



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE SISTEMAS E INFORMÁTICA

**TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO
DE INGENIERO EN SISTEMAS E INFORMÁTICA**

**“IMPLEMENTACIÓN DE UN SOFTWARE PARA LA EJECUCIÓN DE
PRUEBAS EN APLICACIONES WEB EXTENDIENDO LA
HERRAMIENTA SELENIUM. CASO DE ESTUDIO EN EL PRODUCTO
GESTOR G5 TRUST”**

AUTORES:
PONCE GUERRÓN, ALEJANDRA DENNIS
NARANJO ERAZO, RAÚL DAVID

DIRECTOR:
ING. CORAL CORAL, HENRY RAMIRO

Sangolquí

2018



ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación "IMPLEMENTACIÓN DE UN SOFTWARE PARA LA EJECUCIÓN DE PRUEBAS EN APLICACIONES WEB EXTENDIENDO LA HERRAMIENTA SELENIUM. CASO DE ESTUDIO EN EL PRODUCTO GESTOR G5 TRUST", realizado por los señores: ALEJANDRA DENNIS PONCE GUERRÓN y RAÚL DAVID NARANJO ERAZO, ha sido revisado en su totalidad y analizado por el software anti-plagio, el mismo que cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de las Fuerzas Armadas ESPE, por lo tanto me permito acreditarlo y autorizar a los señores ALEJANDRA DENNIS PONCE GUERRÓN y RAÚL DAVID NARANJO ERAZO para que lo sustenten públicamente.

Sangolquí, 14 de febrero del 2018

Ing. Henry Ramiro Coral Coral

DIRECTOR



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE SISTEMAS E INFORMÁTICA

AUTORÍA DE RESPONSABILIDAD

Nosotros, Alejandra Dennis Ponce Guerrón y Raúl David Naranjo Erazo, con cédulas de identidad No 0503337909 y 1722487483 respectivamente, declaramos que este trabajo de titulación "Implementación de un software para la ejecución de pruebas en aplicaciones web extendiendo la herramienta Selenium. Caso de estudio en el producto GESTOR G5 TRUST", ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaramos que este trabajo es de nuestra autoría, en virtud de ello nos declaramos responsables del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 20 de febrero del 2018

Alejandra Dennis Ponce Guerrón

0503337909

Raúl David Naranjo Erazo

1722487483



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN

Nosotros, Alejandra Dennis Ponce Guerrón y Raúl David Naranjo Erazo, autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar en la biblioteca Virtual de la institución el presente trabajo de titulación "Implementación de un software para la ejecución de pruebas en aplicaciones web extendiendo la herramienta Selenium. Caso de estudio en el producto GESTOR G5 TRUST", cuyo contenido, ideas y criterios son de nuestra autoría y responsabilidad.

Sangolquí, 20 de febrero del 2018



Alejandra Dennis Ponce Guerrón

0503337909



Raúl David Naranjo Erazo

1722487483

DEDICATORIA

A las tres mujeres más importantes de mi vida.

Alejandra Ponce

A mi madre que siempre me apoyó.

Raúl Naranjo

AGRADECIMIENTO

Agradezco a mi familia y amigos que estuvieron a mi lado durante todos estos años de vida universitaria.

Alejandra Ponce

Doy gracias a mi madre y hermano que confiaron en mí y me dieron todo su apoyo.

Raúl Naranjo

ÍNDICE

CARÁTULA

CERTIFICADO DE TUTOR.....i

AUTORÍA DE RESPONSABILIDAD.....ii

AUTORIZACIÓN.....iii

DEDICATORIA.....iv

AGRADECIMIENTO.....v

ÍNDICE.....vi

ÍNDICE DE TABLAS.....x

ÍNDICE DE FIGURAS.....x

GLOSARIO.....xii

RESUMEN.....xviii

ABSTRACT.....xix

CAPÍTULO I

DEFINICIÓN DEL PROYECTO.....1

1.1. Antecedentes.....1

1.2. Planteamiento del problema.....2

1.3. Objetivos.....3

1.3.1. Objetivo General.....3

1.3.2. Objetivos Específicos.....	4
1.4. Justificación.....	4
1.5. Alcance.....	5
1.6. Selección de la metodología de investigación.....	6
CAPÍTULO II	
ESTADO DE LA CUESTIÓN.....	9
2.1. Gestión de calidad en la ingeniería de software.....	9
2.2. Aseguramiento de la calidad.....	12
2.3. Pruebas.....	14
2.3.1. Clasificación de técnicas de pruebas	22
2.4. Aplicaciones web.....	32
2.5. Selenium.....	34
2.5.1. Componentes de Selenium.....	35
2.5.2. Automatización de pruebas para aplicaciones web.....	39
2.5.3. Investigaciones de la academia	40
2.6. Estándares utilizados para definir y medir la calidad.....	43
2.6.1. ISO/IEC 25010.....	44
2.6.2. ISO/IEC 25040.....	46
CAPÍTULO III	
METODOLOGÍA DE LA INVESTIGACIÓN.....	48

CAPÍTULO IV

SOFTWARE PARA AUTOMATIZACIÓN DE PRUEBAS.....	53
4.1. Proyecto Java.....	53
4.2. Selenium.....	59
4.3. Pruebas funcionales.....	63
4.4. Pruebas de regresión.....	64
4.5. Reportes.....	68

CAPÍTULO V

DESARROLLO DEL CASO DE ESTUDIO.....	72
5.1. Título.....	72
5.2. Autor.....	72
5.3. Resumen.....	72
5.4. Introducción.....	73
5.4.1. Planteamiento del problema.....	73
5.4.2. Objetivos de la investigación.....	75
5.4.3. Contexto.....	75
5.5. Diseño del caso de estudio.....	76
5.5.1. Preguntas de la investigación.....	76
5.5.2. Selección del caso de estudio y unidades de análisis.....	76
5.5.3. Procedimientos de recolección de datos.....	77

5.5.4. Procedimientos de análisis de datos	82
5.5.5. Procedimientos de validación de la información	83
5.6. Resultados.....	83
5.6.1. Discusión.....	89
5.6.2. Consideraciones.....	95
CAPÍTULO VI	
CONCLUSIONES Y TRABAJOS FUTUROS.....	100
4.1. Conclusiones.....	100
4.2. Trabajos Futuros.....	102
BIBLIOGRAFÍA.....	105

ÍNDICE DE TABLAS

Tabla 1	<i>Proceso de evaluación de atributos de calidad de GESTOR G5 TRUST</i>	79
Tabla 2	<i>Descripción de las pruebas funcionales</i>	80
Tabla 3	<i>Descripción de las pruebas de regresión</i>	81
Tabla 4	<i>Tiempo de creación y ejecución de pruebas manuales</i>	84
Tabla 5	<i>Tiempo de creación y ejecución de pruebas automatizadas</i>	86
Tabla 6	<i>Diferencia de tiempos entre pruebas manuales y automatizadas</i>	88
Tabla 7	<i>Resumen comparativo de los tiempos para pruebas manuales y automáticas</i>	89
Tabla 8	<i>Porcentaje de incremento de tiempo para pruebas funcionales</i>	101
Tabla 9	<i>Número de horas extra en pruebas de regresión</i>	102

ÍNDICE DE FIGURAS

Figura 1	<i>Proceso de Gestión de la Calidad</i>	10
Figura 2	<i>Relación del costo vs el tiempo de corregir defectos</i>	13
Figura 3	<i>Clasificación de técnicas estáticas</i>	21
Figura 4	<i>Clasificación de técnicas dinámicas</i>	22
Figura 5	<i>Arquitectura Selenium WebDriver</i>	38
Figura 6	<i>Modelo de calidad del software de acuerdo a la ISO 25010</i>	45
Figura 7	<i>Proceso de evaluación de la calidad</i>	46
Figura 8	<i>Estructura módulo dbTesting</i>	54
Figura 9	<i>Estructura módulo demonioTesting</i>	55
Figura 10	<i>Módulo demonioTesting en ejecución</i>	55

Figura 11 Módulo demonioTesting ejecutando prueba	56
Figura 12 Estructura módulo feTesting	57
Figura 13 Ventana administración casos de prueba	58
Figura 14 Ventana administración suites de prueba	58
Figura 15 Ventana administración Pruebas	59
Figura 16 Acciones grabadas por plugin Selenium	60
Figura 17 Opción para exportar casos a JUnit 4	61
Figura 18 Fragmento código generado por Selenium	62
Figura 19 Fragmento de código generado por Proyecto Testing	63
Figura 20 Ejemplo prueba funcional	64
Figura 21 Ejemplo prueba de regresión.....	65
Figura 22 Diagrama físico de la base de datos	66
Figura 23 Script consulta Pruebas terminadas	66
Figura 24 Resultado consulta Pruebas terminadas.....	67
Figura 25 Script consulta pruebas con resultado correcto	67
Figura 26 Resultado consulta pruebas con resultado correcto	68
Figura 27 Reporte de resultados de las pruebas	69
Figura 28 Reporte detallado del resultado de las pruebas	70
Figura 29 Diagrama de Ishikawa de la problemática	73
Figura 30 Esquema de caso de estudio y unidad de análisis.....	76
Figura 31 División de la característica de mantenibilidad según ISO 25010.	78
Figura 32 Gráfica de incremento de tiempo en pruebas funcionales	91
Figura 33 Número de horas ahorradas con pruebas de regresión automatizadas	93

GLOSARIO

ACS: Acrónimo de Aseguramiento de la Calidad de Software (del inglés Software Quality Assurance). El conjunto de actividades sistemáticas que se desarrollan para proporcionar evidencia de la idoneidad para el uso del producto software en su totalidad se conoce como aseguramiento de calidad (ISO/IEC 25010:2011, 2011).

Ad-hoc: “Pruebas ad hoc son pruebas llevadas a cabo de manera informal; no se realiza una preparación formal de la prueba, no se utilizan técnicas de diseño reconocidas, no existen expectativas para con los resultados y la arbitrariedad guía la actividad de ejecución” (Globe Testing, 2017).

Aplicación web: “En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador” (López, 2015).

Caso de estudio: “Llamamos casos a aquellas situaciones o entidades sociales únicas que merecen interés de investigación. Así, por ejemplo en educación, un aula, un alumno autista o un programa de enseñanza pueden considerarse un caso (Barrio del Castillo, 2018). El estudio de casos puede incluir tanto estudios de un solo caso como de

múltiples casos (según sea una o varias las unidades de análisis) pero su propósito fundamental es comprender la particularidad del caso, en el intento de conocer cómo funcionan todas las partes que los componen y las relaciones entre ellas para formar un todo” (Muñoz y Serván, 2001).

Combobox: es un tipo de control que se utiliza para mostrar datos en un cuadro combinado desplegable y seleccionar uno de ellos.

Demonio: “En sistemas UNIX se conoce como demonio o daemon (Disk And Execution Monitor) a un proceso que se ejecuta en segundo plano del sistema operativo, se ejecuta en todo momento y no posee interacción directa con el usuario, también se le conoce genéricamente como servicio o proceso, del cual no percibimos su ejecución. Un demonio realiza una operación específica en tiempos predefinidos o en respuesta a ciertos eventos del sistema. El origen del nombre se refiere a su función de *vigilante* y no a una connotación negativa como se podría pensar en un principio” (Universidad Técnica Federico Santa María, 2012)

Feedback: “Puede traducirse como realimentación o retroalimentación. Se trata de la alimentación de un sistema a través del regreso de un sector o de un porcentaje de su salida. Es un mecanismo que supone el retorno de una parte de los elementos que salen del sistema. Esto quiere decir que algo sale del sistema y luego, al menos un fragmento o una proporción, vuelve a ingresar al mismo. Muchas veces el regreso de la

salida permite corregir el sistema en cuestión a partir de los datos que ingresan” (Pérez Porto & Merino, 2017).

Framework: “Es a menudo una estructura en capas que indica qué tipo de programas pueden o deben ser construidos y cómo se interrelacionan. Algunos marcos de trabajo de sistemas informáticos también incluyen programas reales, especifican interfaces de programación u ofrecen herramientas de programación para usar los marcos. Un framework puede servir para un conjunto de funciones dentro de un sistema y cómo se interrelacionan; las capas de un sistema operativo; las capas de un subsistema de aplicación; cómo debería normalizarse la comunicación en algún nivel de una red; etcétera. Un marco de trabajo es generalmente más completo que un protocolo y más prescriptivo que una estructura” (Rouse, 2016).

GCS: Acrónimo para Gestión de la Calidad del Software (del inglés “Software Quality Management”). Es la recopilación de todos los procesos que garantizan que los productos de software, los servicios y las implementaciones de procesos del ciclo de vida cumplan los objetivos de calidad del software de la organización y logren la satisfacción de los interesados (IEEE Std. 12207-2008, 2008).

HCI (Human Computer Interaction): “La Interacción Persona-Ordenador, es la disciplina que estudia el intercambio de información entre las personas y los ordenadores. Su objetivo es que este intercambio sea más eficiente: minimiza los errores, incrementa

la satisfacción, disminuye la frustración y en definitiva, hace más productivas las tareas que envuelven a las personas y los ordenadores” (Manchón, 2003).

IDE (Entorno de Desarrollo Integrado): “Un entorno de desarrollo integrado, es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java, C#, Delphi, Visual Basic, etc.” (fergarciac, 2013).

Lenguaje de programación: Lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas (Alegsa, 2010).

Log: También conocido como registro “es la documentación producida automáticamente y con sello de tiempo de los eventos relevantes para un sistema en particular. Prácticamente todas las aplicaciones de software y sistemas producen archivos de registro” (WhatIs.com, 2017).

Plugin: También Plug-in, es un “Programa que puede anexarse a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la aplicación principal). No se trata de un parche ni de una actualización, es un módulo aparte que se incluye opcionalmente en una aplicación” (Alegsa, 2010)

Renderizar: en el ámbito de aplicaciones web, rendizar una página involucra todas las acciones que permiten analizar, procesar y finalmente visualizar en la pantalla los elementos de dicha página.

RollOver: es evento del ratón que constituye en mover el cursor sobre un elemento de la pantalla.

Stakeholder: en español se le conoce como los interesados. El término hace referencia a un individuo que tiene un interés particular en uno o varios elementos del software. Ejemplos de stakeholder son usuarios, dueños del producto, personas que detallan los requerimientos de software, etc.

Test: Un caso de prueba o test case es, en ingeniería del software, un conjunto de condiciones o variables bajo las cuáles un analista determinará si una aplicación, un sistema software (software system), o una característica de éstos es parcial o completamente satisfactoria.

Tester: “Un tester investiga un producto de software con el objetivo de obtener información acerca de su calidad y del valor que representa para quienes lo utilizan. Asume el desafío de detectar la mayor cantidad de fallas severas (incidentes de alto impacto) con el mínimo esfuerzo, antes de que el software salga a producción” (CES, 2009).

Testing: “El testing es una actividad desarrollada para evaluar la calidad del producto software, y para mejorarlo al identificar defectos y problemas. Consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto finito de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado” (Cristiá, 2009).

RESUMEN

La mayoría de aplicaciones que se desarrollan en la industria del software hoy en día son orientadas a la web, y como parte fundamental del proceso de desarrollo del producto, los esfuerzos de la ingeniería de software por desarrollar cada vez métodos más efectivos con los que asegurar la calidad del producto se mantienen. Así, la automatización del proceso de pruebas se ha vuelto más que necesaria en el mercado global. A pesar de ello, se ha observado que muchas industrias ecuatorianas aún utilizan métodos manuales para llevar a cabo sus pruebas, ya sea esto por la complejidad del proceso de automatización o los costos involucrados que aparentemente no representan una inversión conveniente. Por ello, esta investigación se propuso llevar a cabo un caso de estudio donde se implementaron pruebas funcionales y de regresión extendiendo la herramienta libre Selenium y estableciendo un análisis de los beneficios de la automatización de pruebas con el producto de la empresa ecuatoriana GESTORINC S.A, en términos de cuánto mejoraron los atributos de calidad del software previo y post a la automatización. Los resultados de la investigación son favorables señalando un nivel medio de complejidad en el proceso de automatización de pruebas, así como también demostrando en el análisis de costo-beneficio altos porcentajes de mejora en la efectividad para hacer pruebas y en la capacidad del software para ser modificado.

Palabras clave:

- **PRUEBAS FUNCIONALES**
- **PRUEBAS DE REGRESIÓN**
- **AUTOMATIZACIÓN DE PRUEBAS**
- **SELENIUM**

ABSTRACT

The majority of today's applications developed in the software industry are web-oriented, and as a main element of the product development process, software engineering efforts to mature more effective methods which will ensure the product's quality are maintained. Thus, the process of testing automation has become more necessary in the global market. Despite this, it has been observed that many Ecuadorian industries still use manual methods to carry out their tests, either because of the complexity of the automation process or the costs involved that apparently do not represent a suitable investment. Therefore, this research proposed to carry out a case study where functional and regression tests were implemented extending the free tool Selenium with the product of the Ecuadorian company GESTORINC S.A. After that, the study established an analysis of the benefits of the automation of tests in terms of how much the software quality attributes were improved previous and post automation. The results of the research are favorable pointing out a medium level of complexity in the process of tests automation, as well as demonstrating in the cost-benefit analysis high percentages of improvement in the effectiveness to make tests and in the software's capacity to be modified.

Key words:

- **FUNCTIONAL TESTING**
- **REGRESSION TESTING**
- **TESTING AUTOMATION**
- **SELENIUM**

CAPÍTULO I

DEFINICIÓN DEL PROYECTO

1.1. Antecedentes

La empresa GESTORINC S.A lleva brindando, durante 19 años, soluciones software que se adapten a las necesidades de sus clientes. Sus productos y servicios se centran principalmente en el desarrollo de aplicaciones web 2.0 para el área de Finanzas. Actualmente, el sistema GESTOR G5 TRUST es uno de sus productos con mayor demanda.

Al igual que empresas como Corporación Kruger o COBIS que desarrollan software en Ecuador, GESTORINC S.A. busca calidad en los productos que ofrece a sus clientes, razón por la cual desarrolla técnicas de “testing”, que le permiten evaluar la calidad del software desarrollado y minimizar futuros costos de mantenimiento.

Actualmente, GESTORINC S.A. realiza pruebas de caja negra y de caja blanca sobre el software que desarrolla con el objetivo de obtener una vista objetiva e independiente del estado del producto que se está testeando, pudiendo evaluarse varias características de calidad. Dentro de las pruebas de caja negra, se establecen pruebas funcionales para controlar que el sistema desarrollado se desempeña de acuerdo a las especificaciones funcionales y requisitos del cliente.

Desde hace cerca de un año la empresa GESTORINC S.A. ha investigado la herramienta Selenium, la cual se utiliza para la automatización de pruebas funcionales,

como alternativa para el proceso manual de “testing”, pero sin embargo no ha realizado la integración de la herramienta en su proceso de “testing”.

1.2. Planteamiento del problema

Desde su nacimiento, las aplicaciones web han venido evolucionando constantemente y adaptándose a las necesidades de diversas industrias e individuos particulares, tanto así, que la mayoría de aplicaciones que se desarrollan hoy en día son orientadas a la web (Forbes, 2017). Dado la necesidad y confianza que en la actualidad muchos negocios y personas ponen en el software, todo proceso de desarrollo de software involucra esfuerzos por el aseguramiento de la calidad (Lewis, 2014). Dentro de las actividades de aseguramiento de la calidad, las pruebas cumplen un rol fundamental, brindando la capacidad de evaluar el nivel de calidad de un producto software. (Graham, 2007). Siendo así, esta es una actividad indispensable y que ha tratado de optimizarse por mucho tiempo en la industria. En especial, después de la aceptación universal de metodologías de desarrollo ágil, fue ineludible buscar un método que permitiera hacer pruebas a la misma velocidad. Por ello, la automatización del proceso de pruebas se ha vuelto más que necesaria en el mercado global; muchas organizaciones ven la automatización de pruebas de software como una solución para reducir los costos de prueba y minimizar el tiempo del ciclo del desarrollo de software (Garousi & Mäntylä, 2016).

A pesar de ello, se ha observado que muchas industrias ecuatorianas aún utilizan métodos manuales para llevar a cabo sus pruebas, ya sea esto por la complejidad del proceso de automatización o porque no creen que los costos asociados al cambio

representan una inversión conveniente en su caso. Para esta investigación en particular se ha decidido estudiar el caso de GESTORINC S.A.

La empresa ha deducido que durante la ejecución de sus proyectos se han dado errores durante el ciclo de sus proyectos, como el no usar métodos y técnicas apropiadas para “testing” dado que entre su personal no cuentan con expertos en pruebas y ciertos miembros del área de desarrollo no poseen los conocimientos suficientes para realizar pruebas adecuadas, sino que simplemente realizan pruebas manuales después de cada modificación. Así también, en varias ocasiones, se ha tratado de usar herramientas para “testing”, que sin embargo, presentan configuraciones genéricas que no se adaptan a todas las necesidades de la empresa y pueden en ciertos casos resultar económicamente costosas.

Todos estos errores han generado consecuencias negativas, como son costos extra en el mantenimiento de software, retrasos en los tiempos de entrega, productos finales de baja calidad que no tardan en presentar problemas, insatisfacción del cliente y en varios proyectos pérdidas económicas.

1.3. Objetivos

1.3.1. Objetivo General

Implementar un software para la ejecución de pruebas funcionales y de regresión en aplicaciones web extendiendo la herramienta Selenium utilizando como caso de estudio el producto GESTOR G5 TRUST, con la finalidad de mejorar los atributos de calidad, específicamente la característica de mantenibilidad.

1.3.2. Objetivos Específicos

- 1) Desarrollar una herramienta que permita implementar pruebas funcionales sobre aplicaciones web.
- 2) Crear un traductor de pruebas funcionales a pruebas de regresión.
- 3) Realizar un análisis de la relación existente entre los atributos de mantenibilidad del software y la automatización de pruebas, mediante un estudio del desempeño del sistema sin y con el uso de la herramienta de automatización.

1.4. Justificación

Por todo lo descrito en la sección de antecedentes y de planteamiento del problema se entiende la necesidad urgente de desarrollar una aplicación orientada a realizar pruebas funcionales y de regresión, las cuales permitirán consecuentemente probar, en conjunto, distintas funcionalidades de una aplicación web 2.0, para verificar que el software cumpla con los requisitos funcionales especificados. El proyecto propuesto facilitaría la tarea de evaluación de calidad del software, de manera que se minimicen el número de ocasiones en las que el producto no cumple con las especificaciones levantadas en la fase de análisis. Los beneficios que aportaría a la empresa incluyen: reducción de los costos de desarrollo y mantenimiento, mejora de la estabilidad del producto, creación de valor para la empresa y finalmente un incremento en la satisfacción de los clientes y la imagen que mantienen de la empresa.

1.5. Alcance

El proyecto de investigación plantea implementar un software que permita exclusivamente ejecutar pruebas funcionales y de regresión.

El software constará de los siguientes componentes:

- Traductor de Casos de Prueba, que permitirá traducir casos de prueba diseñados al lenguaje específico de una aplicación web 2.0.
- Ejecución de casos de prueba traducidos, que se encargará de automatizar la ejecución de los casos de prueba traducidos.
- Reportes de ejecución, que permitirá visualizar los resultados generados por la ejecución de los casos de prueba traducidos.

El software GESTOR G5 TRUST maneja varios módulos, entre los que se encuentran “Contabilidad”, “Auditoría”, “Inversiones”, “Procesos legales”, “Producto” y demás. El software a desarrollarse, y de igual forma las pruebas que se generen, serán asignadas únicamente al módulo de “Personas” del sistema GESTOR G5 TRUST.

En lo que respecta al análisis de la relación que mantienen los atributos de mantenibilidad del sistema versus la automatización del proceso de pruebas se ha decidido realizar un análisis cuantitativo del rendimiento del sistema en dos estados, uno antes de la implementación del software desarrollado y la segunda después de utilizar lo propuesto en esta investigación. El análisis está dirigido específicamente a la mantenibilidad del producto software

1.6. Selección de la metodología de investigación

El presente proyecto se enmarca en el método deductivo, el cual consiste en partir de principios generales o enunciados de carácter universal, y mediante el uso de instrumentos científicos, inferir enunciados de carácter particular.

De acuerdo a Carvajal “el método deductivo de investigación deberá ser entendido como un método de investigación que utiliza la deducción o sea el encadenamiento lógico de proposiciones para llegar a una conclusión o, en este caso, un descubrimiento” (Carvajal, 2013). Mediante el método deductivo de investigación se identificarán todo el conjunto axiomático de partida, que para el caso de esta investigación, serán todos los enunciados referentes a pruebas de caja negra, el proceso de pruebas y herramientas para la automatización de pruebas; a partir de este conjunto axiomático se deberá realizar un proceso de deducción lógica, mediante el cual se obtendrá una conclusión o un enunciado de carácter particular.

