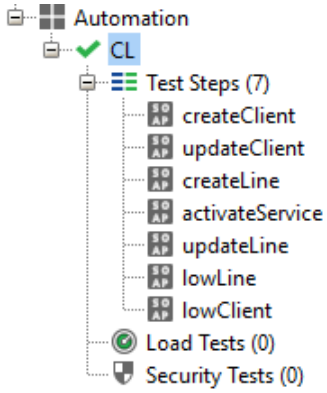


Figura 13: TestSuite construido para el proyecto Chile

Proyecto CA

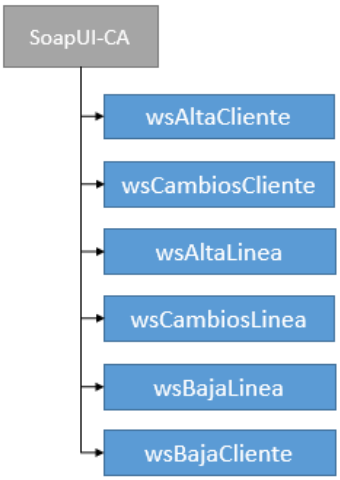


Figura 14: WebServices utilizados para proyecto Centroamérica

wsAltaCliente: Crea un nuevo cliente y lo registra en él sistema.

wsCambiosCliente: Modifica y actualiza la información del cliente seleccionado.

wsAltaLinea: Crea una nueva línea, asignándolo al cliente seleccionado.

wsCambiosLinea: Modifica y actualiza información de la línea seleccionada.

wsBajaLinea: Da de baja una línea, eliminando su registro del sistema.

wsBajaCliente: Da de baja un cliente, eliminando lógicamente su registro del sistema.

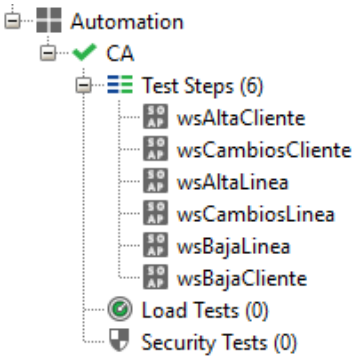


Figura 15: TestSuite construido para el proyecto Centroamérica

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

4.1 Métricas de Evaluación

Para evaluar los esfuerzos requeridos para cada una de las fases, ya sea de diseño de pruebas, diseño de automatización o ejecución, se decidió utilizar como métrica las Horas Hombre (HH).

4.1.1 Esfuerzos de elaboración de pruebas

La construcción de pruebas fue medida por proyecto, tal como se presenta en la siguiente tabla:

Tabla 3: Esfuerzos dedicados a la elaboración de pruebas funcionales

PROYECTO	ÍTEM	ESFUERZO [HH]
CA	Diseño y elaboración de Pruebas Funcionales	7
CL	Diseño y elaboración de Pruebas Funcionales	4

Para el proyecto en Centroamérica se requirieron 7 HH para la elaboración de 22 pruebas funcionales. Misma cantidad de pruebas fueron elaboradas para el proyecto de Chile en el transcurso de 4 HH, en donde se reutilizaron y modificaron algunas de las pruebas elaboradas para Centroamérica.

El esfuerzo dedicado a la elaboración de las pruebas será la base sobre la que se compararán los esfuerzos de generar y ejecutar pruebas de regresión de forma manual frente a diseñar pruebas automáticas y ejecutarlas.

4.1.2 Esfuerzos de diseño de pruebas automatizadas

En la siguiente tabla se presentan los esfuerzos requeridos para plasmar las pruebas funcionales en pruebas automatizadas sobre las herramientas de automatización utilizadas.

Tabla 4: Esfuerzos dedicados al diseño y construcción de pruebas automatizadas

PROYECTO	**ÍTEM**	**ESFUERZO [HH]**
CA	Diseño y construcción de pruebas automatizadas en Selenium Webdriver	20
	Diseño y construcción de pruebas automatizadas en SoapUI	2,5
CL	Diseño y construcción de pruebas automatizadas en Selenium Webdriver	8,5
	Diseño y construcción de pruebas automatizadas en SoapUI	2

Tal como se presenta en tabla 4, el esfuerzo dedicado al proyecto de Centroamérica corresponde a 22,5 HH en total, tiempo bajo el cual se diseñaron los *PageObjects* que posteriormente serían utilizados para el proyecto de Chile, lo cual significo una disminución considerable de esfuerzo sobre este proyecto el cual suma 10,5 HH de dedicación.

Estos esfuerzos serán posteriormente evaluados sobre el esfuerzo requerido para la realización de pruebas automatizables, pues este punto no es requerido para ejecutar de forma manual las pruebas elaboradas.

4.1.3 Esfuerzos de ejecución de pruebas manuales

Para la ejecución de las pruebas manuales se realizaron tres iteraciones, de tal manera de obtener un promedio y disminuir el error humano propio de este tipo de ejecuciones.

La tabla y figuras siguientes presentan el esfuerzo requerido por ejecución y el promedio obtenido.

Tabla 5: Esfuerzos dedicados a la ejecución manual de las pruebas funcionales

PROYECTO	ÍTEM	ESFUERZO [mm:ss]	ESFUERZO [HH]
CA	Ejecución Manual: Iteración N°1	14:19	0,239
	Ejecución Manual: Iteración N°2	14:18	0,238
	Ejecución Manual: Iteración N°3	11:39	0,194
	Ejecución Manual: Promedio	**13:25**	**0,224**
CL	Ejecución Manual: Iteración N°1	11:48	0,197
	Ejecución Manual: Iteración N°2	10:07	0,169
	Ejecución Manual: Iteración N°3	09:43	0,162
	Ejecución Manual: Promedio	**10:33**	**0,176**

4.1.4 Esfuerzos de ejecución de pruebas automatizadas

Para la ejecución de las pruebas automatizadas se realizaron cinco iteraciones, con el fin de minimizar las diferencias de tiempo por razones de retrasos en la conexión de red o *hardware* del equipo utilizado.

Los tiempos plasmados en las tablas que se presentarán a continuación reflejan los tiempos requeridos para la ejecución de las pruebas sobre ambas herramientas, es decir, las pruebas web sobre Selenium Webdriver y las pruebas de Webservice sobre SoapUI.

En el caso del proyecto Chile, las pruebas fueron realizadas en dos navegadores distintos, con el fin de comparar el comportamiento de distintos *drivers* en la herramienta Selenium webdriver.