Adicionalmente de acuerdo a Gómez la “deducción, tanto si es axiomática como matemática, puede emplearse de manera que facilite el análisis estadístico y el contraste” (Gómez López, 2004). Por lo que para el caso de esta investigación se podrán utilizar datos estadísticos sobre los recursos invertidos y la cantidad de errores encontrados en procesos de pruebas, para llegar a una conclusión de carácter particular.

Esta investigación es de tipo cuantitativa ya que de acuerdo a Wohlin (Wohlin, Höst, & Henningsson, 2006) “se ocupa principalmente de cuantificar una relación o de comparar dos o más grupos; es decir su objetivo es identificar una relación causa-efecto”. La investigación cuantitativa usualmente se desarrolla al establecer experimentos

controlados o también realizando una recopilación de datos mediante casos de estudio. Las investigaciones cuantitativas son adecuadas cuando se prueba el efecto de alguna manipulación o actividad. Los datos cuantitativos promueven las comparaciones y el análisis estadístico, mismo hecho que se considera una ventaja e incentiva a investigadores a utilizarla sobre una investigación cualitativa que es menos formal. (Fenton & Bieman, 1997).

En general, un estudio empírico requiere de los siguientes pasos principales para poder desarrollar la investigación: Definición, Planificación, Operación, Análisis e Interpretación, Conclusiones y Presentación (Runeson & Höst, 2009).

Las investigaciones cuantitativas son varias, y suelen ser clasificadas de acuerdo al objetivo de la investigación y a la rigurosidad que se requiera aplicar en ella. Entre los más conocidos se tiene a: experimentos formales, casos de estudio, encuestas y análisis *post-mortem*. (Wohlin, Höst, & Henningson, 2006).

Para la presente investigación se ha decidido utilizar un caso de estudio. Un caso de estudio, mejor conocido como *case study* se la suele conocer a veces como “la investigación típica” ya que un caso de estudio normalmente analiza y se basa en un proyecto real y, por lo tanto, la situación es “típica”. (Kitchenham, Pickard, & Pfleeger, 1995). Los casos de estudio se utilizan para supervisar proyectos, y plantean actividades o tareas a llevarse a cabo. Los datos se recopilan para un propósito específico en el transcurso de la investigación. Posteriormente, en base a los datos recopilados, se pueden realizar análisis de los resultados; usualmente este tipo de análisis son estadísticos pero no se limitan exclusivamente a ello. Normalmente, este tipo de

metodología está dirigida a inquirir en un atributo específico o en el establecimiento de relaciones entre diferentes atributos (Runeson & Höst, 2009). Más detalles sobre las actividades dentro de la metodología y como fueron desarrolladas dentro de esta investigación se exponen en el capítulo 3.

CAPÍTULO II

ESTADO DE LA CUESTIÓN

2.1. Gestión de calidad en la ingeniería de software

El aseguramiento de la calidad del software es un proceso fundamental dentro de las actividades de la ingeniería de software (Bourque & Fairley, 2014). De acuerdo a varios autores, se define a la calidad del software como la capacidad que posee un producto de software para satisfacer las necesidades, deseos y expectativas, declaradas e implícitas del usuario, cliente o interesado bajo condiciones específicas (ISO/IEC 25010:2011, 2011) (IEEE P730™/D8, 2012). Partiendo de la premisa anterior, el término calidad se enfoca exclusivamente en el grado de conformidad con los requisitos establecidos, sin hacer un énfasis particular en el tipo de requisito que debe ser cumplido (por ejemplo, funcional, de rendimiento, confiabilidad o cualquier otra característica de calidad). Por tanto, se puede concluir que la calidad depende de los requisitos, y consecuentemente cualquier tipo de tarea para medir y/o mejorar la calidad de un producto de software debe estar relacionada directamente a la satisfacción del interesado. La administración de estas tareas conforma una pieza esencial en la rama de la Ingeniería de Software y se conoce como gestión de la calidad.

La gestión de la calidad del software o GCS (del inglés “Software Quality Managment”) es la recopilación de todos los procesos y actividades que certifican que los productos, los servicios y las implementaciones de procesos del ciclo de vida del software cumplan con los objetivos de calidad del software de la organización y logren la satisfacción de los interesados (ISO 9000:2005, 2005).

La GCS comprende cuatro subcategorías: (1) la planificación de la calidad de software, (2) el aseguramiento de la calidad del software o ACS (del inglés “Software Quality Assurance”), (3) el control de calidad de software y (4) la mejora del proceso de software (Bourque & Fairley, 2014). Aunque algunos autores optan por incluir a la cuarta categoría de las otras tres primeras, cada vez más organizaciones establecen a la “mejora del proceso” en una categoría separada que puede abarcar muchos proyectos (Bourque & Fairley, 2014). Cada una de estas cuatro fases forma parte de un proceso cíclico que ayuda a que los requisitos esperados del sistema se cumplan progresivamente (ver figura 1).

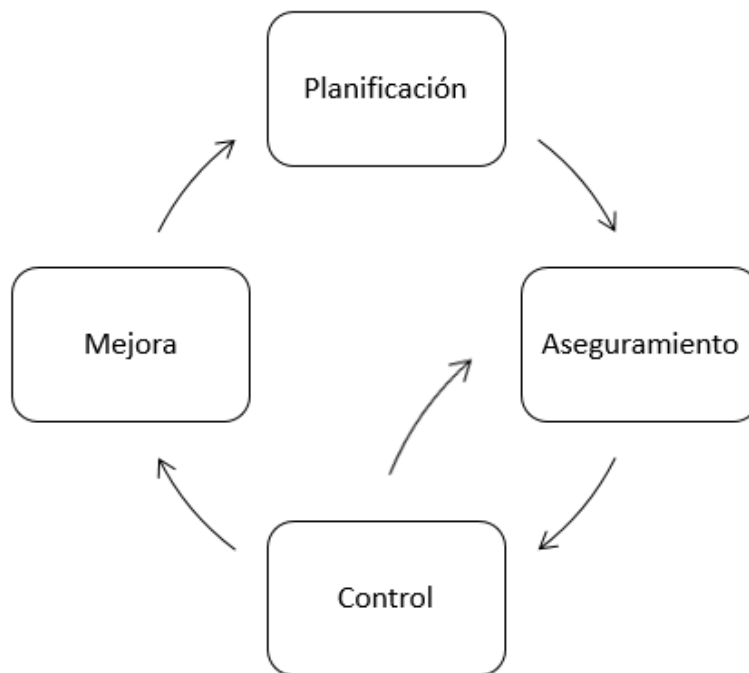


Figura 1 Proceso de Gestión de la Calidad.

Fuente: (Bourque & Fairley, 2014)

Adaptado por los autores.

La primera fase de planificación de la calidad del software incluye: determinar qué estándares de calidad van a ser usados, definir los objetivos específicos de calidad, estimar el esfuerzo y elaborar un calendario de las actividades del proceso de aseguramiento de calidad del software. En algunos casos, la planificación también incluye la definición de los procesos de calidad del software que se utilizarán (Bourque & Fairley, 2014). En base a esta planificación inicial se establecen las actividades de ACS.

Las tareas de ACS (que son la segunda fase) tienen como objetivo definir y evaluar la adecuación de los procesos de software para proporcionar evidencia suficiente, que demuestre confianza de que los procesos de software son apropiados y generan productos de software de calidad adecuada para los fines requeridos (IEEE P730™/D8, 2012).

En paralelo a estas tareas de aseguramiento de la calidad, se realiza el control de calidad. La fase tres se encarga de examinar artefactos específicos del proyecto, tales como documentos y ejecutables, para determinar si cumplen con los estándares establecidos por el proyecto (incluidos los requisitos, limitaciones, diseños, contratos y planes). El control de calidad de software evalúa tanto los productos intermedios como los productos finales (Bourque & Fairley, 2014).

Finalmente, la mejora del proceso de software busca optimizar la efectividad del proceso, la eficiencia y otras características con el objetivo final de incrementar la calidad del software. Esta categoría se conoce con varios nombres en la industria, tales como mejora de la calidad del software o acción correctiva y preventiva del software (Bourque & Fairley, 2014).

2.2. Aseguramiento de la calidad

El “conjunto de actividades sistemáticas que se desarrollan para proporcionar evidencia de la idoneidad para el uso del producto software en su totalidad” (ISO/IEC 25010:2011, 2011) se conoce como aseguramiento de calidad o ACS. El ACS conlleva la realización de actividades diseñadas para controlar la calidad, de forma que se pueda garantizar la integridad del software y prolongar su vida útil.

A pesar de que se espera que los ingenieros de software compartan un compromiso grupal implícito por la calidad del software como parte de su cultura (Bourque & Fairley, 2014), el aseguramiento de la calidad desempeña una función catalizadora que debe alentar a actitudes de calidad y disciplina por parte de la gerencia y los trabajadores de una organización (Lewis, 2014).

El ACS tiene dos aspectos: el aseguramiento del producto y el aseguramiento del proceso. El aseguramiento del producto se enfoca en evaluar que el software final proporcione satisfacción al interesado respecto de sus requerimientos. Por otro lado, el aseguramiento del proceso establece actividades que permitan que el producto software sea de calidad. En conjunto, el ACS instaure varias actividades a lo largo de la vida del software para obtener el mayor grado de calidad que pueda adquirir el producto.

Al considerar los diversos argumentos de la importancia de implementar procesos de ACS, ninguno es más convincente que el de costos asociados para corregir errores (Tian, 2015). El costo de encontrar y corregir defectos aumenta considerablemente a lo

largo del ciclo de vida (Graham, 2007), es decir cuanto más tiempo pase desapercibido un error, más costoso será repararlo.

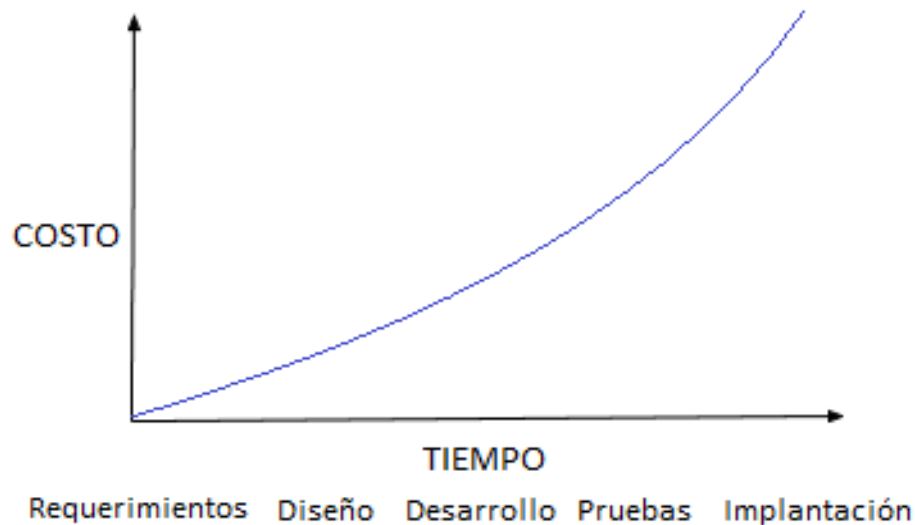


Figura 2 Relación del costo vs el tiempo de corregir defectos.
Fuente: (Graham, 2007)
Adaptado por los autores.

Cómo se puede observar en la curva de costo vs. tiempo de la figura 2 la función es casi exponencial; en otras palabras, mientras más avanzada sea la etapa en la que se encuentra el proyecto de software el costo de reparar defectos aumenta; así durante la etapa de diseño, la corrección de errores representa un costo bajo, mientras que si se tuviera que solucionar dicho defecto en la etapa de implantación conllevaría un esfuerzo mucho mayor.

Uno de los métodos más utilizados dentro del ACS son las pruebas. Estas ayudan en la tarea de medir la calidad del software de acuerdo a tres parámetros: el número de defectos hallados, la cantidad de pruebas que se ejecutaron y el porcentaje de cobertura

del sistema. Estas pruebas pueden ser diseñadas para evaluaciones de los atributos y características funcionales y no funcionales del software (Lewis, 2014).

Se debe reconocer que el software aún contiene fallas, incluso después de la finalización de periodos extensos de prueba. Las fallas del software experimentadas después de la entrega del producto se manejan independientemente como parte del proceso de mantenimiento correctivo (Bourque & Fairley, 2014); también es importante notar que la definición de calidad es relativa al punto de vista del interesado que realiza la evaluación.

2.3. Pruebas

De acuerdo a (Lewis, 2014) “Las pruebas de software son el proceso de evaluación de un elemento de software para detectar diferencias entre la entrada determinada y la salida esperada”; así mismo, estas se utilizan para evaluar la posesión o no de una característica en un elemento de software (Sommerville, 2011). En otras palabras, las pruebas establecen un proceso de verificación y validación del software. La verificación garantiza que el producto cumple con las condiciones declaradas al inicio de la fase de desarrollo, y luego los requisitos especificados al final de la fase de desarrollo son comprobados con el proceso de validación.

El proceso de pruebas está conformado por cinco fases (Lewis, 2014):

- (1) Planificación y control
- (2) Análisis y diseño

(3) Implementación y ejecución

(4) Evaluación y reporte

(5) Actividades de cierre.

La primera etapa “planificación y control” busca entender las metas y objetivos de las pruebas para consecuentemente orientar un plan de acuerdo a la estrategia de prueba; para esto se definen un plan y actividades de control (Lewis, 2014).

El plan de pruebas debe determinar el alcance y los riesgos, establecer el enfoque de la prueba (lo que incluye técnicas, elementos de prueba, el software de prueba, componentes de cobertura, actividades para identificar e interactuar con los usuarios que participan en las pruebas) y los recursos de pruebas (como personas, entorno de prueba, entre otros) (Lewis, 2014).

Además, el plan de pruebas debe calendarizar tareas de análisis y diseño de pruebas, implementación de pruebas, ejecución de pruebas, evaluación de resultados de prueba y determinar los criterios de salida. Las actividades de control de prueba son tareas que regularizan y miden el progreso en relación al plan; estas son: medir y analizar los resultados de las revisiones y pruebas, monitorear y documentar el progreso, informar e iniciar si es necesario acciones correctivas (IEEE Std. 12207-2008, 2008).

En la fase “análisis y diseño” se definen los objetivos generales de prueba identificados durante la planificación y así como también se construyen los diseños y los procedimientos de prueba (Lewis, 2014).

A continuación, la fase de “implementación y ejecución” se enfoca en desarrollar y priorizar los casos de prueba y agruparlos en suites de pruebas para una ejecución de pruebas eficiente; así mismo en esta fase se debe realizar un registro de los resultados de la ejecución de pruebas (Lewis, 2014).

La cuarta fase, evalúa criterios de salida estableciendo una comparación de los resultados reales con los resultados esperados, decide si se necesitan más pruebas o si se deben cambiar los criterios de salida especificados y redacta un informe con el resumen del proceso de prueba para las partes interesadas. Finalmente, las actividades de cierre de pruebas: (1) terminan y archivan el software de pruebas (tales como secuencias de comandos, entorno de prueba y cualquier otra infraestructura de prueba) para su posterior reutilización, y (2) evalúan cómo fue la prueba y analizan las lecciones aprendidas para futuros proyectos (Lewis, 2014).

Las pruebas de software generalmente se realizan en diferentes niveles. Los niveles se pueden distinguir en función del objeto de prueba o el objetivo de prueba (Graham, 2007). El objeto de la prueba varía de acuerdo a la perspectiva desde la que se evalúa, siendo así posible tres tipos: un único módulo, una combinación de varios módulos (relacionados entre ellos por finalidad, uso, comportamiento o estructura) o un sistema completo (Sommerville, 2011). De acuerdo a la división de los tipos de objetos de prueba, se pueden distinguir tres etapas de pruebas: de unidad, de integración y de sistema (Graham, 2007).

Las pruebas unitarias o de unidad verifican el funcionamiento en forma aislada de los elementos de software que se pueden probar por separado. Dependiendo del

contexto, estos podrían ser los subprogramas individuales o un componente más grande hecho de unidades altamente cohesivas. Normalmente, las pruebas unitarias se llevan a cabo accediendo directamente al código que se prueba y se suele contar con el soporte de herramientas de depuración. Aunque no siempre, es típico que los programadores que escribieron el código son quienes realicen las pruebas unitarias (Bourque & Fairley, 2014).

Las pruebas de integración verifican las interacciones entre los componentes de software. Sistemas estructurados jerárquicamente suelen utilizar estrategias clásicas de pruebas de integración, tales como “top-down” y “bottom-up” (Kan, 2002). Las estrategias de integración suelen establecerse con base a la arquitectura del producto, de manera que el desarrollo sea sistemático y los componentes y subsistemas del software se vayan integrando progresivamente y relacionándose de acuerdo a sus funciones. A menudo, las pruebas de integración representan una actividad continua en cada etapa del desarrollo del producto, durante la cual los ingenieros de software se enfocan en abstraer las perspectivas del nivel en el que se están integrando y se olvidan las perspectivas de niveles inferiores (Lewis, 2014). Para software que se considera complejo o de un gran volumen, generalmente se optan por estrategias de prueba de integración incremental para unir todos los componentes a la vez, este tipo de prueba se la conoce como prueba de "Big Bang" (Bourque & Fairley, 2014).

Las pruebas del sistema se refieren a probar el comportamiento de un sistema completo que actúa como una sola entidad. Mientras que las pruebas de unidad e integración deben identificar defectos del software, las pruebas del sistema se suelen

emplear para evaluar los requisitos no funcionales del sistema, como seguridad, velocidad, precisión y confiabilidad; las interfaces externas a otras aplicaciones, utilidades, dispositivos de hardware o entornos operativos también suelen evaluarse a este nivel (Bourque & Fairley, 2014).

En la segunda categoría de pruebas, basadas en el objetivo o meta de la prueba; la evaluación se lleva a cabo en función de los objetivos específicos. Para esto es necesario establecer los objetivos de la prueba en términos precisos y cuantitativos que faciliten la medición y el control del proceso de prueba (Bourque & Fairley, 2014).

Las pruebas pueden estar dirigidas a verificar diferentes propiedades. Los casos de prueba, se pueden diseñar para verificar que las especificaciones funcionales se hayan implementado correctamente (como las pruebas funcionales) y otras propiedades no funcionales (como confiabilidad o seguridad) (Dustin, 2002).

A pesar de que la división y definición de este tipo de pruebas difiere ampliamente de autor a autor, entre las pruebas por objetivos más comunes se encuentran (Bourque & Fairley, 2014) (Sommerville, 2011):

- Pruebas de aceptación o calificación: determinan si un sistema satisface los criterios de aceptación, esto usualmente se lleva a cabo al verificar los comportamientos del sistema deseado versus los requisitos expresados del cliente. Aquí la participación del cliente es activa ya que especifica las actividades que se deben verificar.

- Pruebas de instalación: comprenden procedimientos de instalación y condiciones del ambiente durante y después de la instalación.
- Pruebas alfa y beta: aquí el software se entrega a un grupo seleccionado de usuarios potenciales para su uso, previo al lanzamiento del producto. Estos usuarios informan de problemas encontrados durante su experiencia con el producto.
- Pruebas de logro y evaluación de confiabilidad: este tipo de pruebas permiten establecer medidas estadísticas del nivel de confiabilidad que ofrece el software, usualmente generando de manera aleatoria casos de pruebas de acuerdo con el perfil operativo del software. Además de ello, este tipo de pruebas permiten identificar y corregir fallas para mejorar el nivel de confiabilidad.
- Pruebas de regresión: de acuerdo con (ISO/IEC/IEEE 24765:2010, 2010) este es un “tipo de prueba selectiva de un sistema o componente que permite verificar que las nuevas modificaciones a un software no hayan causado efectos involuntarios, y que el sistema o componente todavía cumpla con los requisitos especificados”. En la práctica, el enfoque de este tipo de pruebas es demostrar que el software todavía cumple con casos de prueba especificados previamente en un conjunto de pruebas. Este tipo de pruebas son utilizadas generalmente en el desarrollo incremental, y su propósito es corroborar que el comportamiento del software no se ha modificado mediante cambios

incrementales en el software, excepto en la medida en que debería (Sommerville, 2011), por ejemplo en el caso de la corrección de una falla.

- Pruebas de rendimiento: verifican que el software cumple con los requisitos de rendimiento especificados y evalúan las características de rendimiento, por ejemplo, la capacidad y el tiempo de respuesta.
- Pruebas de seguridad: se centran en la verificación de que el software esté protegido contra ataques externos; confirma características de confidencialidad, integridad y disponibilidad de los sistemas y sus datos.
- Pruebas de carga: este tipo de pruebas someten al software a su carga de diseño máxima, o más si es posible, para verificar los límites de comportamiento y probar los mecanismos de defensa en sistemas críticos. Este tipo de pruebas también se conocen como pruebas de “estrés”.
- Pruebas “Back-to-Back”: son aquellas en las que dos o más variantes de un programa se ejecutan con las mismas entradas, las salidas se comparan y los errores se analizan en caso de discrepancias.
- Pruebas de recuperación: confrontan las capacidades de reinicio del software después de un bloqueo o falla total del sistema.
- Pruebas de interfaz: identifican si los componentes se han conectado correctamente, proporcionando el intercambio correcto de datos e información de control. Un objetivo específico de la prueba de interfaz es simular el uso de una API por parte de las aplicaciones del usuario final.

- Pruebas de configuración: evalúa el comportamiento del software bajo diferentes configuraciones especificadas para diferentes usuarios.
- Prueba de interacción o usabilidad humano-computadora: evalúa qué tan fácil es para los usuarios finales aprender y usar el software. También se conocen como pruebas de HCI (Human Computer Interaction).

Uno de los objetivos principales de una prueba es detectar tantas fallas en el software como le sea posible, independiente del tipo de prueba o la técnica que se utilice (Bourque & Fairley, 2014). La selección de la técnica de prueba varía de acuerdo al objeto de prueba y la meta que se espera de la prueba. Existen dos categorías de técnicas de pruebas, dinámicas o estáticas, mismas que se subdividen en otras categorías (ver figura 3 y figura 4).

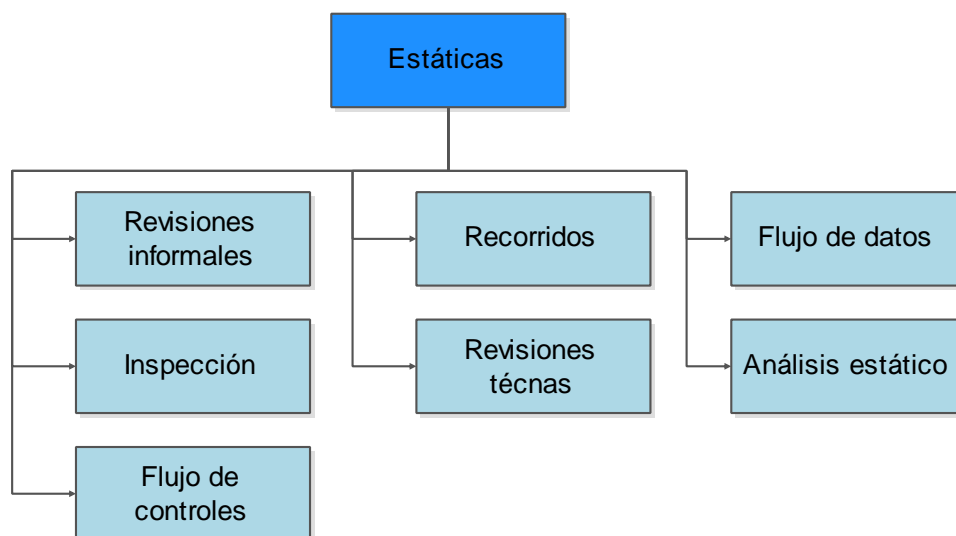


Figura 3 Clasificación de técnicas estáticas.

Fuente: (Bourque & Fairley, 2014)

Adaptado por los autores.

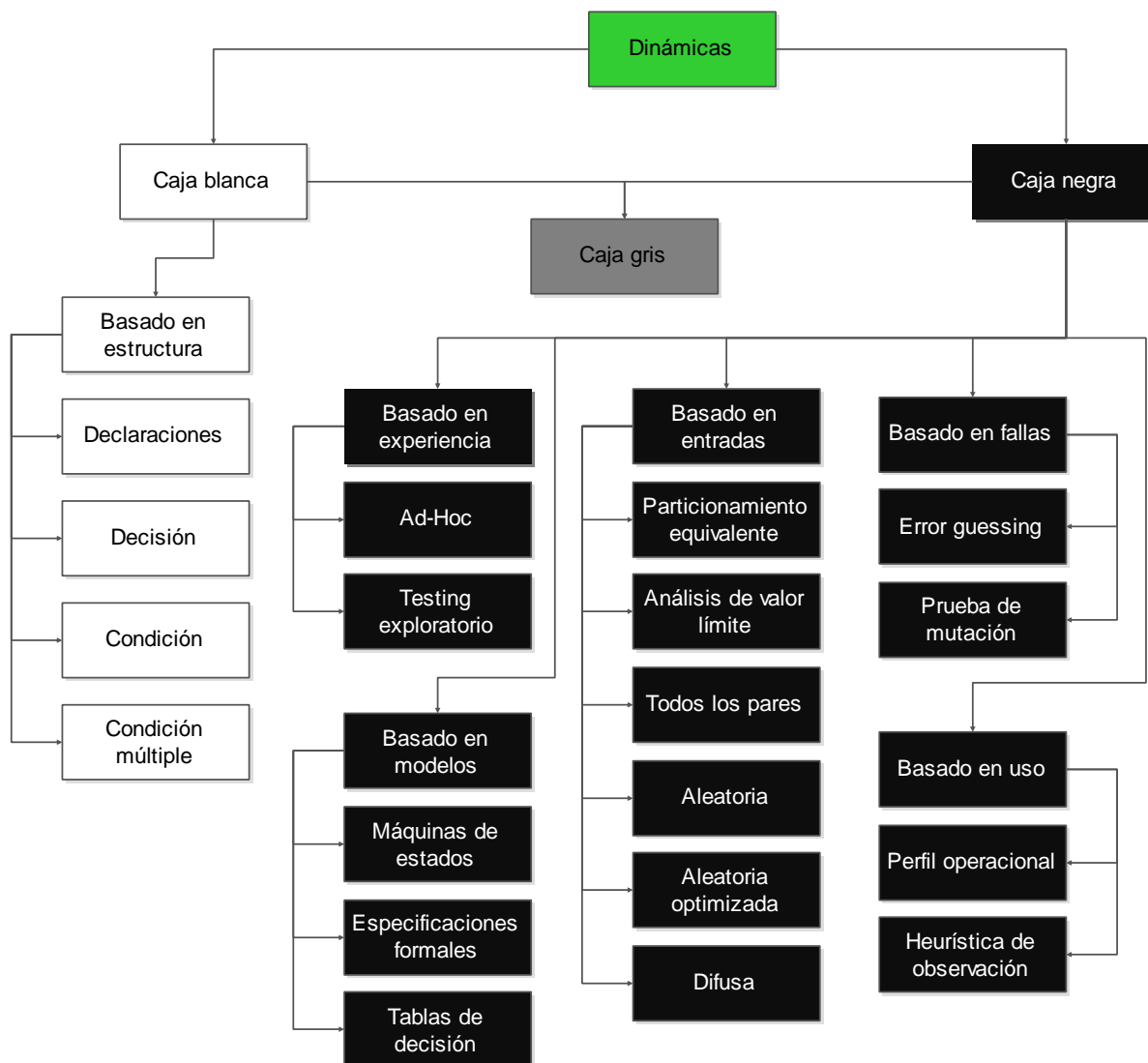


Figura 4 Clasificación de técnicas dinámicas.

Fuente: (Bourque & Fairley, 2014)

Adaptado por los autores.

2.3.1. Clasificación de técnicas de pruebas

Las técnicas estáticas son aquellas que están basadas en el código (Bourque & Fairley, 2014). Estas pueden ser categorizadas de acuerdo al criterio que utilizan:

- (1) Criterios de control de flujo: Estas pruebas apuntan a cubrir todas las declaraciones en un programa; la más intensa de ellas es la prueba de ruta, que tiene como objetivo ejecutar todas las rutas de flujo de control de entrada a salida en un gráfico de flujo de control del programa. Como las pruebas de ruta exhaustivas generalmente no son factibles debido a bucles existentes del software, se suelen emplear otros criterios menos estrictos, estos se centran en la cobertura de rutas que limitan las iteraciones de bucle, como la cobertura de extracto, la cobertura de bifurcación y las pruebas de estado / decisión.
- (2) Criterios basados en flujo de datos: En este tipo de prueba, el gráfico de flujo de control se anota con información sobre cómo se definen, utilizan y eliminan las variables del programa.
- (3) Modelos de referencia para pruebas basadas en códigos: Aunque no es una técnica en sí misma, para este tipo de pruebas la estructura de control de un programa se representa de forma gráfica utilizando un diagrama de flujo. En el diagrama de flujo los nodos y arcos corresponden a elementos del programa.

Las técnicas dinámicas son aquellas que se evalúan cuando el software es ejecutado (Bourque & Fairley, 2014). A veces estas técnicas se clasifican como caja blanca, si las pruebas se basan en información sobre cómo se ha diseñado o codificado el software, o como caja negra si los casos de prueba se basan únicamente en variaciones de comportamiento del software con distintas entradas y/o salidas (Sommerville, 2011).

Las técnicas de caja blanca o caja de cristal, con su nombre lo indica, hacen referencia a aquellas pruebas donde se puede ver en el sistema, ya que requieren el conocimiento de cómo se implementa el software, es decir, cómo funciona. Los datos de prueba se controlan examinando la lógica del programa o sistema, sin preocuparse por los requisitos del programa (Lewis, 2014).

Las técnicas de caja negra o input/output tienen su nombre dado que consideran al software simplemente como una caja negra que posee entradas y salidas, pero no tienen conocimiento de cómo el sistema o componente está estructurado dentro de la caja (Bourque & Fairley, 2014). En esencia, el “tester” se concentra en lo que hace el software, no en cómo lo hace. Es necesario mencionar que este tipo de pruebas abarca tanto pruebas funcionales y no funcionales. Las pruebas funcionales se refieren a lo que hace el sistema, sus características o funciones. Las pruebas no funcionales se refieren a examinar qué tan bien el sistema hace algo, en lugar de qué hace. Los aspectos no funcionales incluyen el rendimiento, la usabilidad, la portabilidad, la capacidad de mantenimiento, etc.

Existe una tercera categoría de pruebas, conocida como pruebas grises, que son un híbrido de pruebas de caja blanca y caja negra, dando varias posibilidades de acuerdo a la necesidad derivada de la meta u objeto de la prueba (Dustin, 2002).

A continuación se detallan algunas de las categorías y subcategorías de técnicas de caja negra.

Técnicas basadas en la intuición y la experiencia del ingeniero de software:

(1) Ad Hoc: Esta probablemente es una de las técnicas más utilizadas en la industria; las pruebas que usan esta técnica se basan en la habilidad, la intuición y la experiencia del ingeniero de software con programas similares.

(2) “Testing” exploratorio: Aquí las pruebas son diseñadas, ejecutadas y modificadas dinámicamente.

Es necesario hacer una nota y mencionar que extensos autores incluyen a estas categorías como pruebas de caja blanca también.

Técnicas basadas en la entrada (Naik & Tripathy, 2008):

(1) Particionamiento equivalente: Se hace un conjunto representativo de pruebas basado en un criterio específico, como las entradas válidas y no válidas que el sistema acepta y procesa y que deberían fluir normalmente o generar un mensaje de error. La técnica de partición de equivalencia requiere que solo se necesite probar una condición de cada partición. Esto se debe a que asumimos que todas las condiciones en una partición serán tratadas de la misma manera por el software. Si una condición en una partición funciona, se asume que todas las condiciones en esa partición funcionarán, y entonces no tiene mucho sentido probar cualquiera de estas otras.

(2) Análisis del valor límite: Aquí los casos de prueba se eligen en o cerca de los límites del dominio de entrada de las variables, con el fundamento subyacente de que muchas fallas tienden a concentrarse cerca de los valores extremos de las entradas.

(3) Todos los pares: En esta técnica los casos de prueba son diseñados para ejecutar todas las combinaciones discretas posibles de cada par de parámetros de entrada. Su objetivo principal es tener un conjunto de casos de prueba que cubra todos los pares.

(4) Prueba aleatoria: Las pruebas se generan puramente al azar. En lo que respecta a procesos de automatización, este tipo de pruebas proporciona un enfoque relativamente sencillo de adaptar.

(5) Pruebas aleatorias optimizadas: Son pruebas mejoradas en los que el muestreo de entrada aleatorio está dirigido por otros criterios de selección de entrada (Chen, Kuo, Merkel, & Tse, 2010).

(6) Pruebas difusas: Usan técnicas de lógica difusa para la selección de casos de prueba.

Técnicas de prueba basadas en modelos. Este tipo de prueba busca validar los requisitos, verificar su coherencia y generar casos de prueba centrados en características de la conducta del software (Bourque & Fairley, 2014). Los componentes clave de las pruebas basadas en modelos son:

- La notación utilizada para representar el modelo o los requisitos del software.
- Modelos de flujo de trabajo o modelos similares.
- La estrategia de prueba o el algoritmo utilizado para la generación de casos de prueba.

Debido a la complejidad que presentan este tipo de técnicas los enfoques de prueba basados en modelos se desarrollan dentro de procesos de automatización de pruebas. Aquí se encuentra tres tipos de pruebas (Utting & Legeard, 2007):

(1) Tablas de decisión: Este tipo de pruebas utiliza tablas de decisión, mismas que representan “relaciones lógicas entre las condiciones (entradas) y las acciones (salidas). Los casos de prueba se derivan sistemáticamente al considerar todas las combinaciones posibles de condiciones y sus correspondientes acciones resultantes” (Bourque & Fairley, 2014).

(2) Máquinas de estados finitos: El programa se modela como una máquina de estados finitos y las pruebas se diseñadas para cubrir distintos estados y transiciones. Esta técnica también se la conoce como técnica de prueba de transición del estado.

(3) Especificaciones formales: los casos de prueba se derivan de la descripción de los requerimientos en un lenguaje formal permite que proporciona un método para verificar los resultados de las pruebas. Un ejemplo de lenguaje que se utiliza con este tipo de técnica es TTCN3 (Testing and Test Control Notation versión 3) (Bourque & Fairley, 2014).

Técnicas basadas en fallas. Los casos de prueba están destinados a revelar categorías de fallas probables o predefinidas. Es común que en esta técnica se diseñe un modelo de falla que clasifica los diferentes tipos de fallas y luego permita dirigir mejor

la generación o selección de casos de prueba, esto en particular para actividades de automatización de pruebas.

(1) “Error guessing”: Los casos de prueba intentan anticipar las fallas más probables del programa; una buena fuente de información para la selección de casos es el historial de fallas descubiertas en proyectos anteriores, así como la experiencia del ingeniero de software. Esta técnica es incluida por muchos autores dentro de la clasificación de técnicas basadas en la experiencia del programador.

(2) Prueba de mutación: Un mutante es una versión ligeramente modificada del programa bajo prueba, que difiere de la versión original por un pequeño cambio sintáctico. Cada caso de prueba se corre tanto en el programa original como en todos los mutantes generados. La hipótesis de éxito de esta técnica supone que es el efecto de acoplamiento; es decir, que al buscar fallas sintácticas simples, se encontrarán fallas más complejas pero reales. Para que la técnica sea efectiva, se debe generar y ejecutar automáticamente una gran cantidad de mutantes de manera sistemática (Jia & Harman, 2010).

Técnicas basadas en el uso. Las pruebas tratan de replicar entornos en los que se desenvolvería el software al ser utilizado, tanto el ambiente como en el usuario:

(1) Perfil operacional: También conocidas como pruebas operacionales, el entorno de prueba reproduce el entorno operativo del software lo más fielmente posible. El objetivo es inferir de los resultados de las pruebas observadas la fiabilidad futura del software cuando se encuentre en uso. Para hacer esto, las entradas se asignan

por probabilidades o perfiles, de acuerdo con su frecuencia de ocurrencia en la operación real (Lyu, 1996).

(2) Heurística de observación del usuario: La heurística de usabilidad se aplica para la observación del uso del sistema en condiciones controladas con el fin de determinar qué tan bien las personas pueden usar el sistema y sus interfaces. Ejemplos de este tipo de técnica son los recorridos cognitivos, análisis de reclamos, observaciones de campo, pensamiento en voz alta, e incluso enfoques indirectos, como cuestionarios de usuarios y entrevistas (Nielsen, 2013).

Dentro del universo del “testing” es importante reconocer la diferencia entre las medidas de una prueba y las técnicas de una prueba. Las medidas proporcionan una evaluación basada en los resultados observados del programa que se encuentra bajo prueba; las técnicas ayudan a garantizar el logro de los objetivos de la prueba (Bourque & Fairley, 2014).

Adicionalmente a la elección de las técnicas y medidas para una prueba, es necesario establecer un criterio de adecuación. La combinación de estos tres permite realizar la evaluación de la prueba desarrollada, misma que determinará si el elemento de software aprobó o no. Los criterios de adecuación de las pruebas requieren que los casos de prueba ejerzan sistemáticamente un conjunto de elementos identificados en el programa o en las especificaciones, por lo que la minuciosidad puede medirse en función de la relación entre los elementos cubiertos (aquellos que si cumplieron con el criterio) y el número total de elementos que estuvieron sujetos a evaluación (Kan, 2002).

Otro tipo de criterio podría ser el de siembra de fallas, donde algunas fallas se introducen artificialmente en un programa antes de la prueba, y dependiendo de cuáles y cuántas fallas artificiales se descubran, se puede evaluar la efectividad de las pruebas y se puede estimar el número restante de fallas genuinas (Bourque & Fairley, 2014). Las posibles interpretaciones en las que se podría evaluar serían: el número de pruebas necesarias para encontrar la primera falla, la relación entre el número de fallas encontradas durante la prueba y todas las fallas encontradas durante y después de la prueba, el porcentaje de mejora de la confiabilidad.

Así también, en el caso de establecer pruebas de mutación, el criterio de adecuación podría basarse en la relación de mutantes muertos con el número total de mutantes generados; esta sería una medida adecuada de la efectividad del conjunto de prueba ejecutado (Bourque & Fairley, 2014).

Finalmente, otro aspecto significativo que cabe discutir sobre el “testing”, sus elementos, técnicas, tipos y procesos, son algunos de los problemas claves que se presentan al momento de planificación y ejecución. Uno de ellos ocurre con los criterios de selección de prueba. Un criterio de selección de prueba es un medio para seleccionar casos de prueba o determinar que un conjunto de casos de prueba es suficiente para un propósito específico. Los criterios de adecuación se pueden usar para decidir cuándo se han realizado o se realizarán suficientes pruebas. Muchas veces así mismo resulta retador reconocer que tipo de prueba es la que se necesita; en estos casos, siempre se debe basar la selección de la técnica en los objetivos que se buscan de la prueba. Solo en base al objetivo se puede guiar la evaluación de la efectividad del “testing” (Naik &

Tripathy, 2008). Un ejemplo de esto sería la relación de una prueba de detección de defectos. En este caso el objetivo de la prueba es hacer que el sistema colapse; aquí la prueba sería exitosa. Sin embargo, un método similar podría utilizarse para demostrar que el software cumple con sus especificaciones u otras propiedades deseadas, en cuyo caso la prueba es exitosa si no se observan fallas en casos de prueba realistas.

Otra limitación es la que se ve entre teoría y la práctica al momento de realizar pruebas. La teoría del “testing” advierte contra la atribución de un nivel de confianza injustificado a una serie de pruebas exitosas. Desafortunadamente, la mayoría de los resultados establecidos de la teoría de las pruebas son negativos, ya que instituyen que las pruebas nunca podrán cubrir en su totalidad al software. La cita de (IEEE P730™/D8, 2012) explica sencillamente este dilema: "las pruebas de programa pueden usarse para mostrar la presencia de errores, pero nunca para mostrar su ausencia". La razón obvia de esto es que las pruebas completas no son factibles en un software realista. Debido a esto, las pruebas deben basarse en el riesgo y se pueden ver como una estrategia de gestión de riesgos.

El problema de las rutas inviables también es común al momento de diseñar pruebas. Las rutas inviables son rutas de control de flujo que no pueden ser ejercidas por ningún dato de entrada. Son un problema importante en las pruebas basadas en ruta, particularmente en la derivación automatizada de las entradas (Bourque & Fairley, 2014).

Dentro de esta materia, también se puede hablar de la probabilidad de éxito del proceso de pruebas. La capacidad de prueba del software tiene dos significados relacionados pero diferentes: por un lado, se refiere a la facilidad con la que se puede

satisfacer un criterio de cobertura de prueba específico; por otro lado, se define como la probabilidad de que un conjunto de casos de prueba exponga una falla si el software es defectuoso (esta se espera se represente estadísticamente) (Naik & Tripathy, 2008). Ya sea en uno u ambos de los criterios, muchas veces resulta complicado elaborar una prueba que tenga una alta probabilidad de éxito.

Finalmente se tiene al problema oracle. Dentro del área del software, un oracle es “cualquier agente, humano o mecánico, que decide si un programa se comportó correctamente en una prueba determinada y, en consecuencia, da como resultado un veredicto de aprobación o error” (Bourque & Fairley, 2014). Usualmente, la automatización de oracles mecanizados puede ser difícil y costosa.

2.4. Aplicaciones web

El desarrollo de aplicaciones web sigue una evolución similar a la observada para los sistemas de software: la producción pasa de una fase artística basada en artesanos altamente calificados a una fase industrial en la que la calidad se controla mediante la introducción de un flujo de trabajo estructurado (Ricca & Tonella, 2001). La rápida difusión de internet y las tecnologías de estándares abiertos está produciendo un crecimiento significativo de la demanda de sitios web y aplicaciones web con requisitos cada vez más estrictos de usabilidad, confiabilidad, interoperabilidad y seguridad. Si bien varias propuestas metodológicas y tecnológicas para desarrollar aplicaciones web se basan tanto en la industria como en el mundo académico, hay una falta general de métodos y

herramientas para llevar a cabo los procesos clave que afectan significativamente la calidad de una aplicación web, como el aseguramiento de la calidad (Di Lucca, Fasolino, Faralli, & De Carlini, 2002).

Las técnicas de prueba tradicionales no son adecuadas para las aplicaciones basadas en la web, ya que pierden sus características adicionales, como su naturaleza de múltiples niveles, la estructura basada en hipervínculos y la función basada en eventos (Mansour & Hourri, 2006). En su investigación del estado del arte del “testing” en aplicaciones web, Di Lucca señala que en los últimos años a pesar de que varios problemas en la área de pruebas de aplicaciones basadas en la web han sido abordados por el trabajos de intelectuales, se debe dedicar más investigación para reducir y evaluar la eficacia de modelos, métodos, técnicas y herramientas para pruebas que combinen los enfoques tradicionales de “testing” con los nuevos existentes (Fasolino & Lucca, 2006).

Del mismo modo que las pruebas de software tradicionales, probar la funcionalidad de una aplicación web debe basarse en los siguientes aspectos básicos: (1) Modelos de prueba, que representan las relaciones entre los elementos o la implementación de un componente de software (Binder, 1999); (2) niveles de prueba, especificando los diferentes ámbitos de las pruebas que se ejecutarán; (3) estrategias de prueba, que heurísticas o algoritmos para crear casos de prueba; (4) procesos de prueba, definiendo el orden de las actividades de prueba, y otras decisiones sobre cuándo se deben iniciar las pruebas, quién debe realizar las pruebas, cuánto esfuerzo se debe usar y problemas similares.

La mayoría de los esfuerzos de la comunidad científica se han centrado en desarrollar modelos representativos de aplicaciones web como ayuda al proceso de “testing”, tales como el trabajo de (Bangio, Ceri, & Fraternali, 2000) (Conallen, 2000) en lenguajes de modelado, modelos de estructura como (Liu, Kung, Hsia, & Hsu, 2000) (Ricca & Tonella, 2001) y modelos de comportamiento como (Di Lucca, Fasolino, Faralli, & De Carlini, 2002) que utiliza tablas de decisión y (Andrews, OVutt, & Alexander, 2005) que implementa máquinas de estado para diseñar casos de prueba para técnicas de “testing” de caja negra.

Algunas de las soluciones privadas que se han utilizado para automatizar pruebas funcionales de aplicaciones web son programas como Mercury Interactive’s WinRunner or Rational Robot. Estas soluciones son costosas y los resultados obtenidos varían de escenario a escenario (Holmes & Kellogg, 2006).

2.5. Selenium

Selenium es un conjunto de diferentes herramientas de software que permiten en conjunto la automatización de pruebas para aplicaciones basadas en la web. Las herramientas de Selenium realizan operaciones altamente flexibles, lo que habilita muchas opciones para localizar elementos de la interfaz de usuario en el navegador y comparar los resultados esperados de las pruebas respecto del comportamiento real de la aplicación. Una de las ventajas que presenta Selenium es la versatilidad del soporte en diferentes entornos, permitiendo así la ejecución de pruebas en múltiples plataformas de navegador. (Selenium.org, 2017)

Originalmente, Selenium fue desarrollado por Jason Huggins en 2004 para un proyecto privado; aquí él desarrolla una librería en JavaScript que se convirtió en el core de Selenium, misma que subyace a todas las funciones de Selenium Remote Control (RC) y Selenium IDE. Posteriormente Huggins se une con Simon Stewart de Google que había desarrollado un WebDriver también para automatización de pruebas. En 2008 ambos proyectos emergen como uno brindando una herramienta más potente (Selenium.org, 2017).

Selenium es un software de código abierto que se distribuye bajo la licencia apache 2.0 por lo que puede ser descargado directamente desde el sitio oficial y ser usado sin costo. Así mismo, la empresa se encuentra desarrollando otros proyectos como Selenium Grid, que permite hacer pruebas de concurrencia múltiple o Flash Selenium para experimentar programas escritos en Adobe Flex o Selenium Silverlight (Selenium.org, 2017).

2.5.1. Componentes de Selenium

Selenium no es una única herramienta, sino que comprende una suite de 4 herramientas de software, donde cada una realiza un rol específico; estas son: Selenium IDE, Selenium RC, Selenium WebDriver, Selenium Grid.

(1) Selenium IDE: es una herramienta de creación de prototipos para construir scripts de prueba. Esta viene como un plugin de Firefox que proporciona una interfaz para desarrollar pruebas automatizadas. Este IDE tiene la función de grabación y exportar, es decir registra las acciones del usuario en el navegador a medida que las realiza y luego las exporta como una secuencia de comandos,

mismos que son reutilizables en varios lenguajes de programación para que posteriormente se pueden ejecutar. Actualmente Selenium provee un API para Java, C#, Ruby y Python (Selenium.org, 2017).

(2) Selenium Remote Control (RC): también se lo conoce como Selenium 1, ya que fue reemplazado por Selenium WebDriver (Selenium 2). Este es un servidor desarrollado en Java que acepta comandos al navegador vía HTTP. Esta herramienta hace posible escribir pruebas automatizadas para aplicaciones web, en cualquier lenguaje de programación lo que resulta en una mejor integración a entornos de prueba ya existentes. Actualmente Selenium RC está obsoleto y no cuenta con soporte (Selenium.org, 2017).

(3) Selenium WebDriver: es la nueva herramienta de automatización de Selenium que incluye nuevas características tales como una API más cohesiva y orientada a objetos, así como también una respuesta a las limitaciones de Selenium 1. Es compatible con WebDriver y ejecuta la interfaz Selenium RC para compatibilidad con versiones anteriores. Selenium WebDriver acepta comandos que se pueden enviar con Selenese o con el API de cliente hacia el navegador de preferencia (Selenium.org, 2017).

Selenese es un lenguaje específico de dominio para escribir pruebas que se pueden ejecutar en un amplio número de lenguajes de programación (en el caso de esta investigación Java). Para desarrollar esto se debe implementar un controlador del navegador específico para al que se envían los comandos y da como respuesta los resultados de la prueba. La mayoría de los controladores del

navegador lanzan y acceden a la aplicación de navegador, tales como Mozilla Firefox o Internet Explorer, pero también existe la posibilidad de usar un simulador de navegador como HtmlUnit (Selenium.org, 2017).

Una gran diferencia y ventaja que posee Selenium WebDriver respecto de su antecesor es que en Selenium 1 el servidor Selenium RC era indispensable; en esta nueva versión la herramienta inicia una instancia del navegador y la controla para la ejecución de las pruebas sin la necesidad de un servidor externo. Para los usuarios que sin embargo, desean la opción de ejecución remota de pruebas, existe Selenium Grid.

Otra diferencia interesante en las versiones 1 y 2 es que Selenium 1 apuntaba a suministrar una interfaz diversa, con varias opciones de operaciones, mientras que Selenium WebDriver busca proveer bloques de construcción básicos, o mejor dicho prototipos, sobre los cuales los desarrolladores puedan escribir en su propio lenguaje específico de dominio las pruebas.

Selenium WebDriver funciona realizando llamadas directas al navegador haciendo uso del soporte nativo de cada navegador para la automatización. Dado la variedad de navegadores web a disposición y varios lenguajes de programación populares, para las desarrollar y correr las pruebas se necesita una especificación común, misma que es proporcionada por la API de WebDriver. Cada navegador tiene que implementar esta API llamada Remote WebDriver o Remote WebDriver Server. En un nivel superior, la arquitectura Selenium WebDriver se parece a lo mostrado en la figura 5.