Las pruebas fueron realizadas en el siguiente equipo:

Modelo : HP 240 G5

CPU : Intel® Core™ i5-6200U CPU @ 2.30GHz 2.40Ghz

Memoria RAM : 8,00 GB

Sistema Operativo : Windows 10 Pro

La siguiente tabla presenta los esfuerzos requeridos para el proyecto Chile, separado por los navegadores utilizados y considerando el mismo tiempo para las pruebas de Webservice.

Tabla 6: Esfuerzos dedicados a la ejecución automatizada de las pruebas sobre el proyecto Chile

NAVEGADOR	ÍTEM	ESFUERZO [mm:ss]	ESFUERZO [HH]
Mozilla Firefox	Ejecución Automatizada: Iteración N°1	00:24	0,007
	Ejecución Automatizada: Iteración N°2	00:23	0,006
	Ejecución Automatizada: Iteración N°3	00:26	0,007
	Ejecución Automatizada: Iteración N°4	00:23	0,006
	Ejecución Automatizada: Iteración N°5	00:23	0,006
	Ejecución Automatizada: Promedio	**00:24**	**0,007**
Google Chrome	Ejecución Automatizada: Iteración N°1	00:22	0,006
	Ejecución Automatizada: Iteración N°2	00:22	0,006
	Ejecución Automatizada: Iteración N°3	00:21	0,006
	Ejecución Automatizada: Iteración N°4	00:22	0,006
	Ejecución Automatizada: Iteración N°5	00:23	0,006
	Ejecución Automatizada: Promedio	**00:22**	**0,006**

Tabla 7: Esfuerzos dedicados a la ejecución automatizada de las pruebas sobre el proyecto Centroamérica

PROYECTO	ÍTEM	ESFUERZO [mm:ss]	ESFUERZO [HH]
CA	Ejecución Automatizada: Iteración N°1	01:53	0,031
	Ejecución Automatizada: Iteración N°2	01:48	0,030
	Ejecución Automatizada: Iteración N°3	01:50	0,031
	Ejecución Automatizada: Iteración N°4	01:49	0,030
	Ejecución Automatizada: Iteración N°5	01:52	0,031
	Ejecución Automatizada: Promedio	**01:50**	**0,031**

4.2 Ejecución de pruebas

4.2.1 Pruebas Manuales

Las pruebas manuales fueron ejecutadas directamente sobre las páginas web correspondientes a cada proyecto. Para ambos casos se respetaron los pasos de la ejecución descritos en cada una de las pruebas elaboradas. En la sección *3.3 Diseño de pruebas* se hacen mención a los campos que presenta cada una de estas pruebas. Para la realización de la ejecución de pruebas se completaron los siguientes campos:

Tabla 8: Campos a evaluar luego de la ejecución de cada prueba.

Resultado Obtenido	Estado	Condición

Resultado Obtenido: Dependiendo del campo **Resultado Esperado** este campo presentara como valor "**OK"** o "**NOK"**.

Estado: Si la prueba ha sido realizada, independiente de su resultado, el valor será "**ejecutada"**, en caso contrario será **"no ejecutada"**. En caso que la prueba no haya podido ser realizada por falta de ambiente o por la no existencia de la sección el valor será **"pendiente"**.

Condición: Se verifica el campo **Resultado Obtenido**. Si es OK, entonces el valor de este campo será **"aprobado"**, en caso contrario será **"rechazado"**.

A continuación se presenta el resultado de la ejecución de las pruebas, dividida según región y requerimiento:

Tabla 9: Resumen ejecución de pruebas Centroamérica (1)

Proyecto	Requerimiento	Resultado Obtenido	Estado	Condición
Centroamérica	Ingreso de Usuario	OK	Ejecutada	Aprobada
	Crear Usuario	OK	Ejecutada	Aprobada
	Crear Rol	OK	Ejecutada	Aprobada
	Crear Período	OK	Ejecutada	Aprobada
	Crear Calendario Periodo	OK	Ejecutada	Aprobada
	Crear Tarifa	OK	Ejecutada	Aprobada
	Crear Bloque de Tiempo	OK	Ejecutada	Aprobada
	Crear Estructura de Tajadas	OK	Ejecutada	Aprobada
	Crear Plan Tarifario	OK	Ejecutada	Aprobada
	Crear Plan	OK	Ejecutada	Aprobada
	Crear Destino	OK	Ejecutada	Aprobada
	Crear Tipo de Llamada	OK	Ejecutada	Aprobada
	Crear Grupo Tipo de Llamada	NOK	Ejecutada	Rechazada
	Crear Cliente (WS)	OK	Ejecutada	Aprobada

Tabla 10: Resumen ejecución de pruebas Centroamérica (2)

Proyecto	Requerimiento	Resultado Obtenido	Estado	Condición
Centroamérica	Crear Cliente	OK	Ejecutada	Aprobada
	Ver Información Cliente	OK	Ejecutada	Aprobada
	Crear Línea	OK	Ejecutada	Aprobada
	Modificar Cliente (WS)	OK	Ejecutada	Aprobada
	Crear Línea (WS)	OK	Ejecutada	Aprobada
	Modificar Línea (WS)	OK	Ejecutada	Aprobada
	Eliminar Línea (WS)	NOK	Ejecutada	Rechazada
	Eliminar Cliente (WS)	NOK	ejecutada	Rechazada

Tabla 11: Resumen ejecución de pruebas Chile (1)

Proyecto	Requerimiento	Resultado Obtenido	Estado	Condición
Chile	Ingreso de Usuario	OK	Ejecutada	Aprobada
	Crear Usuario	OK	Ejecutada	Aprobada
	Modificar Usuario	OK	Ejecutada	Aprobada
	Crear Rol	OK	Ejecutada	Aprobada
	Crear Período	OK	Ejecutada	Aprobada
	Crear Calendario Periodo	OK	Ejecutada	Aprobada
	Crear Tarifa	OK	Ejecutada	Aprobada

Tabla 12: Resumen ejecución de pruebas Chile (2)

Proyecto	Requerimiento	Resultado Obtenido	Estado	Condición
Chile	Crear Bloque de Tiempo	NOK	Ejecutada	Rechazada
	Crear Plan Tarifario	OK	Ejecutada	Aprobada
	Crear Plan	OK	Ejecutada	Aprobada
	Crear Producto	OK	Ejecutada	Aprobada
	Crear Destino	OK	Ejecutada	Aprobada
	Crear Tipo de Llamada	OK	Ejecutada	Aprobada
	Crear Grupo de Tipo de Llamada	OK	Ejecutada	Aprobada
	Ver Información Cliente	OK	Ejecutada	Aprobada
	Crear Cliente (WS)	OK	Ejecutada	Aprobada
	Modificar Cliente (WS)	OK	Ejecutada	Aprobada
	Crear Línea (WS)	OK	Ejecutada	Aprobada
	Activar Servicio (WS)	OK	Ejecutada	Aprobada
	Modificar Línea (WS)	OK	Ejecutada	Aprobada
	Eliminar Línea (WS)	OK	Ejecutada	Aprobada
	Eliminar Cliente (WS)	OK	Ejecutada	Aprobada

Tabla 13: Resumen resultados de pruebas

Proyecto	Nro. De Pruebas	Aprobadas	Rechazadas
Centroamérica	22	19	3
Chile	22	21	1

4.2.2 Pruebas Automatizadas

Las pruebas automatizadas fueron ejecutadas haciendo uso de las herramientas seleccionadas: Selenium Webdriver para las pruebas web y SoapUI para las pruebas de webservices.

Para las pruebas realizadas mediante Selenium Webdriver, el navegador por defecto utilizado fue *Google Chrome*, haciendo uso del *driver chromedriver* en su versión 2.33.506120.

Para las pruebas automatizadas realizadas en el navegador *Mozilla Firefox* fue utilizado el *driver geckodriver* en su versión 0.19.1.

A continuación, y en primer lugar se presentan los resultados obtenidos para la región Centroamérica. En las siguientes figuras se muestran las pruebas que fueron correctamente ejecutadas, junto a los resultados obtenidos.

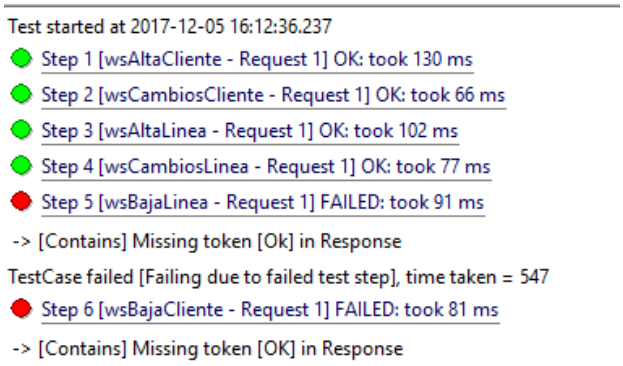


Figura 16: SoapUI - Resultado pruebas automatizadas Centroamérica

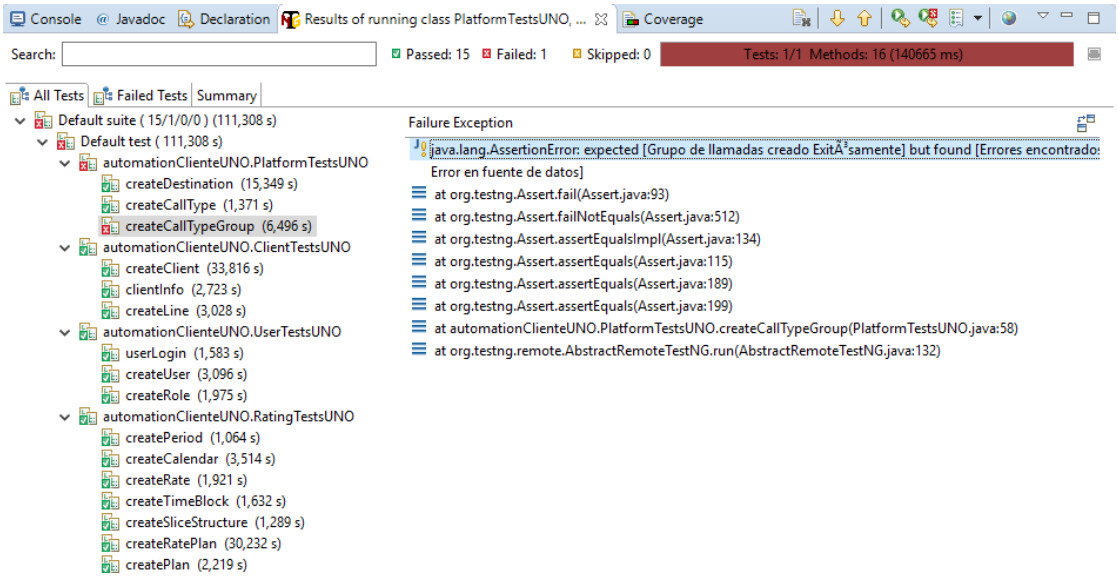


Figura 17: Selenium Webdriver - Resultado pruebas automatizadas Centroamérica

A continuación, se presentan los resultados obtenidos para la región Chile. En esta ocasión se añaden los resultados de la ejecución de las pruebas web realizadas con dos *drivers* de navegadores distintos. Los navegadores utilizados corresponden a *Google Chrome* y *Mozilla Firefox.*