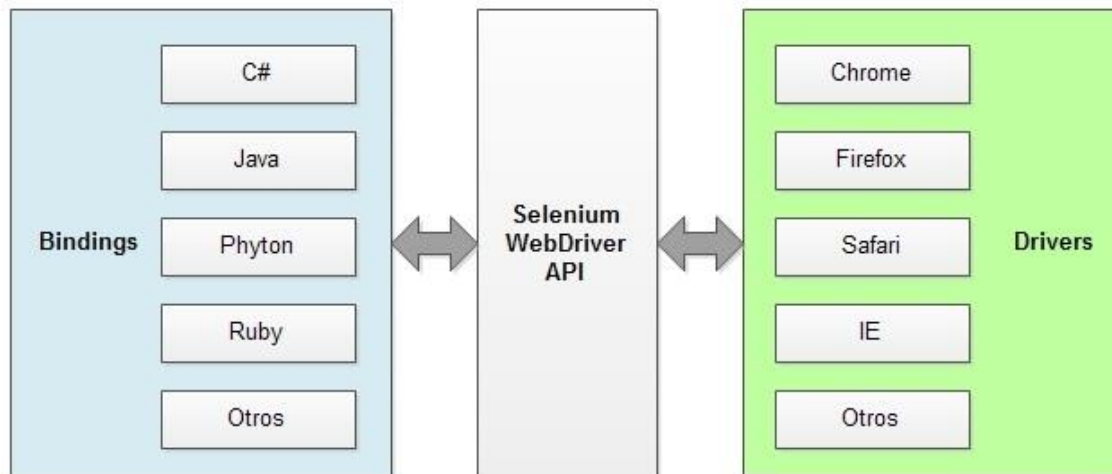


Figura 5 Arquitectura Selenium WebDriver.

Fuente: (Selenium.org, 2017)

Adaptado por los autores.

El proceso de pruebas con Selenium WebDriver explicado superficialmente es el siguiente: el enlace de idioma envía los comandos a través del controlador común que proporciona la API, en el otro extremo existe también un controlador que escuchará estos comandos y los ejecutará en el navegador seleccionado utilizando el WebDriver remoto; finalmente este devolverá el resultado a través de API en el código de enlace. Hay que tomar en cuenta que sea cualesquiera los comandos emitidos en el código, estos se interpretarán en métodos de servicio web, con el protocolo JSON y luego el controlador remoto recibirá la solicitud HTTP, los ejecutará en el navegador y luego devolverá la respuesta (Selenium.org, 2017).

Desde el 2012, Simon Stewart del equipo de Selenium y David Burns de la Fundación Mozilla realizan negociaciones con el World Wide Web Consortium (W3C) para que Selenium WebDriver se convierta en un estándar de Internet,

apuntando de esta forma a que la herramienta sea la implementación de referencia del estándar WebDriver en varios lenguajes de programación.

(4) Selenium Grid: permite escalar Selenium RC para grandes suites de prueba que deben ejecutarse en múltiples entornos, es decir facilita la ejecución de pruebas en paralelo en varias máquinas manejando distintas versiones y configuraciones de manera centralizada. La manera en que funciona es con varios servidores y un servidor concentrador; las pruebas se comunican al concentrador para obtener acceso a una instancia del navegador del pool de instancias; así el concentrador lleva una lista de todas las instancias de los navegadores, es decir cada uno de los nodos de Selenium WebDriver y asigna a las pruebas una de las instancias para su ejecución (Selenium.org, 2017).

2.5.2. Automatización de pruebas para aplicaciones web

Varias, tal vez la mayoría de las aplicaciones de software de hoy se escriben como aplicaciones basadas en la web para ejecutarse en un navegador de internet. La efectividad de probar estas aplicaciones varía ampliamente entre empresas y organizaciones. En una era de procesos de software altamente interactivos y receptivos donde muchas organizaciones están usando algún tipo de metodología ágil, la automatización de pruebas se está convirtiendo frecuentemente en un requisito para los proyectos de software (Selenium.org, 2017). La automatización de pruebas es a menudo la respuesta. Automatización de pruebas significa utilizar una herramienta de software

para ejecutar pruebas repetibles contra la aplicación que se probará. En específico, esto proporciona un medio idóneo para la generación de pruebas de regresión.

La automatización de pruebas tiene ventajas específicas para mejorar la eficiencia a largo plazo de los procesos de prueba de un equipo de software. Dentro de las ventajas de automatizar principalmente se encuentra la capacidad de repetitividad de las pruebas y la velocidad a la que se pueden ejecutar las pruebas. Esto se refleja en pruebas de regresión frecuentes, rápida retroalimentación a los desarrolladores, versiones virtualmente ilimitadas de la ejecución de casos de prueba, soporte para metodologías ágiles, documentación disciplinada de casos de prueba, informes de defectos personalizados y el hallazgo de defectos perdidos por pruebas manuales.

Del conjunto de herramientas comerciales y de código abierto disponibles para ayudar con el desarrollo de la automatización de pruebas, Selenium es posiblemente la más utilizada. En un estudio comparativo de automatización de pruebas entre el “framework” de Selenium y QTP (Jagannatha, Niranjanamurthy, Manushree, & Chaitra, 2014), los resultados expuestos mostraron que en un lapso de tiempo muy corto, el marco de prueba de automatización de Selenium está ganando amplia aceptación y que es la mejor herramienta para las pruebas automatizadas de los sitios web de hoy, y que permite reducir costos de operación respecto del uso de QTP.

2.5.3. Investigaciones de la academia

Uno de los trabajos que más se destaca es el de Holmes y Kellogg (Holmes & Kellogg, 2006) que utilizaron Selenium para pruebas de funcionalidad en cinco proyectos diferentes de aplicaciones web durante aproximadamente un año. El resultado reportado

fue que “todos valoraron las pruebas y el feedback” proporcionado por Selenium y que también “las pruebas fueron artefactos extremadamente valiosos para entregar al cliente al final de las iteraciones”. A pesar de esto, ellos mencionan que el experimento fue realizado en la versión 0.5, por lo que se encontraron con ciertos problemas de estabilidad e integridad, así como también funcionalidades que el software no proveía. Así mismo, la investigación se toma desde el punto de vista de desarrollo ágil y el aporte a los programadores; no hacen menciones cuantitativas de la optimización de tiempo y otros recursos que tuvieron después de la automatización. También, el trabajo habla sobre principalmente de pruebas funcionales y como ser las utilizó para integración continua. No hay mención de otro tipo de pruebas. Como recomendación, ellos establecen que más estudios de la herramienta deben efectuarse.

Un poco más actual, la investigación de Leotta et. al (Leotta, Clerissi, Ricca, & Tonella, 2013) presenta un análisis empírico de costo / beneficio de dos categorías diferentes de enfoques de pruebas funcionales de web automatizadas: (1) pruebas web de captura-reproducción (usando Selenium IDE); y, (2) pruebas web programables (usando Selenium WebDriver). Los resultados muestran la evaluación de los costos de aplicar estos enfoques de prueba tanto al desarrollar las suites de prueba iniciales desde cero como cuando se mantienen las suites de prueba, luego del lanzamiento de una nueva versión de software. Aunque el artículo está orientado en números al mostrar los hallazgos en términos de porcentajes, el enfoque es exclusivamente en pruebas funcionales y asumiendo como premisa que el proceso “testing” siempre fue automático, no desde un enfoque que contraste las pruebas mecánicas de automáticas.

Otra investigación que es oportuna mencionar es la de Castro et. al (Castro, Macedo, Collins, & Dias-Neto, 2013) quienes implementaron la extensión Selenium RC para realizar pruebas en aplicaciones web que requieren verificar datos en bases de datos y realizaron un caso de estudio del análisis del impacto de la herramienta en términos de esfuerzo y tasa de automatización en el desarrollo de un nuevo proyecto de aplicación web. Los resultados sugieren una reducción significativa (92% y 88%) en el esfuerzo por ejecutar pruebas automatizadas en la base de datos en comparación con, respectivamente, la ejecución manual y semi-automatizada. Cabe recalcar que el estudio se basó primariamente es los valores que se recuperaban de la base de datos, más no es pruebas de algún tipo en específico.

El trabajo de Bruns et. al (Bruns, Kornstädt, & Wichmann, 2009) describe la ejecución de pruebas de regresión automática para aplicaciones web que usa el marco de prueba de Selenium. Aunque de gran contribución, el reporte se concentra más en la parte técnica del desarrollo de las pruebas que en establecer una base evaluativa sobre de Selenium.

Similar a este último está la investigación de Wang y Xu (Wang & Xu, 2009). En este artículo se analiza un marco de prueba automático basado en Selenium, FitNesse y DbFit; los autores mencionan que su “framework” podría reducir en gran medida los números de línea del código de prueba y el período de desarrollo del proyecto, disminuir la tasa de error aleatorio, facilitar la tabla de accesorios de escritura, mejorar la productividad de codificación y la calidad del producto final, pero no realizan una evaluación que muestre con evidencia adecuada que dichos beneficios son alcanzables.

Así mismo, en el libro *The Cucumber Book* (Wynne, Hellesoy, & Tooke, 2017) se hace una pequeña referencia a Selenium. Esta es exclusivamente práctica y siempre está acompañada del uso en paralelo del software Capybara para el funcionamiento.

Usando el IDE de Selenium, los investigadores Xu et. Al (Xu, Xu, Bavikati, & Wong, 2012) presentan un enfoque para extraer una especificación de comportamiento de un conjunto de pruebas del IDE de Selenium y generar pruebas abstractas que se sintetizan en una red de Petri de alto nivel que captura restricciones temporales y valores de datos. Esto a su vez se utilizó para pruebas de seguridad, como inyecciones SQL y vulnerabilidades XSS.

Las investigaciones mencionadas anteriormente son las más relevantes en el área de automatización de pruebas utilizando Selenium. Como es observable, no se han realizado extensos estudios sobre esta herramienta, y la mayoría de las publicaciones de la comunidad son técnicas, que van guiando en el proceso de implementación más no en un análisis de costo – beneficio que representa Selenium. Por tal razón, se consideró necesario realizar este proyecto de investigación como caso de estudio de la aplicación de Selenium para la automatización de pruebas funcionales y de regresión en una empresa de Ecuador.

2.6. Estándares utilizados para definir y medir la calidad

Dentro del caso de estudio, se requirió de un modelo de calidad del software así como también de un proceso que permita establecer el nivel de calidad del producto software. Para ello se utilizaron el modelo ISO/IEC 25010 y el proceso descrito por la ISO/IEC 25040.

2.6.1. ISO/IEC 25010

La norma ISO 9126 constituye la base de la que deriva la mayoría de los modelos de calidad de software; sin embargo, el estándar ISO 9126 ha sido reemplazado por ISO 25010 y es el más utilizado en la actualidad como marco de referencia para proyectos de la academia y la industria (Adewumi, Misra, & Omoregbe, 2015).

El modelo propuesto por la ISO 25010 constituye la clave de un sistema para la evaluación de la calidad de un producto software; dicho sistema lo conforman todos los estándares de la serie ISO/IEC 25000. La ISO/IEC 2500, también conocida como SQuaRE, es una norma internacional para la calidad del producto software (ISO/IEC 25010, 2014). La serie 25000 consta de las siguientes divisiones bajo el título Requisitos y Evaluación de Calidad de Software y Sistemas (en inglés Systems and Software Quality Requirements and Evaluation):

- ISO/IEC 2500n - División de Gestión de la Calidad,
- ISO/IEC 2501n - División de Modelo de Calidad,
- ISO/IEC 2502n - División de Medición de la Calidad,
- ISO/IEC 2503n - División de Requisitos de calidad, y
- ISO/IEC 2504n - División de Evaluación de la Calidad.

El modelo de ISO/IEC 25010 describe las características de calidad del software, mismas que se toman en cuenta por las personas que realizan el proceso de gestión de la calidad y valoran las propiedades del producto con la finalidad de establecer cuantitativamente y/o cualitativamente su calidad (ISO/IEC 25010, 2014).

Los elementos del modelo se basan en que la calidad de un sistema se figura en el grado en que este cumple con las necesidades declaradas y no declaradas de las diferentes partes interesadas, consecuentemente proporcionando su valor. Las necesidades particulares de los interesados son las que precisamente se representan en el modelo de calidad en manera de 8 características y sub características, ya sean estas relacionadas con la funcionalidad, el rendimiento, la seguridad, la mantenibilidad, etc.

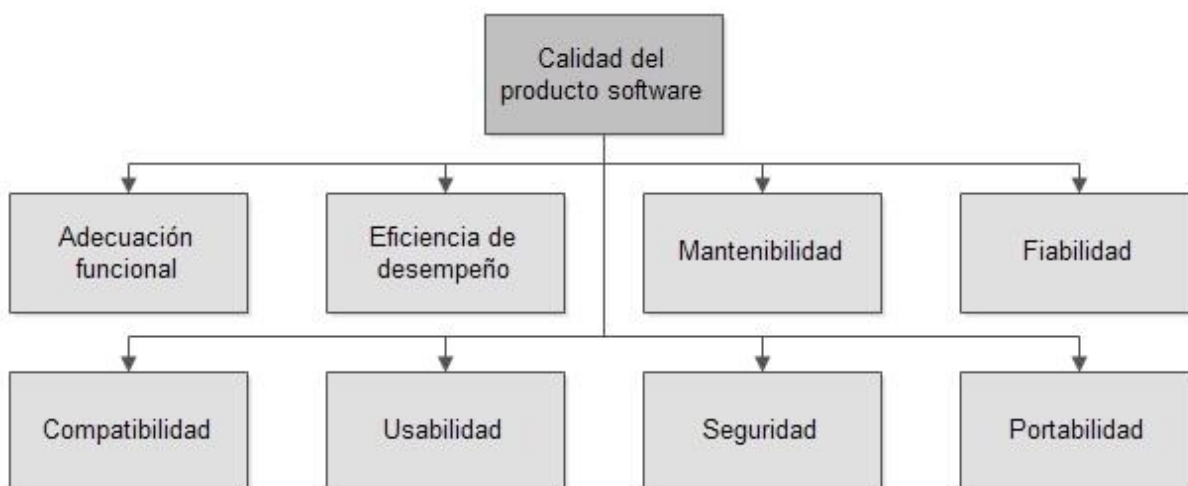


Figura 6 Modelo de calidad del software de acuerdo a la ISO 25010.

Fuente: (ISO/IEC 25010, 2014)

Adaptado por los autores.

El modelo de calidad de la norma ISO/IEC 25010 que se muestra en la Figura 6 identifica 8 características principales de calidad: (1) adecuación funcional, (2) fiabilidad, (3) operabilidad, (4) eficiencia en rendimiento, (5) seguridad, (6) compatibilidad, (7) mantenibilidad y (8) portabilidad (ISO/IEC 25000, 2014). De estas características, la investigación se ha enfocado en la mantenibilidad, y específicamente en la **modificabilidad** y la **testabilidad**.

2.6.2. ISO/IEC 25040

Una vez explicados los atributos de calidad que se pretenden evaluar, es necesario describir el proceso de evaluación, mismo que se basó en el estándar ISO/IEC 25040 de la norma SQuaRE. El estándar ISO/IEC 25040 provee una descripción del proceso que se debe seguir para evaluar la calidad del producto de software y también estipula los requisitos para el desarrollo de dicho proceso. Cómo se puede ver en la figura 7, este proceso se compone de cinco actividades: (1) establecer los requerimientos de evaluación, (2) especificar la evaluación, (3) diseñar la evaluación, (4) ejecutar la evaluación, y (5) concluir la evaluación (ISO/IEC 25040, 2014).

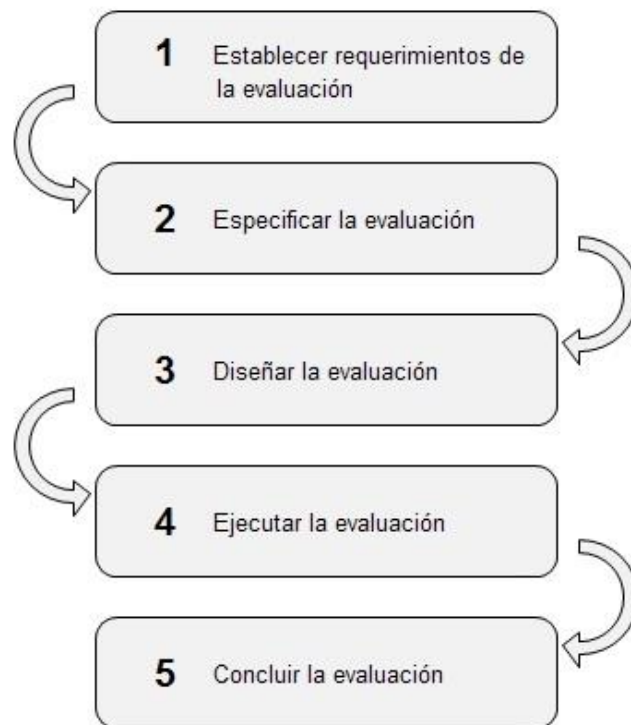


Figura 7 Proceso de evaluación de la calidad.

Fuente: (ISO/IEC 25040, 2014)

Adaptado por los autores.

Las tareas de la actividad de “establecimiento de los requerimientos de evaluación” incluyen: establecer el propósito de la evaluación, obtener los requisitos de calidad del producto de software, identificar las partes del producto que se incluirán en la evaluación y definir la rigurosidad de la evaluación.

La “especificación de la evaluación” involucra las tareas de: seleccionar medidas de calidad (es decir, los módulos de evaluación), definir criterios de decisión para medidas de calidad y definir los criterios de decisión para la evaluación.

Después, el “diseño de la evaluación” consiste en elaborar un plan de cómo se ejecutarán las actividades para evaluar los atributos de calidad seleccionados. La cuarta fase de “ejecución de la evaluación” establece: hacer las mediciones aplicando los criterios de decisión para medidas de calidad y los criterios de decisión para la evaluación.

Finalmente, la “conclusión de la evaluación” incluye: revisión del resultado de la evaluación, redacción de un informe de evaluación, revisar la evaluación de calidad, realizar la disposición de los datos de evaluación y brindar retroalimentación a la organización.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

De acuerdo a lo expuesto en el capítulo 1, esta investigación desarrolla un caso de estudio, puesto que esta es una metodología adecuada para investigaciones en el área de ingeniería de software, donde se estudian los fenómenos contemporáneos en su contexto natural y permite posteriormente reportarlos de una forma apropiada a la comunidad científica de manera que sean entendibles y replicables.

El nivel de control en un caso de estudio es menor que en el de un experimento, puesto que el primero es un estudio observacional mientras que el segundo es un estudio controlado (Zelkowitz & Wallace, 1998); por ejemplo, un caso de estudio puede estar enfocado en construir un modelo para predecir el número de fallas en las pruebas. A pesar de que existen investigaciones de este tipo de estudios que utilizan análisis estadístico multi-variante y métodos de análisis matemático como regresión lineal (Manley, 2016), Runeson menciona que las conclusiones de un caso de estudio “se basan en una clara cadena de evidencia, ya sea cualitativa o cuantitativa, recopilada de múltiples fuentes de forma planificada y consistente” y que por el contrario nunca proveería conclusiones con significancia estadística (Runeson & Höst, 2009).

De acuerdo a Robson, un plan para un caso de estudio debe contener al menos los siguientes elementos (Robson 2002):

- Objetivo: es decir que se desea lograr.

- El caso: ¿cuál es el objeto de estudio?
- Teoría: que permita establecer un marco de referencia.
- Preguntas de investigación: que deben responder a lo que se desea saber.
- Métodos y estrategia de recolección: descripción de cómo y de dónde se van a recolectar los datos.

Dado que se optó por seguir los lineamientos para llevar a cabo y reportar casos de estudio de (Runeson & Höst, 2009), es necesario responder brevemente a estas preguntas para el plan de caso de estudio.

- Objetivo: aplicar la metodología de caso de estudio de tipo explicativo para describir el efecto de la implementación de pruebas automatizadas en los atributos de mantenibilidad de una aplicación web.
- El caso: atributos de mantenibilidad del módulo de PERSONAS del sistema GESTOR G5 TRUST.
- Teoría: varias investigaciones en la academia se han enfocado en la automatización del proceso de pruebas para el software, alegando que existe un incremento del grado de calidad desde varios puntos de vista, por tanto, esta investigación debería obtener resultados similares.
- Preguntas de investigación: ¿el uso de herramientas de automatización de pruebas mejora los atributos de mantenibilidad de una aplicación web?
- Métodos y estrategia de recolección:

- Se desarrolló una herramienta (una aplicación en java que permite la automatización de pruebas funcionales y de regresión).
- Se hicieron mediciones de los niveles de calidad del sistema inicialmente, y también después de utilizar la herramienta desarrollada.

Runeson describe en su investigación (Runeson & Höst, 2009) una plantilla que sirve de base para diseñar y llevar a cabo casos de estudio, esta plantilla mencionada, se va actualizando continuamente. Las partes principales que se deben incluir son:

- General
 - Que proporciona una breve descripción general del proyecto de investigación, así como el método de investigación que utilizará el caso de estudio.
- Procedimientos
 - Que realiza una descripción detallada de los procedimientos para llevar a cabo el caso de estudio, incluyendo detalles prácticos sobre contactos, calendario, presupuestos y demás.
- Instrumentos de investigación
 - Por ejemplo guías de entrevistas, cuestionarios, etc. que se utilizarán para garantizar la recopilación coherente de datos; es decir herramientas de la investigación.
- Análisis de datos

- Que establece una descripción detallada de los procedimientos de análisis de datos, incluyendo esquemas de datos, métodos de análisis, inferencias, etc.
- Reporte
 - Que detalla los resultados obtenidos en el caso de estudio de forma concisa y comprensible.

Siguiendo la guía de dicha plantilla, se realizó el caso de estudio y en este documento se describe la actividad final que es el reporte (ver capítulo 5).

Aunque existen diferentes maneras de reportar un estudio de acuerdo al tipo de público objetivo (Yin, 2003), de acuerdo a Runeson, un reporte de caso de estudio que está dirigido al área académica es de tipo linear – académico (Runeson & Höst, 2009), y debe tener los siguientes elementos:

- Título
- Autor
- Resumen
- Introducción, donde se deben describir:
 - Definición del problema
 - Objetivos de la investigación
 - Contexto
- Diseño del caso de estudio
 - Preguntas de la investigación
 - Selección del caso y sujetos de estudio

- Procedimientos de recolección de datos
- Procedimientos de análisis de datos
- Procedimientos de validación de la información
- Resultados, que engloba:
 - Resultados del caso de estudio
 - Detalles de la ejecución y dificultades encontrados durante el análisis e interpretación.
 - Problemas en la evaluación de validez
- Conclusiones y trabajos futuros, que también incluye:
 - Relaciones a evidencia ya existente
 - Impacto e implicaciones de los resultados
 - Limitaciones
- Reconocimientos
- Referencias
- Anexos

Para la organización del capítulo 5 se ha utilizado la estructura propuesta por Runeson, decidiendo trasladar la parte de conclusiones y trabajos futuros al capítulo 6. Así mismo, los reconocimientos, referencias y anexos se encuentran distribuidos al inicio y final de este documento.

CAPÍTULO IV

SOFTWARE PARA AUTOMATIZACIÓN DE PRUEBAS

El principal instrumento de esta investigación fue la herramienta desarrollada que permite la corrida de las pruebas funcionales y de regresión y la presentación de reportes de los resultados de las pruebas. Así mismo, esta herramienta interactúa con el IDE de Selenium para la ejecución de los casos de pruebas en el navegador web.

4.1. Proyecto Java

Esta herramienta consiste en un proyecto en java compuesto por tres módulos: dbTesting, demonioTesting y feTesting; cada uno de ellos se explica a continuación.

El módulo “dbTesting” contiene todas las clases necesarias para realizar operaciones de consulta, actualización, e inserción de datos sobre la base de datos donde se almacenarán todos los datos relacionados con casos de prueba, suites (puede contener uno o más casos), corridas (serán las denominadas pruebas funcionales o de regresión) y sus respectivos resultados. La estructura de este proyecto se puede ver en la figura 8.



Figura 8 Estructura módulo dbTesting

El módulo “demonioTesting” es una aplicación de consola que busca de forma iterativa pruebas funcionales o de regresión que aún no hayan sido corridas, una vez encontrada una prueba procederá a armar el código java necesario para que corran todas los casos asignados a la prueba. La estructura de este se muestra en la figura 9.