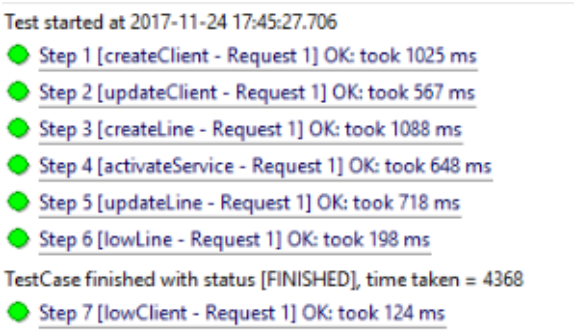


Figura 18: SoapUI - Resultado pruebas automatizadas Chile

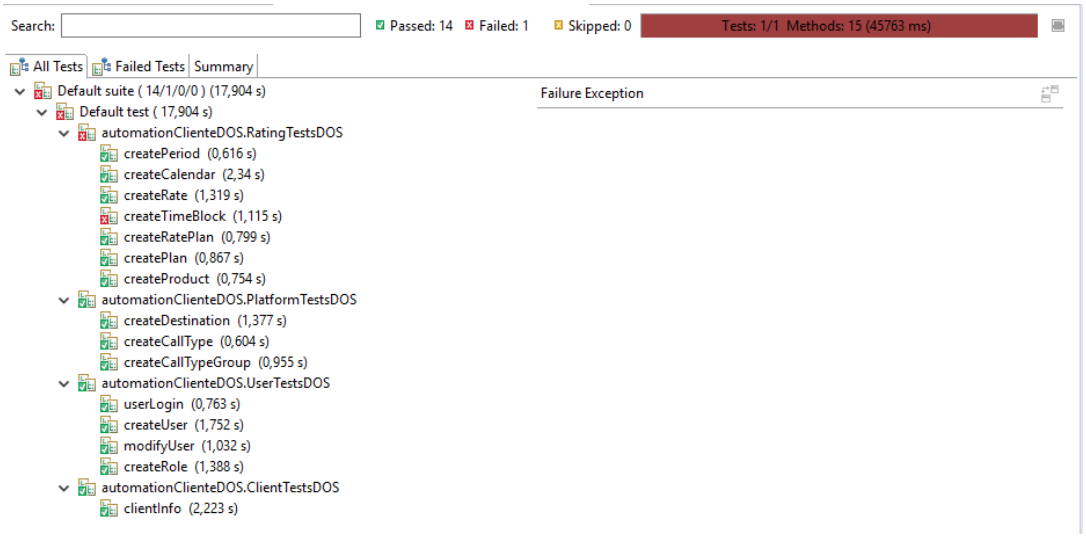


Figura 19: Selenium Webdriver, Google Chrome - Resultado pruebas automatizadas Chile

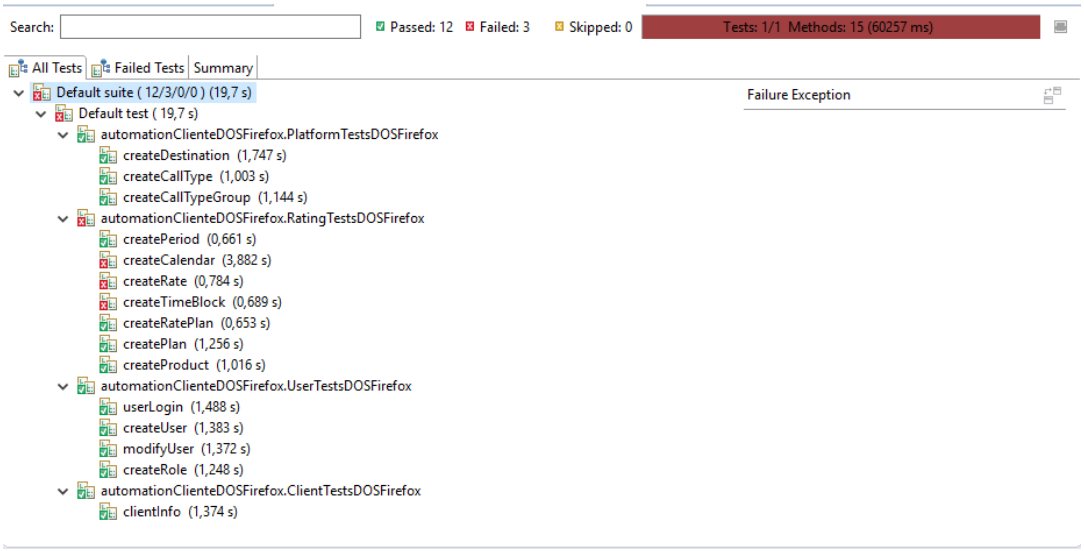


Figura 20: Selenium Webdriver, Mozilla Firefox - Resultado pruebas automatizadas Chile

Los resultados obtenidos en las pruebas automatizadas coinciden con los obtenidos en las pruebas manuales, siendo las mismas pruebas las que presentaron fallas debido a errores de las plataformas.

4.3 Análisis de Resultados

El objetivo de la realización de pruebas manuales y automatizadas es la comparación en los esfuerzos y tiempos requeridos para llevar a cabo cada proceso. Para ello fue presentado en la sección 4.1 los esfuerzos requeridos para cada una de las etapas del proceso de automatización, análisis y construcción de pruebas, construcción de pruebas automatizadas y ejecución manual y automática de estas.

A continuación se presenta un resumen de los esfuerzos obtenidos (en Horas Hombre) desde las tablas 3, 4, 5, 6 y 7.

Tabla 14: Resumen esfuerzos por proyecto

Proyecto	Diseño y Elaboración Pruebas Funcionales	Diseño y Construcción Pruebas automatizadas	Ejecución Manual - Promedio Iteración	Ejecución automatizada - Promedio Iteración
CA	7	22,5	0,224	0,031
CL	4	10,5	0,176	0,006

Las siguientes figuras presentan una comparación entre los esfuerzos requeridos para la ejecución manual y la ejecución automática de las pruebas construidas.