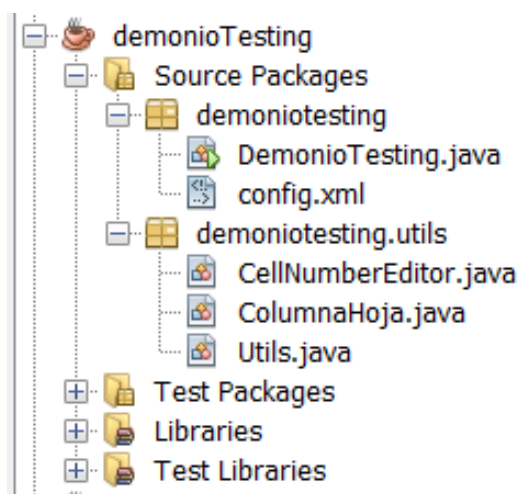


Figura 9 Estructura módulo demonioTesting

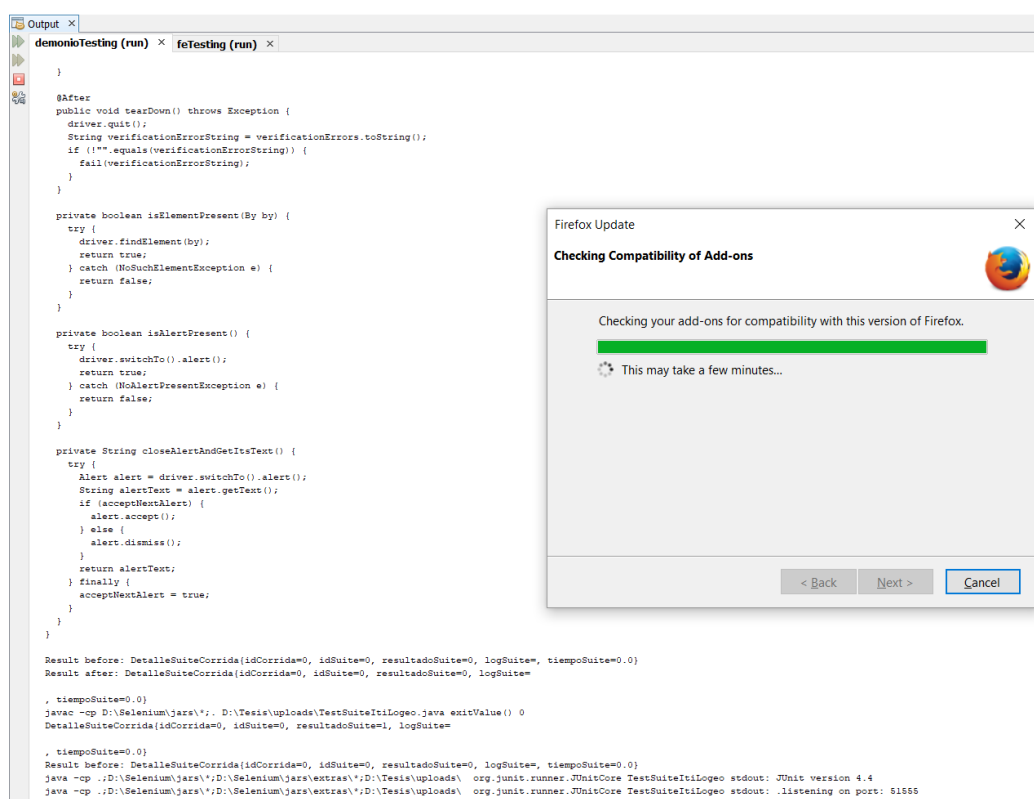
El archivo resultante con el código java de todos los casos es ejecutado como un test de JUnit el cual genera logs de cada caso y de toda la prueba, en los cuales se puede identificar los errores ocurridos en caso de haberlos. La figura 10 muestra una captura de pantalla del módulo en ejecución.

Captura de pantalla de la ejecución del módulo demonioTesting. El texto visible es:

```
Output - demonioTesting (run) x
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
No se han encontrado casos que correr
Corrida Select Not Runned Tests
URL: jdbc:mysql://localhost:3306/dbtesting?zeroDateTimeBehavior=convertToNull
{}
..
```

Figura 10 Módulo demonioTesting en ejecución

Este módulo es capaz de correr de forma asincrónica el número máximo de subprocesos que sea capaz de manejar el procesador de la PC o Servidor donde se esté ejecutando la aplicación. El valor que indica el número de procesos que puede correr de forma asincrónica estará definido en el archivo de configuración del proyecto. En la figura 11 se puede ver al módulo demonioTesting ejecutando una prueba.



The screenshot shows an IDE window with two tabs: 'demonioTesting (run)' and 'feTesting (run)'. The 'demonioTesting (run)' tab is active and displays the following Java code:

```

}

@After
public void tearDown() throws Exception {
    driver.quit();
    String verificationErrorString = verificationErrors.toString();
    if (!"".equals(verificationErrorString)) {
        fail(verificationErrorString);
    }
}

private boolean isElementPresent(By by) {
    try {
        driver.findElement(by);
        return true;
    } catch (NoSuchElementException e) {
        return false;
    }
}

private boolean isAlertPresent() {
    try {
        driver.switchTo().alert();
        return true;
    } catch (NoAlertPresentException e) {
        return false;
    }
}

private String closeAlertAndGetItsText() {
    try {
        Alert alert = driver.switchTo().alert();
        String alertText = alert.getText();
        if (acceptNextAlert) {
            alert.accept();
        } else {
            alert.dismiss();
        }
        return alertText;
    } finally {
        acceptNextAlert = true;
    }
}
}

Result before: DetalleSuiteCorrida[idCorrida=0, idSuite=0, resultadoSuite=0, logSuite=, tiempoSuite=0.0]
Result after: DetalleSuiteCorrida[idCorrida=0, idSuite=0, resultadoSuite=0, logSuite=
. tiempoSuite=0.0]
javac -cp D:\Selenium\jars\*. D:\Tesis\uploads\TestSuiteItiLogeo.java exitValue() 0
DetalleSuiteCorrida[idCorrida=0, idSuite=0, resultadoSuite=1, logSuite=
. tiempoSuite=0.0]
Result before: DetalleSuiteCorrida[idCorrida=0, idSuite=0, resultadoSuite=0, logSuite=, tiempoSuite=0.0]
java -cp .;D:\Selenium\jars\*.D:\Selenium\jars\extras\*.D:\Tesis\uploads\ org.junit.runner.JUnit4 TestSuiteItiLogeo stdout: JUnit version 4.4
java -cp .;D:\Selenium\jars\*.D:\Selenium\jars\extras\*.D:\Tesis\uploads\ org.junit.runner.JUnit4 TestSuiteItiLogeo stdout: .listening on port: 51655

```

Overlaid on the right side of the IDE is a 'Firefox Update' dialog box. The dialog has the title 'Firefox Update' and a subtitle 'Checking Compatibility of Add-ons'. It features a progress bar that is nearly full and the text 'Checking your add-ons for compatibility with this version of Firefox. This may take a few minutes...'. At the bottom, there are three buttons: '< Back', 'Next >', and 'Cancel'.

Figura 11 Módulo demonioTesting ejecutando prueba

El módulo “feTesting” es una aplicación de escritorio que ayuda al tester a crear y administrar casos de prueba, suites de prueba y pruebas funcionales y de regresión (ver figura 12).

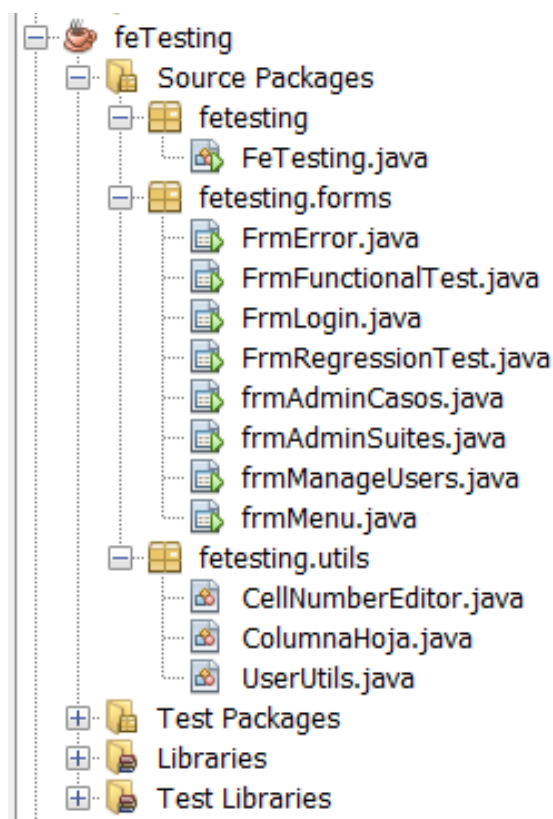


Figura 12 Estructura módulo feTesting

Una vez creada una prueba y después de haber asignado su respectiva suite, la prueba podrá ser ejecutada por el “demonioTesting”. Para esto se cuenta con una consola de administración de casos de prueba (ver figura 13) y también una consola de administración de suites de prueba (ver figura 14).

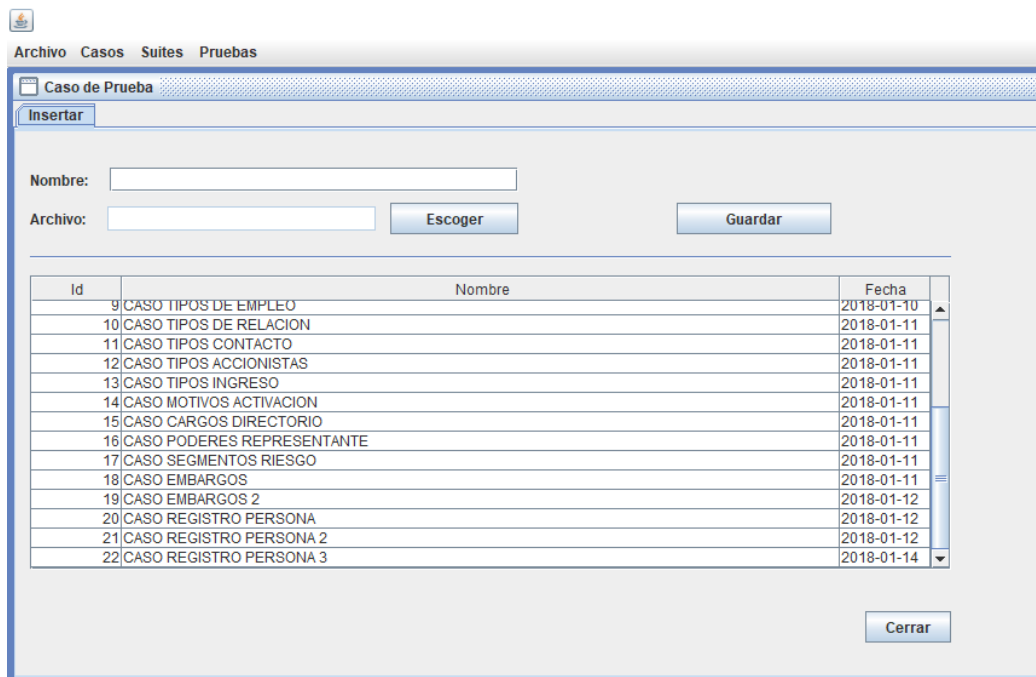


Figura 13 Ventana administración casos de prueba

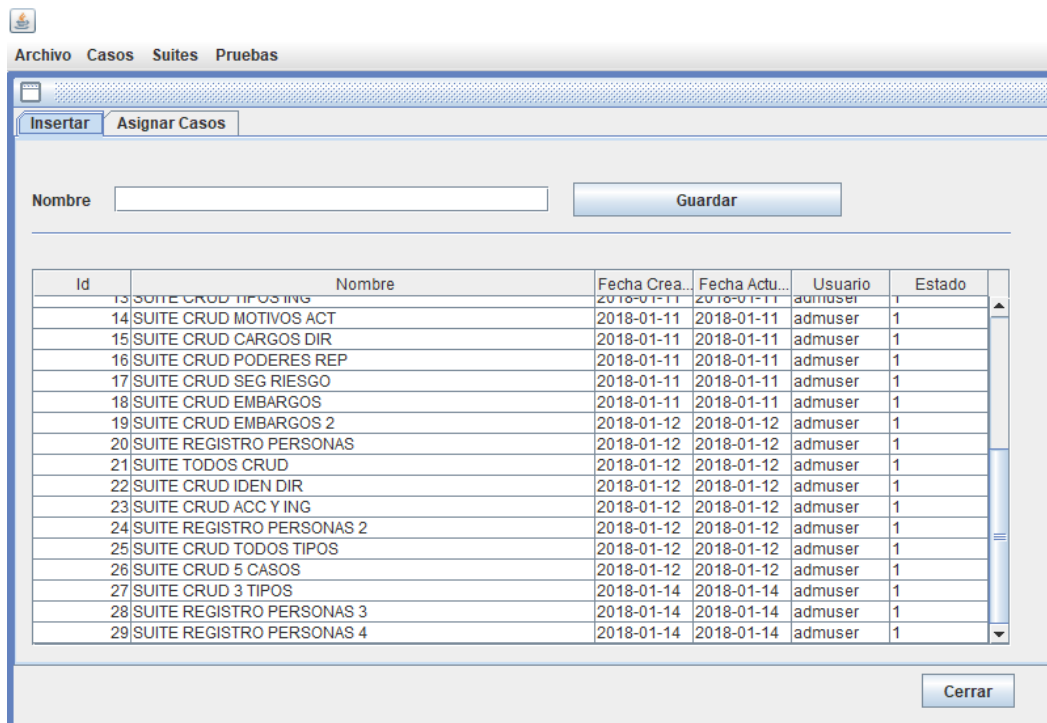
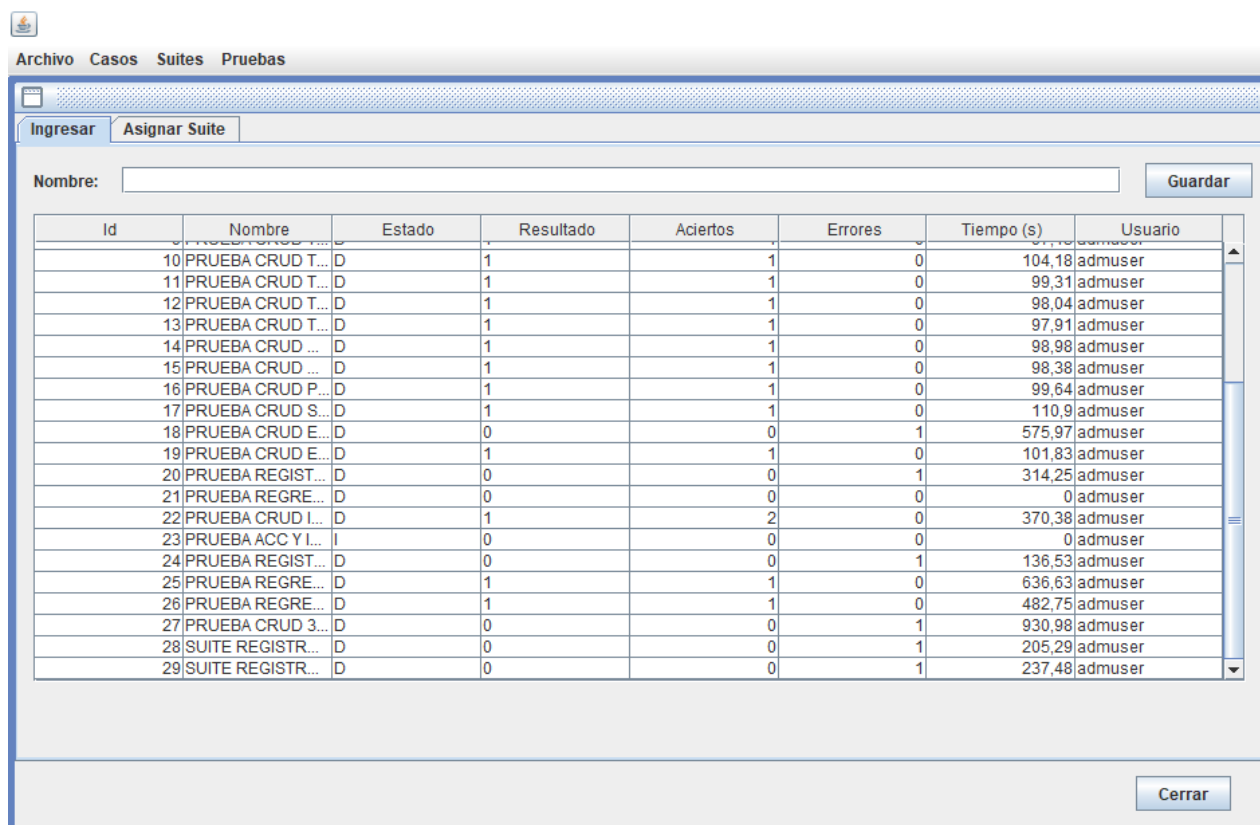


Figura 14 Ventana administración suites de prueba

Al finalizar la prueba algunos de los resultados pueden ser visualizados desde la aplicación, se podrán encontrar datos como número de errores y aciertos, tiempo total de la prueba y el resultado final (ver figura 15).



The screenshot shows a software window titled 'Pruebas' with a menu bar containing 'Archivo', 'Casos', 'Suites', and 'Pruebas'. Below the menu bar are two tabs: 'Ingresar' and 'Asignar Suite'. A 'Nombre:' label is followed by an empty text input field and a 'Guardar' button. The main area contains a table with the following data:

Id	Nombre	Estado	Resultado	Aciertos	Errores	Tiempo (s)	Usuario
10	PRUEBA CRUD T...	D	1	1	0	104,18	admuser
11	PRUEBA CRUD T...	D	1	1	0	99,31	admuser
12	PRUEBA CRUD T...	D	1	1	0	98,04	admuser
13	PRUEBA CRUD T...	D	1	1	0	97,91	admuser
14	PRUEBA CRUD ...	D	1	1	0	98,98	admuser
15	PRUEBA CRUD ...	D	1	1	0	98,38	admuser
16	PRUEBA CRUD P...	D	1	1	0	99,64	admuser
17	PRUEBA CRUD S...	D	1	1	0	110,9	admuser
18	PRUEBA CRUD E...	D	0	0	1	575,97	admuser
19	PRUEBA CRUD E...	D	1	1	0	101,83	admuser
20	PRUEBA REGISTR...	D	0	0	1	314,25	admuser
21	PRUEBA REGRE...	D	0	0	0	0	admuser
22	PRUEBA CRUD I...	D	1	2	0	370,38	admuser
23	PRUEBA ACC Y I...	I	0	0	0	0	admuser
24	PRUEBA REGISTR...	D	0	0	1	136,53	admuser
25	PRUEBA REGRE...	D	1	1	0	636,63	admuser
26	PRUEBA REGRE...	D	1	1	0	482,75	admuser
27	PRUEBA CRUD 3...	D	0	0	1	930,98	admuser
28	SUITE REGISTR...	D	0	0	1	205,29	admuser
29	SUITE REGISTR...	D	0	0	1	237,48	admuser

A 'Cerrar' button is located at the bottom right of the window.

Figura 15 Ventana administración Pruebas

4.2. Selenium

La herramienta Selenium y en específico su plugin para el navegador Firefox fue utilizado para crear los casos de prueba que posteriormente fueron almacenados en el proyecto Java.

El plugin de Selenium permite crear casos de prueba dentro de los cuales podemos grabar muchas de las acciones que realizamos al navegar por una página web, tales

como ingresar datos dentro de una caja de texto o dar clic sobre un botón, pero además de grabar acciones también permite realizar comparaciones de texto e incluso de un formato CSS en específico. Esto es visible en la figura 16.

The screenshot displays the Selenium IDE interface. On the left, a browser window shows the GESTOR website with a login form. The main area shows a list of test cases, with '13-CRUD_TIPOS_DE_INGRESO' selected. To the right, a table lists the recorded actions for this test case.

Command	Target	Value
click	id=frmGestorI_dtd66	
click	id=frmGestorIntCodigo	testing
type	id=frmGestorIntNombre	test tip...
select	id=frmGestorIntEstado	label=L...
type	id=frmGestorIntObservaciones	test tip...
click	id=frmGestorIntGuardar	
click	css=div.rf-edit-c.rf-edt-c-j_dtd56 > div.rf-edt-c-c...	
click	id=frmGestorI_dtd69	
click	id=frmGestorIntCerrar	
click	css=div.rf-edit-c.rf-edt-c-j_dtd56 > div.rf-edt-c-c...	
click	id=frmGestorI_dtd67	
type	id=frmGestorIntNombre	test tip...
select	id=frmGestorIntEstado	label=...
type	id=frmGestorIntObservaciones	test tip...
click	id=frmGestorIntGuardar	
click	css=div.rf-edit-c.rf-edt-c-j_dtd56 > div.rf-edt-c-c...	
click	id=frmGestorI_dtd68	
click	id=frmGestorI_dtd82	
click	link=FGUERRA	
clickAndWait	link=Salir	

The bottom log window shows the following entry:

```
Log Reference Expert UI-Element Rollup
clickAndWait(locator)
Generated from click(locator)
Argumentos:
• locator - an element locator.
```

Figura 16 Acciones grabadas por plugin Selenium

Las acciones grabadas en cada caso se guardan en una estructura HTML que es entendida por el servidor de pruebas de Selenium, pero para nuestra investigación fue necesario exportar las acciones almacenadas al formato Java (ver figura 17).

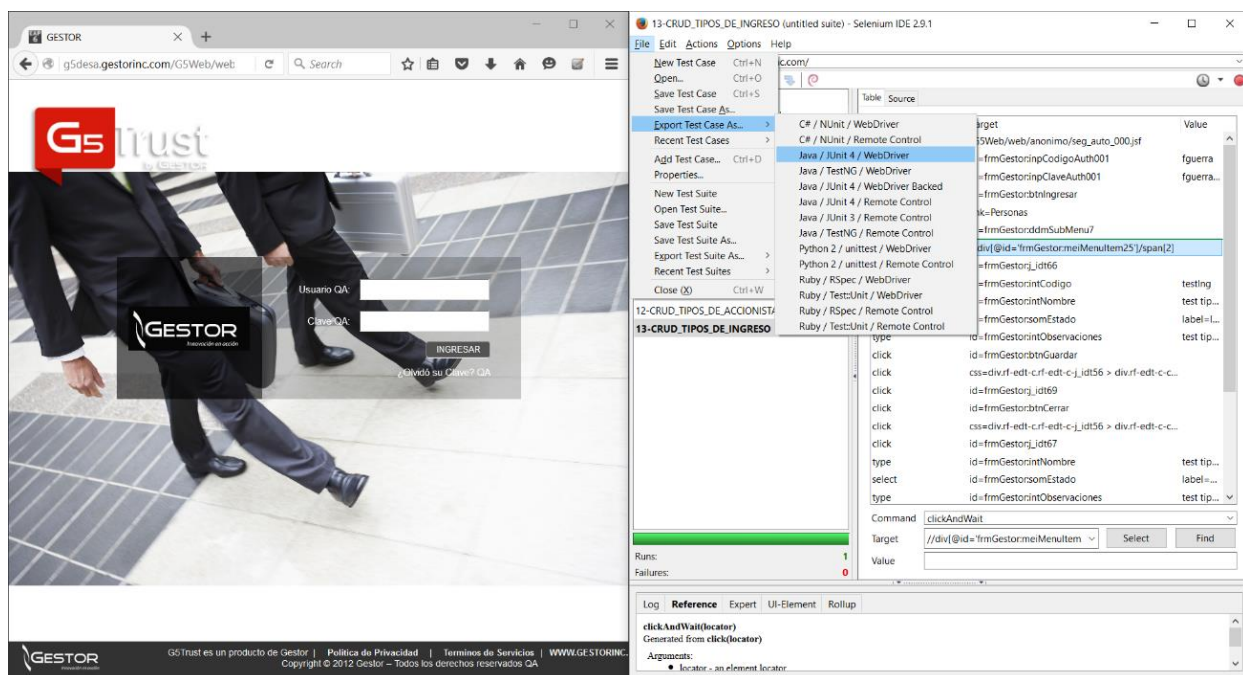


Figura 17 Opción para exportar casos a JUnit 4

La razón principal por la que los casos de prueba fueron exportados al lenguaje Java fue el hecho de que este permite un mayor control sobre el código, es decir, al manejar sentencias Java para ejecutar cada acción, grabada mediante el plugin de Selenium, se presentaron varios inconvenientes y necesidades en la ejecución de cada prueba.

- Uno de estos inconvenientes fue que, al ser sentencias secuenciales, ocurría muy a menudo que los casos de prueba resultaban erróneos a causa de los tiempos de carga que se dan entre una y otra acción dentro de una página web, muy en especial al dar clic en un botón que ejecute alguna transacción o cambio de página. Este problema fue solucionado añadiendo un tiempo de espera después de ejecutar un evento clic.

- Otro inconveniente encontrado fue el hecho de que al no poder ejecutar una sentencia, como por ejemplo ingresar texto dentro de un cuadro de texto, se lanzaba una excepción que impedía que el resto de sentencias sean ejecutadas. Este inconveniente se solucionó mediante la inserción de una estructura try-catch que encierre a cada sentencia, permitiéndonos, además de ejecutar el resto de sentencias, imprimir dentro del log del caso cada sentencia donde ocurrió un error.
- Durante el desarrollo de la herramienta surgió la necesidad de almacenar logs y resultados de cada caso asignado dentro de una prueba, por lo que se agregó dentro del código sentencias que permitan identificar el caso que se encuentra ejecutando y los métodos necesarios para almacenar sus resultados dentro de la base de datos.

Así, podemos realizar una comparativa del código que genera Selenium (ver figura 18), y después cómo la herramienta desarrollada lo depura y lo deja listo para ejecución (ver Figura 19).

```
@Test  
public void test05CRUDESTADOSCIVILES() throws Exception {  
    driver.get(baseUrl + "/G5Web/web/anonimo/seg_auto_000.jsf");  
    driver.findElement(By.id("frmGestor:inpCodigoAuth001")).clear();  
    driver.findElement(By.id("frmGestor:inpCodigoAuth001")).sendKeys("fguerra");  
    driver.findElement(By.id("frmGestor:inpClaveAuth001")).clear();  
    driver.findElement(By.id("frmGestor:inpClaveAuth001")).sendKeys("fguerra12");  
    driver.findElement(By.id("frmGestor:btnIngresar")).click();  
    driver.findElement(By.linkText("Personas")).click();  
}
```

Figura 18 Fragmento código generado por Selenium

```

@Test
public void testSuiteItiLogo() {
    Long startTime;
    Long endTime;
    Long totalTime;
    String currentLine;
    int currentCase;
    String linesByCase;
    boolean isError;
    int currentIteration;
    currentLine = "";
    currentCase = 5;
    linesByCase = "";
    startTime = System.currentTimeMillis();
    isError = false;
    currentIteration = 1;
    try
    {
        currentLine = "driver.get(baseUrl + \"/G5Web/web/anonimo/seg_auto_000.jsf\");";
        driver.get(baseUrl + "/G5Web/web/anonimo/seg_auto_000.jsf");
    } catch (Exception ex) {
        System.out.println("Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]);
        isError = true;linesByCase += "Exception Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]+" \n";
    }
    catch (AssertionError ex) {
        System.out.println("Assertion Error linea: \n"+currentLine+" Descripción: "+ex.toString().split("\n")[0]);
        isError = true;linesByCase += "Assertion Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]+" \n";
    }
    try
    {
        currentLine = "driver.findElement(By.id(\"frmGestor:inpCodigoAuth001\")).clear();";
        driver.findElement(By.id("frmGestor:inpCodigoAuth001")).clear();
    } catch (Exception ex) {
        System.out.println("Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]);
        isError = true;linesByCase += "Exception Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]+" \n";
    }
    catch (AssertionError ex) {
        System.out.println("Assertion Error linea: \n"+currentLine+" Descripción: "+ex.toString().split("\n")[0]);
        isError = true;linesByCase += "Assertion Error linea: "+currentLine+" Descripción: "+ex.toString().split("\n")[0]+" \n";
    }
}

```

Figura 19 Fragmento de código generado por Proyecto Testing

4.3. Pruebas funcionales

Dentro del proyecto las pruebas funcionales son aquellas pruebas que se basen en un caso creado en Selenium, es decir, una prueba funcional será aquella que ejecute un caso que acaba de ser importado desde Selenium y aún no ha sido ejecutado anteriormente dentro de otra prueba creada desde el módulo “feTesting”. En la figura 20 se puede ver un ejemplo de una prueba funcional que va a ser ejecutada por la herramienta.

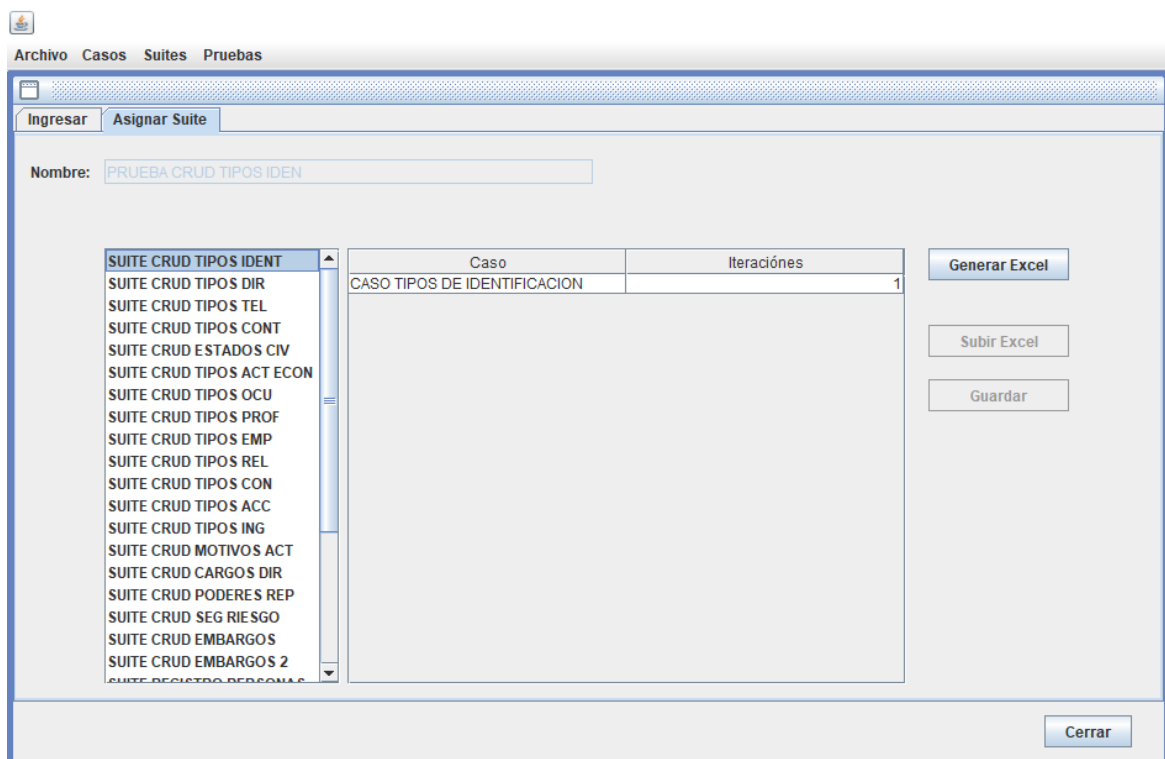


Figura 20 Ejemplo prueba funcional

4.4. Pruebas de regresión

Las pruebas de regresión son aquellas que ejecutan uno o más casos que hayan sido ejecutados anteriormente por una prueba Funcional creada desde el módulo “feTesting” y cuyo resultado haya sido positivo, es decir, las pruebas de regresión deberán contener únicamente casos de prueba que se hayan verificado que corrían sin ningún problema, ya que el objetivo de las pruebas de regresión es validar que un cambio en el código de un módulo o pantalla en particular no cause problemas en otros módulos o pantallas que previo al cambio realizado funcionaban correctamente. Un ejemplo de una prueba de regresión se puede observar en la figura 21.

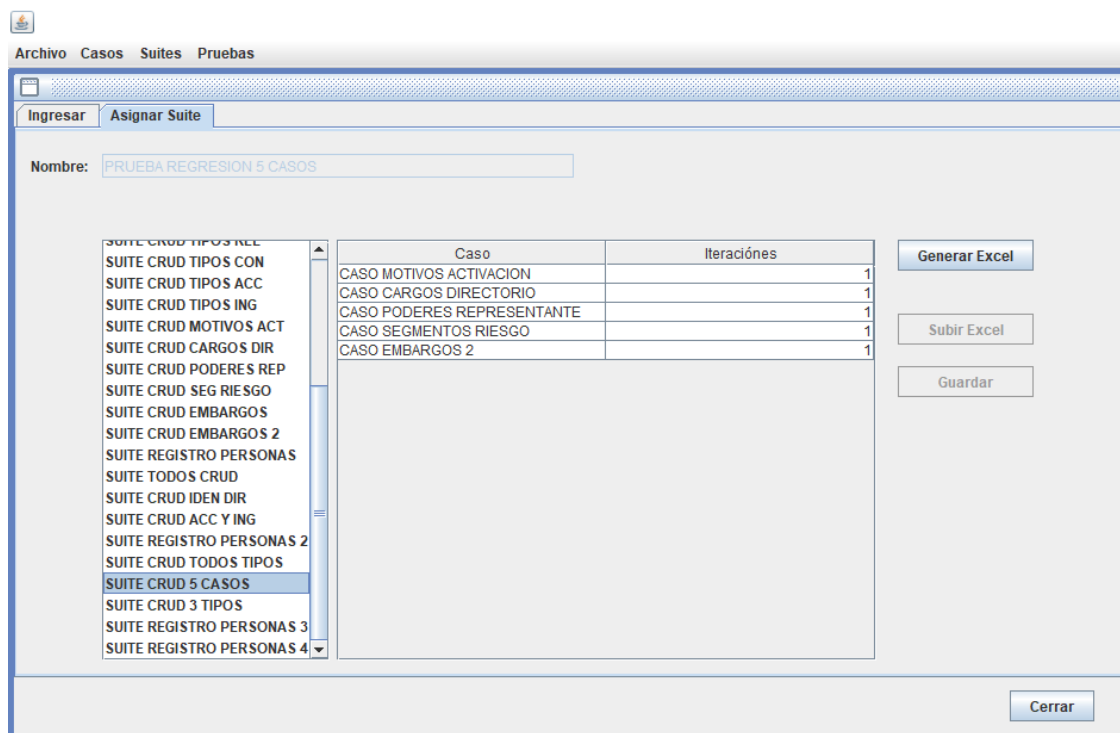


Figura 21 Ejemplo prueba de regresión

3.2.5. Base de Datos

Se ha elegido MySQL como gestor de base de datos. El nombre de la base de datos se ha designado como “dbTesting”, en la misma se almacenan todas tablas y sus relaciones necesarias para gestionar la información de casos, suites y pruebas, tanto funcionales como de regresión. La figura 22 muestra el diagrama físico de la distribución de la base de datos del proyecto.

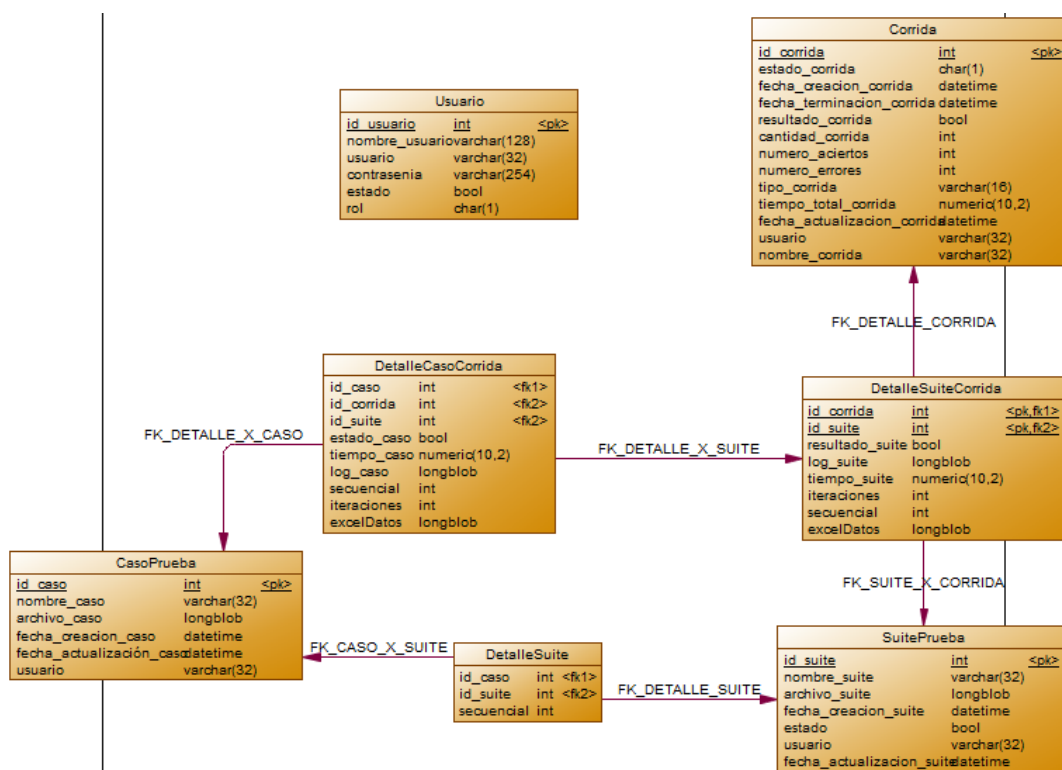


Figura 22 Diagrama físico de la base de datos

Para poder acceder a la información completa de cada prueba, ya sea funcional o de regresión, así como al detalle de resultados de cada caso asignado a la prueba ejecutada, es necesario acceder a la base de datos y realizar consultas mediante sentencias SQL. Por ejemplo en las figuras 23 y 24 podemos ver cómo se realiza la consulta de las pruebas que ya terminaron.

```
select ID_CORRIDA ID, NOMBRE_CORRIDA PRUEBA, TIPO_CORRIDA TIPO, RESULTADO_CORRIDA RESULTADO, TIEMPO_TOTAL_CORRIDA TIEMPO_TOTAL
from dbtesting.corrida
where ESTADO_CORRIDA = 'D'
```

Figura 23 Script consulta Pruebas terminadas

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
11	PRUEBA CRUD TIPOS CON	FUNCIONAL	1	99.31
12	PRUEBA CRUD TIPOS ACC	FUNCIONAL	1	98.04
13	PRUEBA CRUD TIPOS ING	FUNCIONAL	1	97.91
14	PRUEBA CRUD MOTIVOS ACT	FUNCIONAL	1	98.98
15	PRUEBA CRUD CARGOS DIR	FUNCIONAL	1	98.38
16	PRUEBA CRUD PODERES REP	FUNCIONAL	1	99.64
17	PRUEBA CRUD SEG RIESGO	FUNCIONAL	1	110.9
18	PRUEBA CRUD EMBARGOS	FUNCIONAL	0	575.97
19	PRUEBA CRUD EMBARGOS 2	FUNCIONAL	1	101.83
20	PRUEBA REGISTRO PERSONAS	FUNCIONAL	0	314.25
21	PRUEBA REGRESION CRUDS	REGRESION	0	0
22	PRUEBA CRUD IDEN Y DIR	REGRESION	1	370.38
24	PRUEBA REGISTRO PERSONAS 2	REGRESION	0	136.53
25	PRUEBA REGRESION TIPOS	REGRESION	1	636.63
26	PRUEBA REGRESION 5 CASOS	REGRESION	1	482.75
27	PRUEBA CRUD 3 TIPOS	REGRESION	0	930.98
28	SUITE REGISTRO PERSONAS 3	FUNCIONAL	0	205.29
29	SUITE REGISTRO PERSONAS 4	FUNCIONAL	0	237.48

Figura 24 Resultado consulta Pruebas terminadas

Así también es usual que se haga la consulta de aquellas pruebas que pasaron.

En las figuras 25 y 26 se puede notar que se utilizó una codificación para describir el resultado, significando cualquier número distinto de 1 que el resultado fue incorrecto.

```
select ID_CORRIDA ID, NOMBRE_CORRIDA PRUEBA, TIPO_CORRIDA TIPO, RESULTADO_CORRIDA RESULTADO, TIEMPO_TOTAL_CORRIDA TIEMPO_TOTAL
from dbtesting.corrida
where ESTADO_CORRIDA = 'D' and RESULTADO_CORRIDA = 1
```

Figura 25 Script consulta pruebas con resultado correcto

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
4	PRUEBA CRUD TIPOS CONT	FUNCIONAL	1	96.99
5	PRUEBA CRUD ESTADOS CIV	FUNCIONAL	1	96.54
6	PRUEBA CRUD TIPOS ACT ECON	FUNCIONAL	1	97.14
7	PRUEBA CRUD TIPOS OCU	FUNCIONAL	1	97.91
8	PRUEBA CRUD TIPOS PROF	FUNCIONAL	1	97.6
9	PRUEBA CRUD TIPOS EMP	FUNCIONAL	1	97.13
10	PRUEBA CRUD TIPOS REL	FUNCIONAL	1	104.18
11	PRUEBA CRUD TIPOS CON	FUNCIONAL	1	99.31
12	PRUEBA CRUD TIPOS ACC	FUNCIONAL	1	98.04
13	PRUEBA CRUD TIPOS ING	FUNCIONAL	1	97.91
14	PRUEBA CRUD MOTIVOS ACT	FUNCIONAL	1	98.98
15	PRUEBA CRUD CARGOS DIR	FUNCIONAL	1	98.38
16	PRUEBA CRUD PODERES REP	FUNCIONAL	1	99.64
17	PRUEBA CRUD SEG RIESGO	FUNCIONAL	1	110.9
19	PRUEBA CRUD EMBARGOS 2	FUNCIONAL	1	101.83
22	PRUEBA CRUD IDEN Y DIR	REGRESION	1	370.38
25	PRUEBA REGRESION TIPOS	REGRESION	1	636.63
26	PRUEBA REGRESION 5 CASOS	REGRESION	1	482.75

Figura 26 Resultado consulta pruebas con resultado correcto

4.5. Reportes

Para el despliegue de la información más relevante y que esta sea de fácil acceso, se optó por la generación de reportes, tales como:

- Listado de pruebas ejecutadas, en el cual se puede visualizar información sobre el resultado, tipo de prueba y tiempo total de cada prueba (ver figura 27).



GESTOR
Asociación en Acción

RESULTADOS PRUEBAS

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
1	PRUEBA CRUD TIPOS IDEN	FUNCIONAL	CORRECTO	99,1s
2	PRUEBA CRUD TIPO DIR	FUNCIONAL	CORRECTO	97,6s
3	PRUEBA CRUD TIPOS TELF	FUNCIONAL	CORRECTO	97,7s
4	PRUEBA CRUD TIPOS CONT	FUNCIONAL	CORRECTO	97,0s
5	PRUEBA CRUD ESTADOS CIV	FUNCIONAL	CORRECTO	96,5s
6	PRUEBA CRUD TIPOS ACT ECON	FUNCIONAL	CORRECTO	97,1s
7	PRUEBA CRUD TIPOS OCU	FUNCIONAL	CORRECTO	97,9s
8	PRUEBA CRUD TIPOS PROF	FUNCIONAL	CORRECTO	97,6s
9	PRUEBA CRUD TIPOS EMP	FUNCIONAL	CORRECTO	97,1s
10	PRUEBA CRUD TIPOS REL	FUNCIONAL	CORRECTO	104,2s
11	PRUEBA CRUD TIPOS CON	FUNCIONAL	CORRECTO	99,3s
12	PRUEBA CRUD TIPOS ACC	FUNCIONAL	CORRECTO	98,0s
13	PRUEBA CRUD TIPOS ING	FUNCIONAL	CORRECTO	97,9s
14	PRUEBA CRUD MOTIVOS ACT	FUNCIONAL	CORRECTO	99,0s
15	PRUEBA CRUD CARGOS DIR	FUNCIONAL	CORRECTO	98,4s
16	PRUEBA CRUD PODERES REP	FUNCIONAL	CORRECTO	99,6s
17	PRUEBA CRUD SEG RIESGO	FUNCIONAL	CORRECTO	110,9s
18	PRUEBA CRUD EMBARGOS	FUNCIONAL	ERRONEO	576,0s
19	PRUEBA CRUD EMBARGOS 2	FUNCIONAL	CORRECTO	101,8s
20	PRUEBA REGISTRO PERSONAS	FUNCIONAL	ERRONEO	314,2s
21	PRUEBA REGRESION CRUDS	REGRESION	ERRONEO	0,0s
22	PRUEBA CRUD IDEN Y DIR	REGRESION	CORRECTO	370,4s
24	PRUEBA REGISTRO PERSONAS 2	REGRESION	ERRONEO	136,5s
25	PRUEBA REGRESION TIPOS	REGRESION	CORRECTO	636,6s
26	PRUEBA REGRESION 5 CASOS	REGRESION	CORRECTO	482,8s
27	PRUEBA CRUD 3 TIPOS	REGRESION	ERRONEO	931,0s
28	SUITE REGISTRO PERSONAS 3	FUNCIONAL	ERRONEO	205,3s

Figura 27 Reporte de resultados de las pruebas

- Listado detallado de pruebas ejecutadas, donde además de visualizar los resultados de cada prueba también se puede visualizar el detalle de resultados de cada uno de los casos asignados a la prueba (ver figura 28).

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
22	PRUEBA CRUD IDEN Y DIR	REGRESION	CORRECTO	370.38s

DETALLE DE CASOS

ID	CASO	TIEMPO	ESTADO
1	CASO TIPOS DE IDENTIFICACION	90.46s	CORRECTO
2	CASO TIPOS DE DIRECCION	89.57s	CORRECTO

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
24	PRUEBA REGISTRO PERSONAS 2	REGRESION	ERRONEO	136.53s

DETALLE DE CASOS

ID	CASO	TIEMPO	ESTADO
21	CASO REGISTRO PERSONA 2	128.02s	ERRÓNEO

ID	PRUEBA	TIPO	RESULTADO	TIEMPO_TOTAL
25	PRUEBA REGRESION TIPOS	REGRESION	CORRECTO	636.63s

DETALLE DE CASOS

ID	CASO	TIEMPO	ESTADO
1	CASO TIPOS DE IDENTIFICACION	92.22s	CORRECTO
2	CASO TIPOS DE DIRECCION	90.22s	CORRECTO
3	CASO TIPOS DE TELEFONO	89.77s	CORRECTO
4	CASO TIPOS DE CONTRIBUYENTE	89.49s	CORRECTO
5	CASO ESTADOS CIVILES	89.02s	CORRECTO
6	CASO TIPOS ACT ECON	89.32s	CORRECTO
7	CASO TIPOS DE OCUPACION	89.26s	CORRECTO

Figura 28 Reporte detallado del resultado de las pruebas

Una vez generados los reportes requeridos, estos son enviados al área de calidad de la empresa para su posterior análisis. Al contar con un método sencillo para compartir la información de los resultados de las pruebas y al disponer con una presentación ordenada y de fácil lectura se puede reducir el tiempo que toma su análisis.

CAPÍTULO V

DESARROLLO DEL CASO DE ESTUDIO

5.1. Título

Caso de estudio de la implementación de un software para la ejecución de pruebas en aplicaciones web extendiendo la herramienta Selenium y su efecto en las características de mantenibilidad del producto GESTOR G5 TRUST.

5.2. Autor

Alejandra Dennis Ponce Guerrón y Raúl David Naranjo Erazo.

5.3. Resumen

El presente caso de estudio se desarrolló con la finalidad de establecer si la implementación de un software que automatizara pruebas mejoraría el nivel de calidad de los atributos de mantenibilidad. El caso de estudio se centra en el producto software de la empresa GESTORINC S.A. Para ello se recolectó datos del tiempo que toma crear y ejecutar una prueba con el método tradicional (manual) y el tiempo que toma el mismo trabajo con la herramienta que automatiza el proceso.

Las pruebas se llevaron a cabo en el módulo de “PERSONAS” del producto software GESTOR G5 TRUST. Los “testers” área de calidad de la empresa fueron quienes participaron en esta investigación y ayudaron a la recolección de los datos.

Con la información obtenida se desarrolló un análisis comparativo sin el uso de la herramienta para automatizar pruebas y después ya con la utilización de la herramienta.

Para la selección de los atributos que fueron evaluados se utilizó el modelo de calidad ISO/IEC 25010 y para el proceso de evaluación la norma ISO/IEC 25040. Finalmente se obtuvieron resultados positivos que muestran mejoras en los atributos de mantenibilidad del producto.

5.4. Introducción

5.4.1. Planteamiento del problema

El problema base que motivó esta investigación es el actual proceso de pruebas que mantiene la empresa GESTORINC S.A., el cual no es lo suficientemente eficiente como se desea en el área de desarrollo. Para obtener mayor información de la problemática y poder identificar las causas y los efectos de esta situación, se realizó una entrevista informal al jefe del área de pruebas y a los “testers”, así como también se realizaron actividades de observación (durante la ejecución de las pruebas) y análisis de reportes de los resultados de las pruebas. De este proceso, se obtuvo el siguiente diagrama de Ishikawa (ver figura 29):

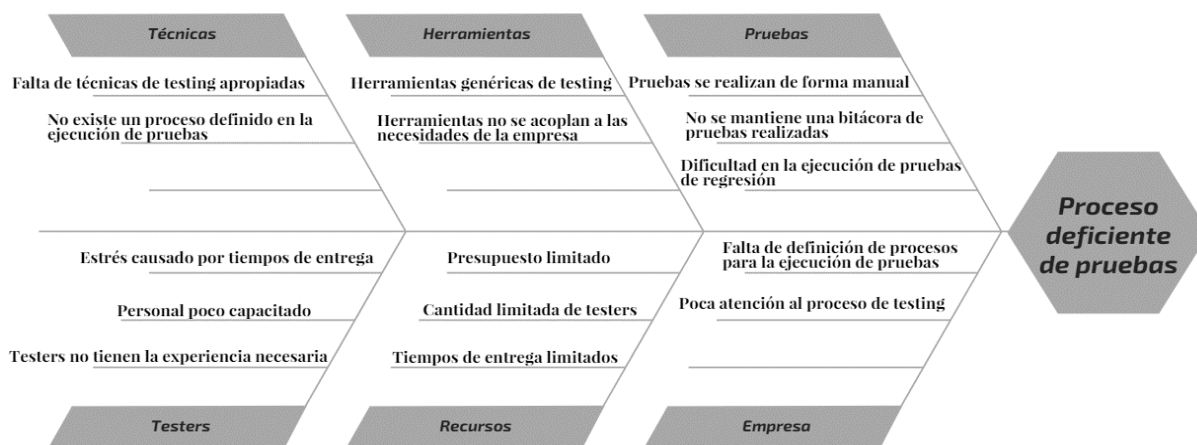


Figura 29 Diagrama de Ishikawa de la problemática

En base a la entrevista desarrollada y a los análisis de los reportes realizados, se pudieron notar los siguientes precedentes:

- Dado que las pruebas son llevadas a cabo de manera manual, cada vez que se realiza una modificación, todo el proceso de prueba se debe rehacer.
- Las herramientas genéricas que existen en el mercado para automatizar pruebas no se acoplan a los requerimientos de la empresa.
- Las herramientas de automatización que sí lo hacen, resultan muy costosas.
- La documentación asociada a la creación y ejecución de las pruebas es exhaustiva.
- En ciertas ocasiones existe desconocimiento por parte del equipo de pruebas de los procesos establecidos por la empresa para desarrollar pruebas, lo que ocasiona que las técnicas y métodos utilizados no sean los apropiados para el tipo de prueba que se realiza.
- Los nuevos “testers” tardan en acoplarse y aprender el procedimiento organizacional de como ejecutar una prueba.