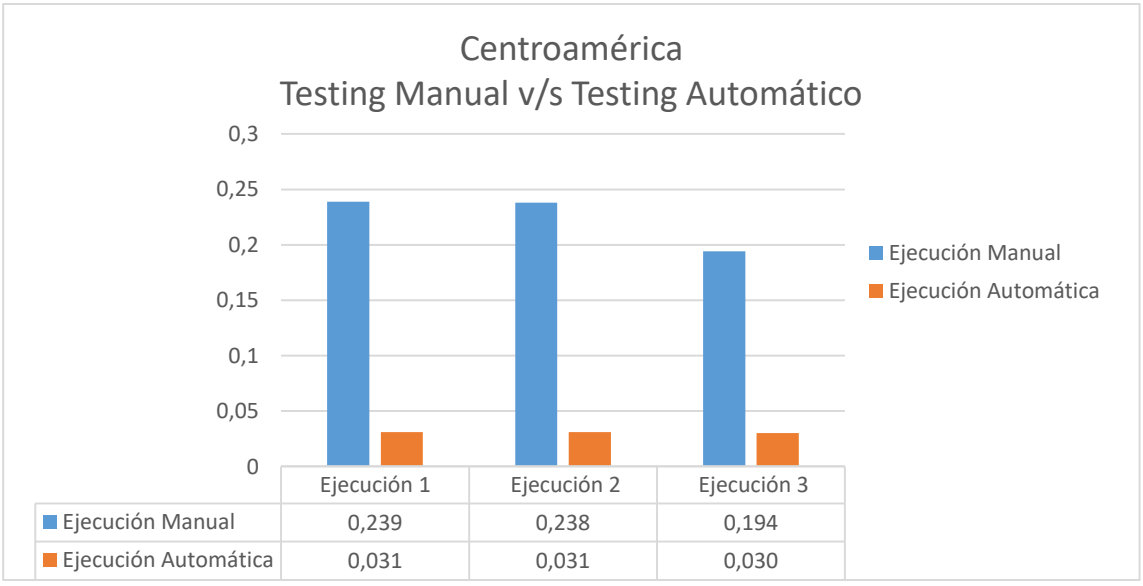


Figura 21: Comparación esfuerzos de ejecución Centroamérica

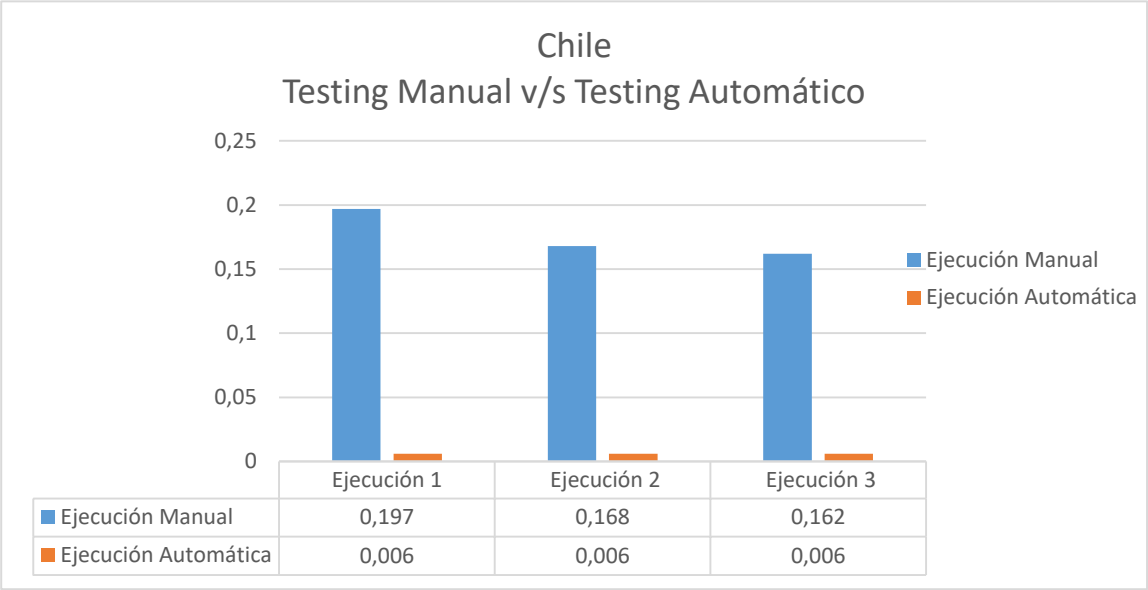


Figura 22: Comparación esfuerzos de ejecución Chile

Se aprecia una notoria diferencia entre los esfuerzos de ejecución necesarios para la realización de cada iteración, en donde las pruebas automatizadas requieren entre un 4 y un 15% del esfuerzo requerido en una ejecución manual.

Ahora bien, el propósito de la construcción de pruebas automatizables es repetirlas una y otra vez, cada vez que se realice un cambio en el producto de software sobre el cual se requiere realizar *testing*. Es por esto que fueron elaboradas un conjunto de tablas en las cuales se busca conocer que tan efectiva es la ejecución de las pruebas automatizadas sobre las pruebas manuales.

Para la realización de pruebas y posterior construcción de las tablas se utilizó la siguiente lógica:

- La ejecución de pruebas manuales solo está condicionado por el diseño y elaboración de las pruebas funcionales. Por tanto se utiliza como tiempo base el esfuerzo requerido para tal ítem.
- La ejecución de pruebas automatizadas está condicionado por el diseño y elaboración de las pruebas funcionales, y el diseño y construcción de las pruebas automatizadas. Por tanto se utiliza como tiempo base la suma de los esfuerzos requeridos para ambos ítems.
- La presentación de esfuerzo es en Horas Hombre (HH)

A continuación se presentan las tablas de esfuerzos (en HH) según el número de iteraciones para el proyecto en Centroamérica y el proyecto en Chile.

Tabla 15: Esfuerzos según número de iteraciones - Centroamérica

Nro. Iteraciones	Ejecución Manual	Ejecución Automática
1	7,224	29,531
2	7,448	29,562
3	7,672	29,593
4	7,896	29,624
5	8,120	29,655
10	9,240	29,810
20	11,480	30,120
30	13,720	30,430
40	15,960	30,740
50	18,200	31,050
60	20,440	31,360
70	22,680	31,670
80	24,920	31,980
90	27,160	32,290
100	29,400	32,600
110	**31,640**	**32,910**
130	**36,120**	**33,530**
150	40,600	34,150

Tabla 16: Esfuerzos según número de iteraciones - Chile

Nro. Iteraciones	Ejecución Manual	Ejecución Automática
1	4,176	14,506
2	4,352	14,512
3	4,528	14,518
4	4,704	14,524
5	4,880	14,530
10	5,760	14,560
20	7,520	14,620
30	9,280	14,680
40	11,040	14,740
50	12,800	14,800
60	**14,560**	**14,860**
70	**16,320**	**14,920**
80	18,080	14,980
90	19,840	15,040
100	21,600	15,100

Según el número de iteraciones y el esfuerzo requerido para ellas es posible determinar el punto en que la construcción y ejecución de pruebas automatizadas es más rentable que la ejecución de pruebas manuales.

Para el caso de Centroamérica es posible notar que el esfuerzo de ejecutar pruebas automatizadas es más rentable alrededor de la iteración número 110, en el caso de Chile comienza alrededor de la iteración número 60.

A continuación, y para presentar la información obtenida de manera más gráfica y precisa se construyeron gráficos en los cuales se percibe el punto en el cual el esfuerzo de las ejecuciones automatizadas comienza a ser menor que el esfuerzo requerido para seguir realizando pruebas manuales.