A raíz de estos acontecimientos, se identificaron las siguientes causas:

- Retraso en las entregas.
- Mayores costos en el proceso de desarrollo.
- Tiempos más largos de pruebas y re-trabajo por parte de los testers.
- Insatisfacción del cliente con el producto.

- Disminución de la productividad.

5.4.2. Objetivos de la investigación

El objetivo principal de esta investigación es establecer la relación que tiene la implementación de un software para la ejecución de pruebas en aplicaciones web, extendiendo la herramienta Selenium en los atributos de mantenibilidad del producto GESTOR G5 TRUST. Con esto, se pretende demostrar que los procesos que automatizan las actividades de pruebas para un producto software tienen una incidencia positiva en el nivel de mantenibilidad, especialmente en las capacidades del software para ser modificado durante su ciclo de vida y poder ser probado.

5.4.3. Contexto

El caso de estudio se llevó a cabo para el producto GESTOR G5 TRUST, en su módulo de personas. El tiempo de desarrollo de la herramienta para automatizar las pruebas tomó aproximadamente 4 meses con el uso de una metodología ágil. Después de cada ciclo de desarrollo se obtuvo la retroalimentación de un “tester” del área de pruebas sobre funcionalidades que debían ser agregadas y/o mejoradas, hasta obtener la versión final. Después de esto se calendarizó un periodo en el cuál un grupo de “testers” de la empresa fueron capacitados en el manejo de la herramienta. Así mismo, se discutió un periodo de dos semanas en los cuales los “testers” pusieron en uso la aplicación (periodo en el que se realizó la recolección de datos).

5.5. Diseño del caso de estudio

5.5.1. Preguntas de la investigación

¿Existe un efecto positivo en los atributos de mantenibilidad de una aplicación web cuando se emplean herramientas de automatización para los procesos de pruebas?

5.5.2. Selección del caso de estudio y unidades de análisis

De acuerdo a la descripción de Yin (Yin, 2003) este es un tipo de caso de estudio holístico, donde se analiza el caso como un todo, y es diferente de un caso de estudio integrado donde se examinan varias unidades de análisis dentro de un caso.

Siendo así, el caso de estudio corresponde al software GESTOR G5 TRUST y la unidad de análisis es el módulo de PERSONAS de la aplicación (ver figura 30). Para futuras investigaciones o replicaciones de este estudio se podrían considerar agregar nuevas unidades de análisis a otros módulos del software como “PRODUCTO” o “AUDITORÍA”.

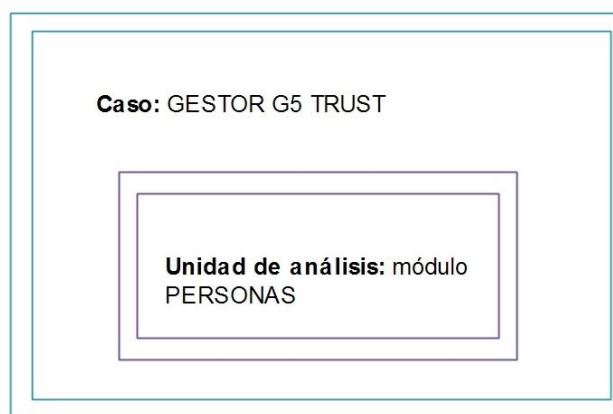


Figura 30 Esquema de caso de estudio y unidad de análisis.

Como parte de la unidad de análisis se obtuvo la participación de varios testers del área de calidad de GESTOR, quienes tienen un amplio conocimiento del problema y colaboraron voluntariamente con todo el proceso de evaluación en el uso de la herramienta de automatización.

5.5.3. Procedimientos de recolección de datos

Puesto que el objetivo es el análisis de si existe una mejora en la mantenibilidad del software, los datos que se reunieron debían establecer una comparativa del nivel de calidad del software cuando (1) se usan pruebas manuales y (2) haciendo uso de la herramienta de automatización.

El primer tipo de fuente de información (calidad con pruebas manuales) involucró el uso de un método de análisis independiente o de tercer nivel, puesto que los investigadores emplearon datos ya disponibles en el área de calidad. Estos eran registros de pruebas ya realizadas en el módulo de "PERSONAS" y que se guardaban como parte de la documentación del proyecto. Los datos que se tomaron fueron los del último ciclo de desarrollo.

Para obtener el segundo grupo de datos, se recurrió al uso de métodos directos o de primer nivel, ya que los investigadores estuvieron en contacto con los "testers" al momento de crear y ejecutar las pruebas, de manera que la recolección de los datos se realizó en tiempo real.

Para entender qué datos se tomaron en consideración, es necesario entonces establecer el proceso de evaluación de la mantenibilidad del software que se utilizó para la generación de esos datos. Este proceso y las características que se van a evaluar se describen en las siguientes subsecciones.

5.5.3.1. Características de la evaluación

La ISO/IEC 25010 describe a la mantenibilidad como “el grado de efectividad y eficiencia con la que un producto o sistema puede ser modificado para mejorarlo, corregirlo o adaptarlo a los cambios en el entorno y en los requisitos” (ISO/IEC 25010, 2014). Dentro de esta característica se abarcaban la modularidad, reusabilidad, analizabilidad, modificabilidad y testabilidad, como se puede ver en la figura 31. De estos atributos, la investigación se ha enfocado en la mantenibilidad, y específicamente en la **modificabilidad** y la **testabilidad**.

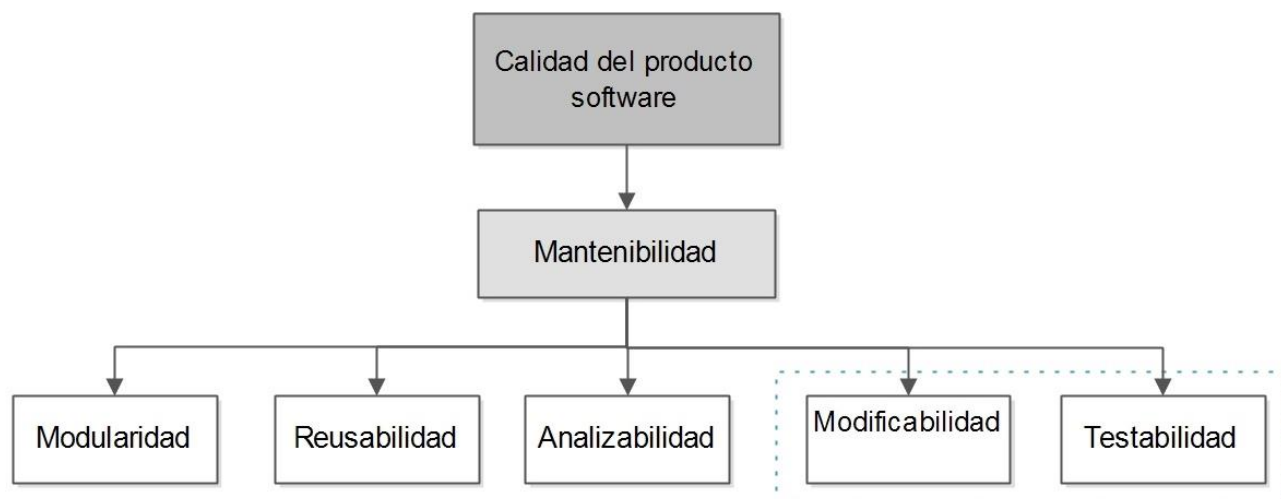


Figura 31 División de la característica de mantenibilidad según ISO 25010.

Fuente: (ISO/IEC 25010, 2014)

Adaptado por los autores.

La modificabilidad es el “grado en que un producto o sistema puede ser modificado de manera efectiva y eficiente sin introducir defectos o degradar la calidad del producto existente”, es decir la capacidad de ser modificado. La testabilidad es el “grado de eficacia y eficiencia con el que se pueden establecer los criterios de prueba para un sistema, producto o componente y se pueden realizar pruebas para determinar si se han cumplido esos criterios”, es decir la capacidad de ser probado (ISO/IEC 25010, 2014).

5.5.3.2. Proceso de evaluación

El proceso de este estándar ISO/IEC 25040f que aplicado a la investigación como una guía para la finalidad de analizar los datos del caso de estudio, que es el objetivo final de este trabajo. Cómo se llevaron a cabo cada una de estas actividades se describe en la tabla 1.

Tabla 1

Proceso de evaluación de atributos de calidad de GESTOR G5 TRUST

Actividad	Descripción
Establecer los requerimientos de evaluación	El objetivo es medir la calidad de los atributos de modificabilidad y testabilidad de GESTOR TRUST G5. El módulo que se probó es el de PERSONAS de la aplicación web. Las partes interesadas son los desarrolladores y los “testers”.
Especificar la evaluación	Medir la capacidad del sistema para ser modificado y probado. Las mediciones se realizaron bajo dos modalidades, con el uso de pruebas manuales y con la implementación de pruebas automáticas.

CONTINÚA →

La medida que se empleó es **el tiempo utilizado para elaborar y ejecutar una prueba**, mismo que se midió en segundos.

En total se desarrollaron 25 pruebas:

- 20 pruebas funcionales.
- 5 pruebas de regresión (que son varias combinaciones de pruebas funcionales en total)

Mayor detalle de estas pruebas se puede observar en la tabla 2 y tabla 3.

Diseñar evaluación	la	El proceso de evaluación se desarrolló en el periodo de dos semanas, con usuarios de prueba, de manera remota y fuera de las horas de uso normal del sistema
Ejecutar evaluación	la	Los resultados de la evaluación se describen en la sección 5.6 y fueron utilizados para cumplir con los objetivos del caso de estudio.
Concluir evaluación	la	Los resultados del análisis comparativo de los niveles de calidad de ambas pruebas fueron compartidos con el área de calidad de la empresa, dentro del reporte que se entregó como parte del caso de estudio

Las pruebas funcionales y de regresión que se aplicaron en el proceso de evaluación son las siguientes:

Tabla 2
Descripción de las pruebas funcionales

Código de la prueba	Descripción de la prueba
PF1	CRUD de tipos de identificación.
PF2	CRUD de tipos de direcciones.
PF3	CRUD de tipos de teléfono.
PF4	CRUD de tipos de contribuyente.

CONTINÚA →

PF5	CRUD de estados civiles.
PF6	CRUD de tipos de actividad económica.
PF7	CRUD de tipos de ocupación.
PF8	CRUD de tipos de profesión.
PF9	CRUD de tipos de empleados.
PF10	CRUD de tipos de relaciones.
PF11	CRUD de tipos de contactos.
PF12	CRUD de tipos de accionistas.
PF13	CRUD de tipos de ingresos.
PF14	CRUD de tipos de motivos de activación.
PF15	CRUD de cargos de directorio.
PF16	CRUD de poderes representantes.
PF17	CRUD de segmentos de riesgo.
PF18	CRUD de embargos 1.
PF19	CRUD de embargos 2.
PF20	Registro de personas.

Las pruebas de regresión involucran una combinación de las pruebas funcionales. Esta combinación ya se encontraba establecida por el área de calidad de GESTOR. Una lista de las pruebas de regresión realizadas se encuentra en la tabla siguiente:

Tabla 3
Descripción de las pruebas de regresión

Código de la prueba	Descripción de la prueba
PR1	Incluye PF1, PF2, PF3, PF4, PF5, PF6 y PF7.

CONTINÚA →

PR2	Incluye PF14, PF15, PF16, PF17 y PF19.
PR3	Incluye PF6, PF8 y PF12.
PR4	Incluye PF1 y PR2.
PR5	Incluye todas las pruebas funcionales. Esta prueba fue diseñada específicamente para esta investigación.

5.5.4. Procedimientos de análisis de datos

Esta investigación presenta un análisis de datos cuantitativo. Los datos recolectados con las dos evaluaciones de calidad, teniendo como métrica el tiempo de creación y ejecución de una prueba, permitieron realizar promedios, porcentajes de incremento/decremento y modelos de cómo fueron afectados los atributos de modificabilidad y testabilidad del producto software (que a su vez representan el nivel de calidad).

Primero se realizó una media de los tiempos de pruebas de todos los testers, esto con la finalidad de tener una visión en general de cuánto tiempo toma desarrollar y correr las pruebas para el módulo de PERSONAS. Esto se realizó tanto con el modo de pruebas manuales y pruebas automáticas, para pruebas funcionales y de regresión. Posteriormente estos tiempos fueron promediados entre sí para establecer el tiempo que toma en crear y ejecutar una única prueba. Estas métricas permitieron establecer comparaciones de cuánto tiempo extra toma la técnica manual versus la técnica automática, estas comparaciones se muestran en forma de porcentajes de incremento.

Así también, se utilizaron estos tiempos para inferir un modelo de proyección del tiempo que tomaría cada técnica en varias instancias de pruebas; es decir, cuando las pruebas se corren una segunda o tercera vez, misma situación que es muy típica. Las curvas de estos modelos así mismo permitieron responder a la pregunta de investigación, confirmando mayor eficiencia en el proceso de pruebas.

5.5.5. Procedimientos de validación de la información

Como parte del proceso de validación, solamente se incluyeron en la recolección de datos aquellos que cumplían con la métrica seleccionada para el proceso de evaluación. Así mismo, ante la presencia de datos incompletos o inconsistentes (tanto para el tester como para una prueba), estos fueron descartados. Para los datos históricos, se utilizaron los del último ciclo de pruebas del producto.

En lo que respecta al ámbito legal y ético, se obtuvo una carta de auspicio de la empresa GESTORINC S.A. y se firmó un acuerdo de confidencialidad, de manera que la información de clientes y/o proyectos no se divulgara. Los participantes que colaboraron con esta investigación fueron asignados por el área de calidad y se mantuvo con ellos una relación cordial. Así también se coordinó la entrega de los resultados del estudio a la entidad competente como parte del proceso de retroalimentación.

5.6. Resultados

El proceso de evaluación llevado a cabo utilizando el estándar ISO/IEC 25040 y basándose en las características del estándar ISO/IEC 25010 como modelo de calidad de un producto software, permitieron hacer un análisis cuantitativo de cómo se vieron afectados positivamente algunos atributos de calidad relacionados con la mantenibilidad del producto GESTOR G5 TRUST; contribuyendo así a la hipótesis inicial de que la implementación de una extensión de la herramienta Selenium para la automatización de pruebas funcionales y de regresión ayudarían a mejorar la mantenibilidad del software.

De acuerdo a la métrica establecida en el proceso de evaluación, se presentan los tiempos para creación y ejecución de una prueba de manera manual y automática.

Tabla 4
Tiempo de creación y ejecución de pruebas manuales

Prueba	Tiempo de creación (en segundos)	Tiempo de ejecución (en segundos)	Tiempo total (en segundos)
PF1	317	0	317
PF2	273	0	273
PF3	283	0	283
PF4	281	0	281
PF5	290	0	290
PF6	301	0	301
PF7	313	0	313
PF8	292	0	292
PF9	282	0	282
PF10	323	0	323

CONTINÚA →

PF11	278	0	278
PF12	275	0	275
PF13	313	0	313
PF14	277	0	277
PF15	285	0	285
PF16	329	0	329
PF17	344	0	344
PF18	518	0	518
PF19	336	0	336
PF20	943	0	943
PR1	2058	0	2059
PR2	1571	0	1571
PR3	869	0	868
PR4	590	0	590
PR5	6864	0	6864

Con esos datos se puede establecer que la media para la realización de una prueba funcional es de 403.20 segundos y el promedio para realizar una prueba de regresión es de 2630.40 segundos. Es importante mencionar que en el caso de las pruebas manuales, el tiempo total de a prueba es igual al tiempo de creación, dado que en esta modalidad mientras se va creando la prueba se da la ejecución.

Por otro lado, las pruebas funcionales y de regresión automatizadas si tienen tiempos distintos de creación y ejecución. En este caso se tomó en cuenta al tiempo de creación como el tiempo que corre desde crear la prueba con el IDE de Selenium hasta cargarla correctamente en la aplicación java desarrollada para esta investigación. El

tiempo de ejecución se enfoca solo los segundos que toma la aplicación en ejecutar las pruebas en el navegador y devolver el resultado de si la prueba pasó o no. Estos tiempos se pueden apreciar en la tabla 5.

Tabla 5
Tiempo de creación y ejecución de pruebas automatizadas

Prueba	Tiempo de creación (en segundos)	Tiempo de ejecución (en segundos)	Tiempo total (en segundos)
PF1	368	99.12	467.12
PF2	325	97.6	422.60
PF3	329	97.74	426.74
PF4	329	96.99	425.99
PF5	336	96.54	432.54
PF6	358	97.14	455.14
PF7	363	97.91	460.91
PF8	346	97.6	443.60
PF9	338	97.13	435.13
PF10	388	104.18	492.18
PF11	334	99.31	433.31
PF12	318	98.04	416.04
PF13	370	97.91	467.91
PF14	321	98.98	419.98
PF15	331	98.38	429.38
PF16	391	99.64	490.38
PF17	409	110.9	519.90
PF18	612	575.97	1187.97

CONTINÚA →

PF19	397	101.83	498.83
PF20	1103	314.25	1417.25
PR1	58	636.63	694.63
PR2	40	482.75	522.75
PR3	25	930.98	955.98
PR4	20	370.38	390.38
PR5	78	0	78.00

Los tiempos de las pruebas automatizadas generan los siguientes promedios: una media de 537.16 segundos para creación y ejecución de pruebas funcionales y una media de 640.94 segundos para pruebas de regresión. Con más detalle se tiene que:

- Promedio para creación de prueba funcional es de 348.15 segundos.
- Promedio para ejecución de prueba funcional es de 133.86 segundos.
- Promedio para creación de prueba de regresión es de 36.75 segundos.
- Promedio para ejecución de prueba de regresión es de 60519 segundos.

Para la prueba de regresión PR5 es necesario mencionar que no tiene tiempo de ejecución ya que por limitaciones de rendimiento de hardware no se pudo realizar, por lo que se la retiró de la muestra.

Finalmente, se presenta una tabla con la diferencia del tiempo de todas las pruebas manuales y las pruebas automatizadas (ver tabla 6):

Tabla 6*Diferencia de tiempos entre pruebas manuales y automatizadas*

Prueba	Tiempo para pruebas manuales (en segundos)	Tiempo para pruebas	
		automatizadas (en segundos)	Diferencia de tiempos
PF1	317	467.12	150.12
PF2	273	422.60	149.60
PF3	283	426.74	143.74
PF4	281	425.99	144.99
PF5	290	432.54	142.54
PF6	301	455.14	154.14
PF7	313	460.91	147.91
PF8	292	443.60	150.60
PF9	282	435.13	153.13
PF10	323	492.18	169.18
PF11	278	433.31	155.31
PF12	275	416.04	141.04
PF13	313	467.91	154.91
PF14	277	419.98	142.98
PF15	285	429.38	144.38
PF16	329	490.38	161.64
PF17	344	519.90	175.90
PF18	518	1187.97	669.97
PF19	336	498.83	162.83
PF20	943	1417.25	474.25
PR1	2058	694.63	1363.37

CONTINÚA →

PR2	1571	522.75	1048.25
PR3	869	955.98	89.98
PR4	590	390.38	199.62
PR5	6864	78.00	6676

5.6.1. Discusión

Para poder cuantificar la mejora en tiempo que proporciona el uso de pruebas automatizadas en el producto software se decidió realizar una comparativa en porcentajes del tiempo que toma la creación y ejecución de las pruebas tanto de manera manual como automática.

El resumen de los tiempos promedios para cada tipo de prueba, su modalidad, y la diferencia entre cada uno se pueden ver en la tabla 6:

Tabla 7

Resumen comparativo de los tiempos para pruebas manuales y automáticas

		Método manual (en segundos)	Método automático (en segundos)	Diferencia (en segundos)
Prueba funcional	Creación	343	403	61
	Ejecución	0	134	134
	Total	343	537	194
Prueba de regresión	Creación	2388	44	2344
	Ejecución	0	484	484
	Total	2388	528	1860

Primero, se puede ver que al crear y ejecutar una prueba funcional, una sola vez, resulta mucho más sencillo optar por el método manual, con el cuál aproximadamente se ahorra cerca de tres minutos del tiempo del tester por prueba. Esto implica que para realizar una prueba funcional manual se necesita de un 38% menos de esfuerzo.

Sin embargo, si se habla de ejecutar una prueba funcional más de una vez, y es lo que sucede en varias ocasiones, para la segunda instancia de la corrida de la prueba, con el método automático, solamente se debe tomar en cuenta el tiempo de ejecución; es decir en la segunda corrida el tester ha ahorrado aproximadamente 15 segundos. Para hacer una comparativa del esfuerzo que toma realizar dos veces una prueba funcional tenemos:

$$\%t = \frac{t_{TFM} * 2}{t_{TCA} + 2 * t_{EFA}}$$

Donde:

- $\%t$ representa el porcentaje de incremento del tiempo total que toma realizar la prueba dos veces.
- t_{TFM} representa el tiempo total que toma crear y ejecutar una prueba funcional manualmente.
- t_{CFA} representa el tiempo que toma crear una prueba funcional automáticamente.
- t_{EFA} representa el tiempo que toma ejecutar una prueba funcional automáticamente.

En porcentajes esto representa que al utilizar el método manual el tiempo empleado para correr dos veces una prueba funcional es de 2% más respecto del tiempo que se utilizaría si se optara por el método automático. Si se generalizara la fórmula anterior para n corridas de la prueba funcional se tiene que:

$$\%t_n = \frac{t_{TFM} * n}{t_{CFA} + t_{EFA} * n}$$

Mientras más veces se necesite ejecutar una prueba funcional la eficiencia de utilizar pruebas automáticas será mayor, tal como se muestra en la figura 31:

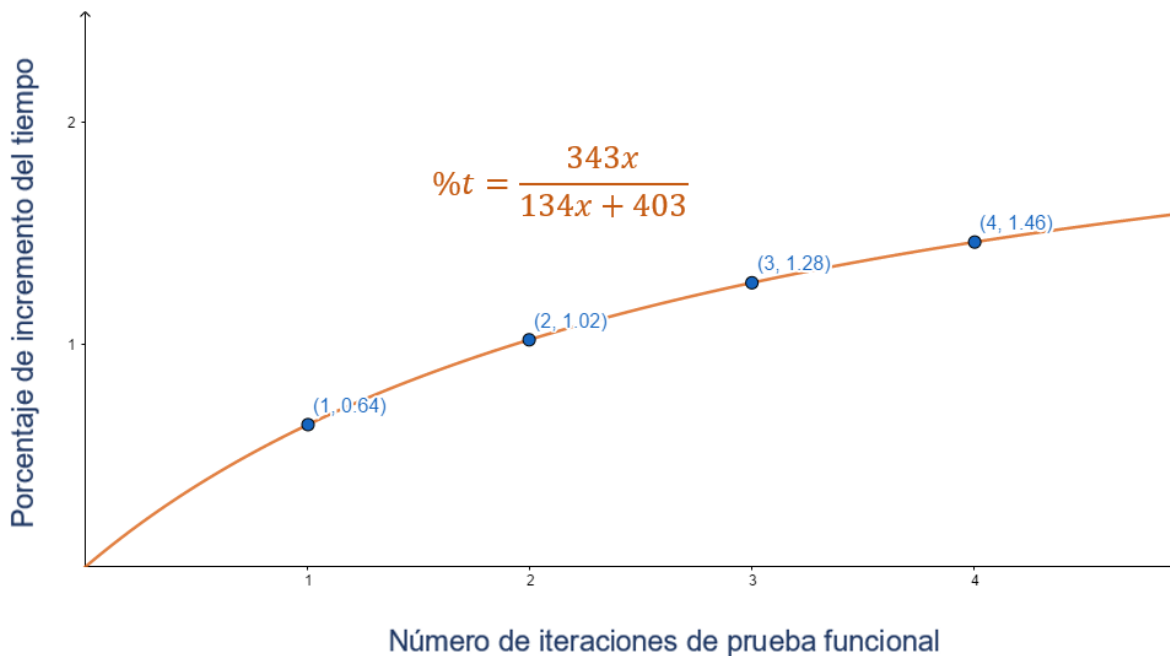


Figura 32 Gráfica de incremento de tiempo en pruebas funcionales

Cómo se puede apreciar en la gráfica de la figura 32, el uso de pruebas funcionales automáticas a largo plazo involucra menos tiempo, lo que a su vez se traduce en una mejor capacidad de ser probado y por lo tanto optimización del atributo de testabilidad.