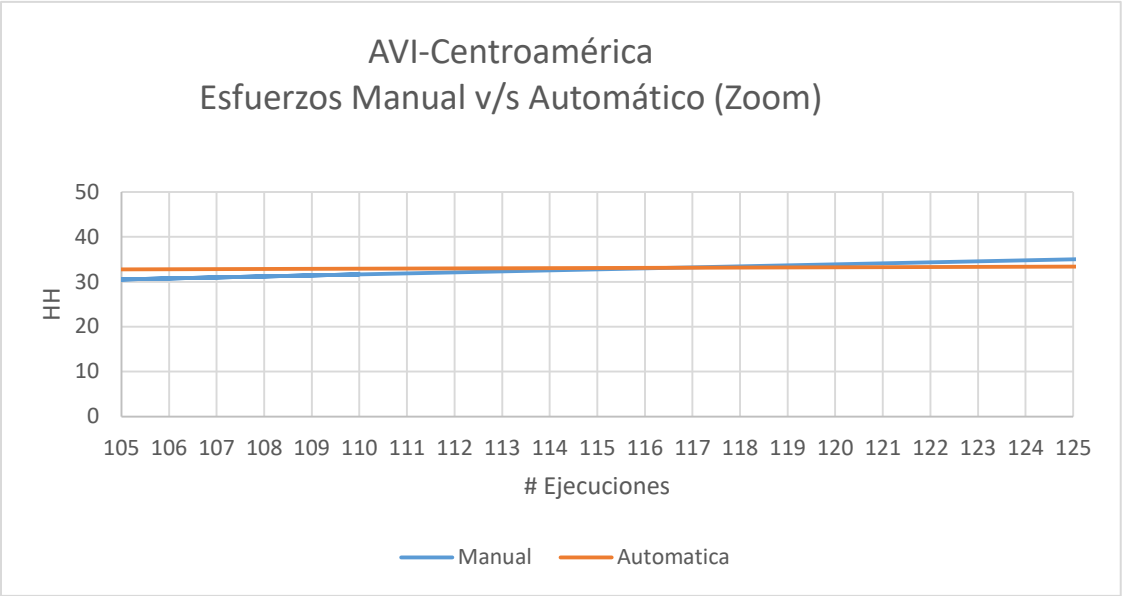


Figura 23: Esfuerzo según número iteraciones - Centroamérica

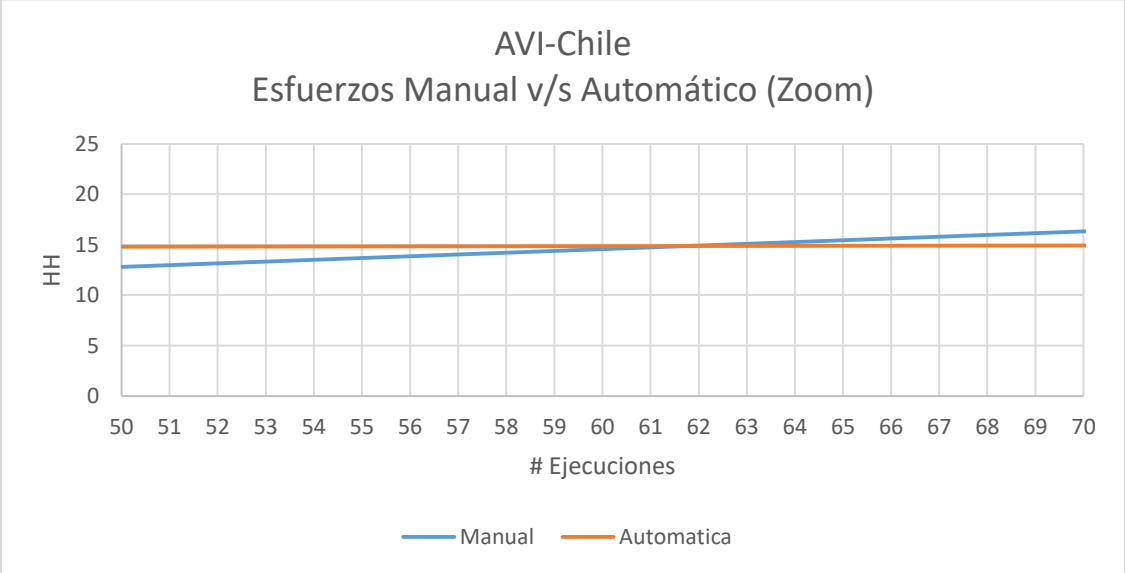


Figura 24: Esfuerzo según número iteraciones - Chile

Las figuras 23 y 24 entregan una visión más precisa del punto que la ejecución automática es más rentable que la ejecución manual. Para el caso de Centroamérica esto es luego de la iteración 117, y en el caso de Chile correspondería en la iteración 62.

Un dato de tremenda importancia que puede ser obtenido posterior a una inversión es el *ROI.* Para calcular este valor nos basamos en los resultados obtenidos en las tablas 14,15 y 16, y en la siguiente formula:

$$\frac{\boldsymbol{N} * (|\boldsymbol{PEM} - \boldsymbol{PEA}|) - \boldsymbol{TID}}{\boldsymbol{TID}} * \mathbf{100}$$

Figura 25: Ecuación Calculo ROI

Donde:

N: Corresponde al número de iteraciones de ejecuciones

PEM: Promedio de tiempo Ejecuciones Manuales

PEA: Promedio de tiempo Ejecuciones Automáticas

TID: Tiempo de Inversión en Diseño

Para el cálculo de este dato se consideró lo siguiente:

- El costo de inversión corresponde al tiempo de dedicación para el diseño y elaboración de las pruebas automatizadas.
- La ganancia de la inversión corresponde a la diferencia de tiempo entre la ejecución manual frente a la ejecución automática del conjunto de pruebas en valor absoluto, este valor representa el ahorro de tiempo al decidir automatizar las pruebas. Este resultado es multiplicado por el número de iteraciones realizadas.

En la siguiente tabla se presentan los ROI obtenidos según el número de iteraciones de ejecuciones.

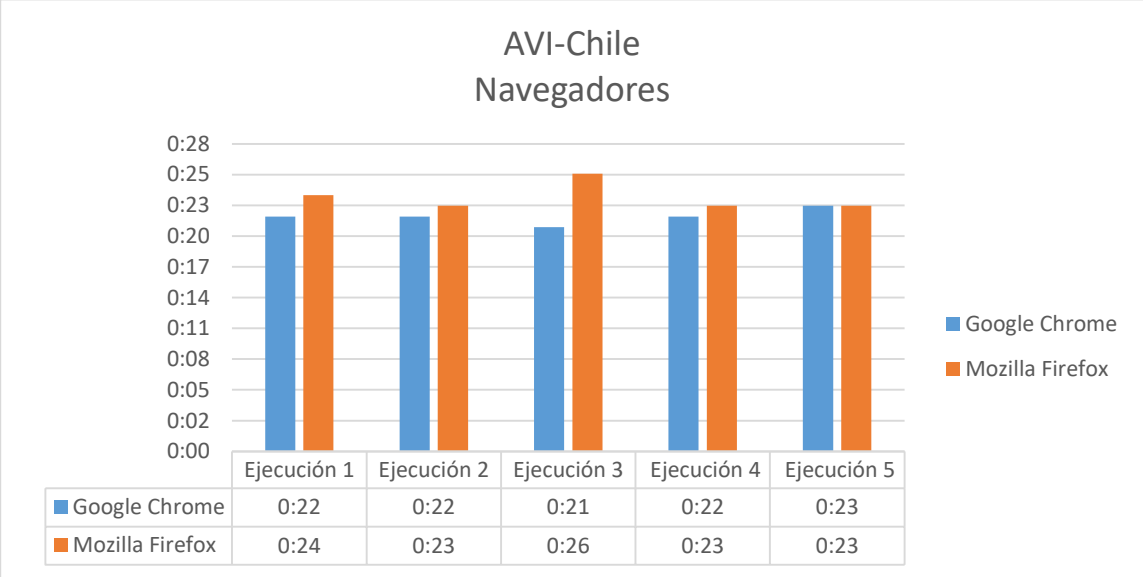
Tabla 17: Retorno de la inversión según número de iteraciones

ITERACIONES	ROI CA (%)	ROI CL (%)
1	-99,142	-98,381
2	-98,284	-96,762
3	-97,427	-95,143
4	-96,569	-93,524
5	-95,711	-91,905
10	-91,422	-83,810
15	-87,133	-75,714
20	-82,844	-67,619
30	-74,267	-51,429
40	-65,689	-35,238
50	-57,111	-19,048
62	-46,818	**0,381**
70	-39,956	13,333
100	-14,222	61,905
117	**0,360**	89,429
150	28,667	142,857
200	71,556	223,810

En base a la tabla 17, es posible notar que el retorno de la inversión aumenta considerablemente mientras más iteraciones se realicen. Por el contrario, si se realizan pocas iteraciones el *ROI* obtenido es negativo. Se remarcan los puntos en que el *ROI* obtenido es prácticamente 0, que corresponde a aquellos en los cuales la ejecución de pruebas automatizadas es igual de rentable que la ejecución de pruebas manuales, iteración 62 para CL e iteración 117 para CA.

Un punto añadido de comparación es el desempeño de distintos navegadores cuando se realizan las mismas pruebas. Para este análisis se evaluaron dos navegadores previamente mencionados: *Google Chrome* y *Mozilla Firefox*.

A continuación se presenta un gráfico comparativo del desempeño de estos navegadores en el proyecto Chile, medido sobre 5 iteraciones de pruebas automatizadas. El desempeño es medido en segundos de ejecución, en donde menor tiempo de ejecución es mejor.



	Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5
Google Chrome	0:22	0:22	0:21	0:22	0:23
Mozilla Firefox	0:24	0:23	0:26	0:23	0:23

Figura 26: Comparación tiempos de ejecución Navegadores

Se puede observar que los tiempos de ejecución en el navegador *Google Chrome* son marginalmente menores que en el navegador *Mozilla Firefox* en cada una de las iteraciones, pero si se toma en consideración que el propósito de las pruebas automatizadas es que estas sean realizadas un gran número de veces entonces los pocos segundos que se observan de diferencia entre ambos navegadores pueden resultar siendo minutos e inclusos horas de tiempo de ejecución.

CAPÍTULO 5: CONCLUSIONES

5.1 Conclusiones Generales

Durante el desarrollo de este trabajo se logró analizar, elaborar y construir un conjunto de pruebas de regresión automatizadas, basadas en ciertas funcionalidades presentes en la plataforma utilizada. El enfoque de estas pruebas es que su ejecución fuese repetible cada vez que se realiza un cambio o ajuste en alguna funcionalidad de la plataforma, con el fin de asegurar el no impacto en las funcionalidades sobre las cuales se construyó este conjunto de pruebas.

La utilización de herramientas enfocadas en la construcción de pruebas automatizadas fue un punto vital para lograr este cometido, tanto la herramientas *Selenium Webdriver* como *SoapUI* demostraron gran utilidad a la hora de programar los conjuntos de pruebas. Por un parte *Selenium Webdriver* posee una interesante gama de lenguajes de programación disponibles para su uso, en particular para este trabajo se utilizó el lenguaje Java debido al conocimiento adquirido sobre este. La herramienta aprovecha las cualidades presentes en este lenguaje, tal como su enfoque orientado a objetos visto en la inclusión de *PageObjects,* lo cual facilita la construcción de pruebas con bases similares. Por otra parte *SoapUI* es una herramienta sencilla de utilizar, se aprovecha la inclusión de *webservices* en las plataformas para construir conjuntos de pruebas de manera ágil y sin necesidad de poseer conocimiento especializado en programación.

El grueso del trabajo recae en la ejecución de las pruebas construidas, en donde fue posible notar la rapidez con la que se ejecutan las pruebas automatizadas gracias a estas herramientas, tardando tan solo un par de segundos frente a los minutos que toma realizar este mismo conjunto de pruebas de forma manual. Ahora bien, los resultados comparativos respecto a los tiempos de ejecución no presentan el cuadro completo, si bien la evaluación frente a frente de los tiempos de ejecución de las pruebas construidas dan como ganador indiscutible a las pruebas automatizadas, es necesario recalcar que el esfuerzo requerido para la construcción de estas pruebas no es menor. Para este caso particular se construyeron pruebas sencillas, que recaen principalmente en el conocido *CRUD (Create, Read, Update, Delete)* sobre una web que posee como *backend* una base de datos relacional, por ende el análisis, diseño y construcción no tardo un tiempo considerable, pero existe la necesidad de que este análisis sea realizado para construir pruebas candidatas a la automatización, por tanto, se recalca que el esfuerzo dedicado a este ítem no es trivial, es más, impacta directamente en la decisión de optar por la construcción o no de pruebas automatizadas.

Los resultados finales obtenidos presentan el punto en que la construcción de pruebas automatizadas comienza a ser más eficiente que la realización de pruebas manuales.

Si bien a primera vista la cantidad de iteraciones requeridas para esto es alto, se debe tener en consideración que el propósito de la automatización de pruebas regresivas es evaluar el producto de software cada vez que se genere un cambio, por menor que este sea, ya que de esta manera comprobamos la integridad y el no impacto de las funcionalidades del software.

Los *ROI* obtenidos respaldan esta afirmación, mientras más veces se repitan el conjunto de pruebas automatizadas, mayor es el retorno, especialmente a partir de los puntos críticos donde la ejecución automática comienza a ser más rentable.

Finalmente, optar por diseñar y construir pruebas automatizadas permite mantener durante el ciclo de vida del *software* un control sobre el estado del producto diseñado, pues alerta de cambios y errores provocados por la inclusión de nuevas funcionalidades o ajustes en las ya presentes en el producto base. En el largo plazo y si se aplican reiterados cambios de versiones sobre el producto no solo facilitará el trabajo de detección de errores sino también disminuirá los tiempos de dedicación necesarios para su detección.

5.2 Cumplimiento de objetivos

En la sección 1.3 se definieron los objetivos específicos que se buscaban cumplir con este trabajo. A continuación se presenta un análisis del cumplimiento de cada uno de ellos.

- **Analizar y evaluar herramientas *Open Source* para crear y ejecutar portafolios de casos de pruebas automatizables y reutilizables.**
  Se utilizaron dos herramientas *Open Source* previamente utilizadas fuera del marco de este trabajo, presentadas en la sección 2.4. La decisión de utilizar estas herramientas fue debido al conocimiento adquirido sobre estas en proyectos personales, además de apuntar directamente a las aplicaciones que se deseaban probar. Por una parte *Selenium Webdriver* facilita la interacción con aplicativos web gracias a sus diversos navegadores y al uso de las herramientas disponibles según el lenguaje de programación que se desee utilizar, en este caso la orientación a objetos presente en Java facilitó las tareas requeridas de automatización. En conjunto a esta herramienta *TestNG* permite agrupar los casos de pruebas construidos en conjuntos de pruebas independientes

entre sí, lo cual facilita la evaluación de aquellas pruebas que poseen algunas características en común.

Por otra parte *SoapUI* es un software muy sencillo de usar y apunta a la ejecución y construcción de casos de pruebas sobre *webservices*, tan solo es necesario disponer de estos y conocer el esquema de consultas necesarios para su invocación.

- **Elaborar casos de pruebas de regresión aplicables sobre desarrollos, apoyándose de las herramientas disponibles.**

  Utilizando los requerimientos presentados en la sección 3.2 en conjunto con la estructura de pruebas definido en la sección 3.3 se obtuvo una buena base para diseñar y construir las pruebas automatizadas en las herramientas seleccionadas. El diseño mostrado en la sección 3.4 presenta la estructura de cómo fueron construidas las pruebas automatizadas para cada una de las herramientas. De esta manera se logró conjuntos de pruebas por módulo en donde cada uno de estos contiene pruebas que poseen relación entre sí. Se incluyeron mejoras tales como *PageObjects* para abstraerse de los contenidos necesarios en cada una de las clases construidas, y de esta manera aprovechar las características de un lenguaje orientado a objetos.

- **Aplicar y ejecutar herramientas y casos de pruebas creados sobre proyectos en desarrollo.**

  Al utilizar herramientas las cuales entregan como salida información de ejecución de las pruebas diseñadas, fue posible presentar en la sección 4.1 los tiempos requeridos para ejecutar los *suites* de pruebas construidos en conjunto con los resultados de estos *suites* y cada prueba individualmente, tal como se presenta en la sección 4.2. Gracias a estas facilidades que presentan las herramientas se pudo realizar un análisis comparativo de manera más fehaciente, apoyado de los resultados obtenidos desde las mismas herramientas.

- **Medir y comparar los esfuerzos entre la ejecución de pruebas automatizadas y ejecución de pruebas manuales.**

  El proceso de medición y comparación fue realizado analizando los tiempos de ejecución de las pruebas diseñadas en la sección 3.4 y presentados en la sección 4.3, en donde se puede apreciar que la elaboración de pruebas automatizadas es una muy eficiente opción frente a las pruebas manuales cuando se trata de proyectos en los cuales se sabe que se van a realizar muchos cambios, lo cual se traduce en un alto número de iteraciones y por tanto en la posibilidad de disminuir los esfuerzos requeridos en validar las funcionalidades bases mediante pruebas de regresión manuales.

5.3 Recomendaciones Futuras

Para este proyecto se presentaron un conjunto reducido de pruebas simples, con la finalidad de evaluar, mediante una muestra pequeña, el desempeño de las herramientas escogidas. Un siguiente paso interesante sería evaluar el desempeño de la herramienta frente a pruebas de alta complejidad tales como el ingreso a *frames* embebidos en una página web.

Como en todo desarrollo de software, una de las mejoras prioritarias es escribir un código fuente eficiente, aprovechando las herramientas y virtudes que entrega el lenguaje de programación escogido. Para estos efectos implementar mejoras en el uso de *PageObjects* es primordial, aún más considerando el enfoque orientado a objetos que impulsa el uso de Java. Modularidad, reutilización de código, herencias ayudará a ordenar el código y hacerlo más eficiente a la hora de ejecutar y depurar.

Incluir un enfoque de integración continua, mediante servidores de automatización tales como *Jenkins*[22] con un enfoque en automatizar ciertas tareas del desarrollo de software, como la compilación y la ejecución de pruebas cada vez que se realiza un *commit* sobre el código fuente del software, y todo lo que esto conlleva.

[22] Un servidor de automatización *open source* escrito en Java. https://jenkins.io/

REFERENCIAS BIBLIOGRÁFICAS

Beust, C. (2017). *TestNG*. Recuperado el 15 de Enero de 2018, de http://testng.org/doc/

Beust, C., & Suleiman, H. (2007). *Next Generation Java Testing TestNG and Advanced Concepts.* New Jersey: Addison-Wesley.

Garg, N. (2014). *Test Automation Using Selenium WebDriver with Java.* AdactIn Group .

IEEE. (1991). IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. En *IEEE Software Engineering Standards Collection* (Corregida, Febrero 1991 ed.). New York.

ISO. (2017). *ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Recuperado el 07 de Marzo de 2018, de https://www.iso.org/standard/35733.html

Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons Learned in Software Testing.* New York: Wiley.

Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance Theory and Practice.* Hoboken, New Jersey: Jhon Wiley & Sons.

Rätzmann, M., & De Young, C. (2003). *Galileo Computing Software Testing and Internationalization.* Salt Lake City: Lemoine International.

Selenium Project. (2018). *Selenium - Web Browser Automation*. Recuperado el 7 de Enero de 2018, de https://www.seleniumhq.org/

SmartBear Software. (2018). *The World's Most Popular API Testing Tool | SoapUI*. Obtenido de https://www.soapui.org/

Viraj, T. (2016). *Quality Assurance in Agile Software Development*. Recuperado el 12 de Enero de 2018, de http://blingtechs.blogspot.cl/2016/01/quality-assurance-in-agile-software.html