Otro análisis se deriva claramente de los tiempos que toma el crear y ejecutar pruebas de regresión. Aquí los beneficios de automatizar pruebas de regresión se ven inmediatamente. Para este caso de estudio, con las pruebas tomadas para la muestra, el ahorro de tiempo promedio para llevar a cabo la primera vez una prueba de regresión de manera automática es de más de media hora, es decir un porcentaje de 452.3% más respecto del método manual. Esto se da porque, las pruebas de regresión consisten en la ejecución combinada de varias pruebas funcionales; entonces, si el tester dedicó tiempo a automatizar pruebas funcionales, el beneficio para las pruebas de regresión es inmediato, ya que el tiempo de creación se reduce a simplemente la selección de las pruebas que se desean correr.

Para este caso, la ecuación se decidió representar no en porcentajes de incremento, si no en número de horas ahorradas mientras más veces se deban repetir las pruebas de regresión:

$$t_n = (t_{TRM} - t_{TRA}) + (n - 1)(t_{TRM} - t_{ERA})$$

Donde:

- t_n representa el número de horas ahorradas al usar pruebas automáticas respecto de pruebas manuales en la corrida n de la prueba de regresión.
- t_{TRM} representa el tiempo total que toma crear y ejecutar una prueba de regresión manualmente.
- t_{TRA} representa el tiempo total que toma crear y ejecutar una prueba de regresión automáticamente.
- t_{ERA} representa el tiempo que toma ejecutar una prueba de regresión automáticamente.

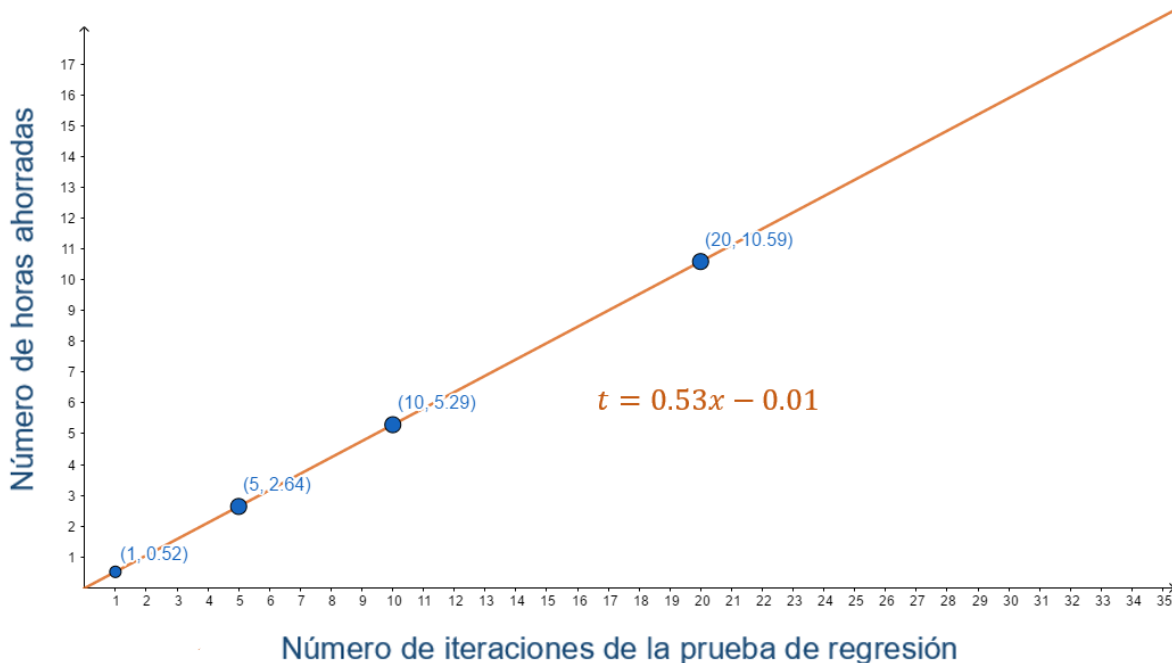


Figura 33 Número de horas ahorradas con pruebas de regresión automatizadas

Cómo se puede apreciar en la figura 33, para la corrida 20 de una prueba de regresión se ha ahorrado un total de 10.59 horas. Hay que tomar en cuenta que la ejecución de pruebas de regresión (refiriéndose en este caso tanto al proceso de creación como al de ejecución) es mucho más frecuente que la ejecución de pruebas funcionales. Dado que el objetivo de las pruebas funcionales es la verificación de cumplimiento con un requisito funcional, estas se suelen ejecutar en la empresa por lo general no más de tres veces. Por otro lado, las pruebas de regresión se ejecutan constantemente, cada vez que se realiza una modificación en la aplicación, de manera que se asegure que los cambios introducidos en el sistema fueron todos intencionales; con esto, una prueba de regresión usualmente puede ser corrida 50 veces.

Este ahorro progresivo en horas invertidas para pruebas de regresión permite deducir que efectivamente la automatización de este tipo de pruebas mejora la mantenibilidad del producto software en el hecho de que:

- Aumenta la capacidad del sistema de ser modificado: con pruebas de regresión más eficientes la probabilidad de introducir accidentalmente errores al sistema, ya sea con agregaciones, modificaciones o mejoras al mismo, es menor, lo que contribuye a que la calidad del sistema no se degrade con la evolución del producto.
- Mejora la capacidad del sistema de ser probado: similar a lo que ocurre con la automatización de pruebas funcionales, la automatización de pruebas de regresión reduce el uso de recursos (tiempo en este caso en particular), lo que se refleja en pruebas más eficientes.

5.6.2. Consideraciones

Adicional al análisis de la sección anterior que muestra mejoras en la mantenibilidad, es necesario describir algunas otras consideraciones previamente no señaladas que son relevantes a los resultados del caso de estudio:

- Parte de las métricas que se utilizaron para la evaluación de calidad fue el tiempo de creación y ejecución de una prueba. En el caso de las pruebas realizadas manualmente hay que aclarar que la métrica de creación no incluye ningún aspecto burocrático que suele ser común en procesos de gestión de calidad; es decir, esta medida no toma en cuenta generación de un reporte, apertura de un caso de prueba, etc., es exclusiva al tiempo de interacción del tester con la aplicación web. Aún el proceso de automatización de las pruebas funcionales y de regresión generó una herramienta que permite el reporte de los resultados de las pruebas, esto fue un objetivo aparte de la investigación, para facilitar la accesibilidad a la información, más no influye en la métrica de evaluación de la mantenibilidad.
- El módulo que se utilizó para el estudio (módulo de “PERSONAS”) presenta una complejidad media - baja de todas las funcionalidades que oferta el producto GESTOR G5 TRUST. Por tal razón, el número de pruebas que se podían aplicar a este elemento eran muy específicas y contadas. Con una muestra estadística de 24 pruebas, 20 de ellas funcionales y 4 de ellas de regresión, es necesario abrir la posibilidad que pudieran existir algunas amenazas a la validez de los resultados obtenidos. Si bien es cierto que las aseveraciones finales sobre la mejora de la

mantenibilidad son en su mayoría cuantitativas, se establecieron varias comparaciones en porcentajes y valores en unidades de tiempo que se incrementaron/redujeron. Aunque no se considera que la totalidad poblacional de los datos pudieran tener una desviación suficiente como para concluir que la implementación de la herramienta tuvo un efecto negativo en la calidad del producto software, sí se podrían llegar a modificar los valores en los que se incrementa la eficiencia de las pruebas automáticas. Esto en particular tiene un mayor potencial en las pruebas funcionales.

Cómo se comentó, la complejidad funcional del módulo de “PERSONAS” permitió que se pudieran automatizar la mayoría de las pruebas. Aun así, algunos de los casos de prueba que se debían implementar concurrían en errores de ejecución ya dentro de la herramienta, mismos que tenían que ser tratados y analizados para buscar una alternativa de automatización. Así mismo se presentaron escenarios donde no fue posible la automatización, esto principalmente por limitaciones de Selenium. Con estos precedentes, se podría llegar a pensar que para módulos más complejos los tiempos para automatización de pruebas funcionales podrían ser llegar a ser mayores que los que se utilizarían si se realizaran las pruebas manualmente.

- También existieron algunos impedimentos con la herramienta y el hardware sobre el cuál esta se ejecutaba:
 - Limitaciones en el número de casos que puede incurrir una prueba de regresión: cuando se empezaron a automatizar las pruebas de regresión se observó que existe un limitante en el número máximo de pruebas que se

pueden agregar. La prueba de regresión PR5 tenía como objetivo correr todas las pruebas funcionales, puesto que esta es la actividad más común cuando se lanza una adición o modificación al sistema. Al momento de ejecutar la PR5 el sistema se quedó procesando la petición y nunca llegó a ejecutar ninguna prueba.

Las pruebas de regresión se corrieron en una máquina que tiene un procesador de 8 hilos, entonces se entiende que para poder ejecutar las 20 pruebas funcionales se necesita más capacidad de procesamiento y memoria. En este caso particular, el máximo de pruebas funcionales que pudieron ser agregadas a la prueba de regresión fueron 7. Es por esto que la PR5 no se tomó en cuenta para el análisis estadístico.

- Tiempo de espera inadecuados entre comandos de los casos de prueba: dado que las pruebas se ejecutan en una aplicación web, que envía peticiones a un servidor, hay ocasiones en las que la respuesta del servidor tarda un poco en llegar; entonces si un comando no espera el tiempo suficiente, y la comparación que hace incluye elementos de la respuesta del servidor, esta instrucción va a fallar, no porque exista un comportamiento erróneo de la aplicación, sino porque aún no existe en el contexto del sistema. Esto es en particular cierto para elementos que despliegan opciones que se consultan el sistemas, como por ejemplo un control “combobox” que permite escoger el tipo de accionista, y debe consultar en el sistema la lista de opciones disponibles.

- Inconvenientes con caracteres especiales y caracteres propios de idioma: los casos de pruebas, que conforman en sí las pruebas, se basan en aseveraciones; es decir, hacen comparaciones. Por ello, habían casos en los que la información que se debía comparar en la aseveración era incorrecta, en particular para letras con acento, y por lo tanto la prueba daba un resultado incorrecto. Un ejemplo de esto es escoger una opción en un control que lea “Juan Pérez”, la letra e con acento “é” no se almacenaba ni se lea adecuadamente en la prueba, por lo que al momento de ejecución fallaba.
- Comandos no existentes: a pesar de que Selenium tiene una comunidad ávida de colaboradores y se ha dedicado a mejorar y ampliar las opciones que oferta con sus herramientas, aún hay funcionalidades y características deseables que aún no se encuentran disponibles. Un ejemplo de esto es el evento “rollOver” dentro de las interacciones de con el navegador. Encontrar alternativas para la simulación de estos eventos puede llegar a ser bastante desafiante.
- Elementos no encontrados: cuando una página es renderizada por el navegador, se crea en el contexto un identificador único para cada elemento. Muchas veces se tuvieron problemas con los casos de prueba que no encontraban ciertos elementos, ya sea porque estos tenían otros nombres o aún no se habían creado. En el caso de los nombres, es porque muchas veces los identificadores de elemento se asignan aleatoriamente; en otras ocasiones por que comandos no se lograban ejecutar, y sin ellos

consecuentemente no se creaban los elementos que la prueba buscaba en la página.

CAPÍTULO VI

CONCLUSIONES Y TRABAJOS FUTUROS

4.1. Conclusiones

Una vez finalizado el caso de estudio, con los resultados obtenidos y en base a los objetivos específicos planteados, se procede a establecer las siguientes conclusiones:

- Se desarrolló el software que extiende la herramienta Selenium de manera que se puedan automatizar pruebas funcionales y luego que estas se traduzcan a pruebas de regresión, aunque se encontraron limitaciones que condicionan la complejidad de las pruebas que pueden automatizarse, tales como: recursos de hardware sobre los que se ejecutan las pruebas que no abastecían las necesidades de procesamiento, y restricciones de la herramienta Selenium respecto de elementos de página no encontrados, controles no disponibles, caracteres no reconocidos y tiempos de espera de la aplicación.
- Se analizó la relación entre la automatización de pruebas y los atributos de mantenibilidad, en base al modelo de calidad del estándar ISO/IEC 25010 para un producto software, mostrando que los procesos de automatización tienen un efecto positivo en las características de modificabilidad y testabilidad. La métrica del tiempo total que se necesita para la creación y la ejecución de una prueba, mostró que existe una mayor eficiencia de las pruebas automatizadas sobre las pruebas que se realizan de manera manual, incrementando el ahorro de tiempo utilizado mientras más instancias de dichas pruebas se corran.

- En el caso de las pruebas funcionales, el ahorro se muestra desde la segunda ejecución (ver en la tabla 8):

Tabla 8

Porcentaje de incremento de tiempo para pruebas funcionales

#Ejecución	Método manual (en segundos)	Método automático (en segundos)	Diferencia (en segundos)	(%)
1ra	343	537	-194	63.87
2da	686	671	15	102.24
3ra	1029	805	224	127.82
4ta	1372	939	433	146.11
enésima	$n \cdot 343$	$403 + 134n$	$209n - 403$	$\frac{343n}{134n + 403}$

Este caso se presenta dado que para la primera iteración el tiempo promedio para crear y ejecutar una prueba funcional de forma manual fue de 343 segundos, mientras que el tiempo promedio para una prueba funcional automatizada fue del 537 segundos, pero para las siguientes iteraciones para el caso de la prueba funcional automatizada ya no se presentan los 403 segundos de creación de la prueba por lo que únicamente se toman en cuenta los 134 segundos de ejecución.

- En el caso de las pruebas de regresión muestran una mejora en los tiempos de ejecución desde la primera iteración, dado que el tiempo total de una prueba de regresión manual es la suma de todos los tiempos de creación y ejecución de cada una de las pruebas funcionales que esta incluye.

Tabla 9
Número de horas extra en pruebas de regresión

#Ejecución	Método manual (en segundos)	Método automático (en segundos)	Diferencia (en segundos)	# de horas extra
1ra	2388	528	1860	0.52
2da	4776	1012	3764	1.05
3ra	7164	1496	5668	1.57
4ta	9552	1980	7572	2.10
enésima	2388n	484n+44	1904n-44	0.53n-0.01

En el caso de las pruebas automatizadas el tiempo de creación es apenas el proceso de selección de las pruebas funcionales dentro del software, cuyo tiempo promedio fue de 44 segundos, y el tiempo de ejecución promedio fue de 484 segundos, dando como total 528 segundos, y finalmente obteniendo una diferencia de 1860 segundos o 31 minutos, siendo este el tiempo de ahorro entre una prueba de regresión ejecutada de forma manual y automática

4.2. Trabajos Futuros

De igual manera, una vez analizadas las problemáticas y potencialidades del proyecto se pueden mejorar en los siguientes trabajos futuros:

- Extender el número de módulos que utilizan pruebas automatizadas para replicar el análisis y comprobar si los resultados son consistentes. Como se había

mencionado previamente, el módulo “PERSONAS” de GESTOR G5 TRUST presenta una complejidad media-baja, por lo que la mayoría de las pruebas se pudieron automatizar y luego analizar; este nivel de complejidad se basa en la percepción del personal del área de calidad de GESTOR de acuerdo al número de funcionalidades del módulo. Sin embargo, en un módulo de mayor complejidad, existe la posibilidad de que Selenium no sea una herramienta adecuada para la tarea debido a limitaciones propias de la herramienta como: de elementos de página no encontrados, controles no disponibles, caracteres no reconocidos y tiempos de espera de la aplicación. Así mismo, representaría un ejercicio interesante observar si los valores de tiempo de creación y ejecución de las pruebas varían significativamente de los medidos en esta investigación, abriendo la posibilidad a porcentajes aún mayores de optimización en la capacidad de modificar y probar el software.

- Una de las observaciones más peculiares que se obtuvo en este estudio fue el desempeño de la herramienta para pruebas de regresión que involucran numerosas pruebas funcionales, mostrando que se necesita de un hardware potente para poder procesar los casos de prueba requeridos. Una investigación a futuro podría intentar establecer un límite de casos de prueba para pruebas de regresión con un hardware más robusto o establecer un aproximado de la capacidad de procesamiento que se necesita en relación al número de pruebas que se desean ejecutar.
- Intrínsecamente relacionadas con la automatización de pruebas funcionales y de pruebas de regresión se encuentran las pruebas de carga. Las pruebas de carga

involucran la ejecución en paralelo de varias pruebas funcionales con la finalidad de establecer el número máximo de instancias que la aplicación podría soportar simultáneamente. La automatización de pruebas funcionales facilitaría la automatización de pruebas de carga. Selenium cuenta con un componente que se llama Selenium Grid, que facilita la ejecución de pruebas en paralelo en varias máquinas manejando distintas versiones y configuraciones de manera centralizada. Otra propuesta de investigación podría analizar la herramienta Selenium Grid y utilizarla para automatizar pruebas de carga con las pruebas funcionales automáticas realizadas en esta investigación y ver como dicho proceso afecta a la calidad del software con las características de “eficiencia en el desempeño” y/o “confiabilidad” de la ISO/IEC 25010.

BIBLIOGRAFÍA

- Adewumi, A., Misra, S., & Omoregbe, N. (2015). Evaluating Open Source Software Quality Models Against ISO 25010. *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications*. IEEE Publications.
- Alegsa. (05 de 12 de 2010). *Alegsa*. Obtenido de <http://www.alegsa.com.ar/Dic/plugin.php>
- Andrews, A. A., OVutt, J., & Alexander, R. T. (2005). Testing Web applications by modeling with FSMs. *Software Systems and Modeling*. Springer-Verlag.
- Bangio, A., Ceri, S., & Fraternali, P. (2000). Web modeling language (WebML): a modeling language for designing Web sites. *Ninth International Conference on the WWW (WWW9)*, pp. 137–157. Amsterdam: Elsevier.
- Barrio del Castillo, I. (2018). *Universidad Autónoma de Madrid*. Obtenido de https://www.uam.es/personal_pdi/stmaria/jmurillo/InvestigacionEE/Presentaciones/Est_Casos_doc.pdf
- Binder, R. V. (1999). *Testing Object-Oriented Systems-Models, Patterns, and Tools*. Boston: Addison-Wesley.
- Bourque, P., & Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society.

- Bruns, A., Kornstädt, A., & Wichmann, D. (2009). Web Application Tests with Selenium. *vol. 26*, pp. 88-91. IEEE Software.
- Carvajal, L. (2013). Método deductivo de investigación. <http://www.lizardo-carvajal.com/el-metodo-deductivo-de-investigacion/>.
- Castro, A., Macedo, G., Collins, E., & Dias-Neto, A. (2013). Extension of selenium RC tool to perform automated testing with databases in web applications. *8th International Workshop on Automation of Software Test*. USA: IEEE Computer Society.
- CES. (2009). *ces.com.uy*. Obtenido de <http://www.ces.com.uy/index.php/ique-es-el-testing/perfil-del-tester->
- Chen, T. Y., Kuo, F.-C., Merkel, R. G., & Tse, T. H. (2010). Adaptive Random Testing: The ART of Test Case Diversity. *vol. 83(no. 1)*, pp. 60–66. Nueva York: Journal of Systems and Software.
- Conallen, J. (2000). Building Web Applications with UML. Massachusetts: Addison-Wesley.
- Cristiá, M. (11 de 2009). *Introducción al Testing de Software*. Obtenido de [fceia.unr.edu.ar](http://www.fceia.unr.edu.ar): <http://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf>
- Di Lucca, G., Fasolino, A., Faralli, F., & De Carlini, U. (2002). Testing Web applications. *International Conference on Software Maintenance*. Canadá: IEEE Computer Society.

- Dustin, E. (2002). *Effective Software Testing: 50 Ways to Improve Your Software Testing*. Addison-Wesley Longman Publishing.
- Fasolino, A. R., & Lucca, G. A. (2006). Testing Web-based applications: The state of the art and future trends. *Conference on Information & Software Technology, vol. 48*. IEEE Computer Society.
- Fenton, N., & Bieman, J. (1997). *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer.
- fergarcia. (25 de 01 de 2013). *fergarcia*. Obtenido de <https://fergarcia.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/>
- Globe Testing. (2017). *Globe Testing*. Obtenido de <https://www.globetesting.com/pruebas-ad-hoc/>
- Gómez López, R. (2004). *Evolución Científica y Metodológica de la Economía*. Publicaciones UNED.
- Graham, D. (2007). *Foundations of software testing*. Londres: Thompson.
- Holmes, A., & Kellogg, M. (2006). Automating functional tests using Selenium. *Agile Conference 2006*. Minneapolis: IEEE Computer Society.
- IEEE P730™/D8. (2012). *Draft Standard for Software Quality Assurance Processes*. IEEE.
- IEEE Std. 12207-2008. (2008). *Standard for Systems and Software Engineering—Software Life Cycle Processes*. IEEE.

ISO 9000:2005. (2005). Quality Management Systems—Fundamentals and Vocabulary. ISO.

ISO/IEC 25010. (2014). System and software quality models. *Quality Model Division*.

ISO/IEC 25010:2011. (2011). Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Systems and Software Quality Models.

ISO/IEC 25040. (2014). Evaluation reference model and guide. *Quality Evaluation Division*.

ISO/IEC/IEEE 24765:2010. (2010). Systems and Software Engineering—Vocabulary. ISO/IEC/IEEE.

Jagannatha, S., Niranjnamurthy, M., Manushree, S. P., & Chaitra, G. S. (2014). Comparative Study on Automation Testing using Selenium Testing Framework and QTP. *International Journal of Computer Science and Mobile Computing*. India.

Jia, Y., & Harman, M. (2010). An Analysis and Survey of the Development of Mutation Testing. *vol. 37(no. 5)*, pp. 649–678. *IEEE Trans. Software Engineering*.

Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering*. 2da ed. AddisonWesley.

Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). *Case Studies for Method and Tool Evaluation*. IEEE Software.

- Leotta, M., Clerissi, D., Ricca, F., & Tonella, P. (2013). Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. *20th Working Conference on Reverse Engineering (WCRE)*. Alemania: IEEE Computer Society.
- Lewis, W. E. (2014). *Software Testing and Continuous Quality Improvement*. Florida: CRC Press LLC.
- Liu, C., Kung, D. C., Hsia, P., & Hsu, C. (2000). Object-based data flow testing of Web applications. *First Asia-Pacific Conference on Quality Software*, pp. 7–16. Los Alamitos: IEEE Computer Society Press.
- López, M. A. (2015). *Mialto*. Obtenido de <http://mialtoweb.es/definicion-de-aplicacion-web/>
- Lyu, M. R. (1996). *Handbook of Software Reliability Engineering*. McGraw-Hill and IEEE Computer Society Press.
- Manchón, E. (02 de 07 de 2003). *Alzado.org*. Obtenido de https://www.alzado.org/articulo.php?id_art=40
- Manley, B. F. (2016). *Multivariate Statistical Methods - A Primer*. CRC Press.
- Mansour, N., & Hourri, M. (2006). Testing web applications. *Conference on Information and Software Technology*, vol. 48, pp. 31-42. IEEE Computer Society.
- Naik, S., & Tripathy, P. (2008). *Software Testing and Quality Assurance: Theory and Practice*. Wiley-Spektrum.
- Nielsen, J. (2013). *Usability Engineering*. Morgan Kaufmann.

Pérez Porto, J., & Merino, M. (2017). *definicion.de*. Obtenido de <https://definicion.de/feedback/>

Radigan, D. (2017). Continuous integration, explained. Atlassian. Obtenido de <https://www.atlassian.com/continuous-delivery/continuous-integration-intro>

Ricca, F., & Tonella, P. (2001). Analysis and testing of Web applications. *23rd International Conference on Software Engineering (ICSE '01)*, pp. 24-34. Washington: IEEE Computer Society.

Rouse, M. (2016). *TechTarget*. Obtenido de <http://searchdatacenter.techtarget.com/es/definicion/Framework>

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering Journal*. Springer US.

Selenium.org. (2017). *Selenium - Web Browser Automation*. Obtenido de <http://www.seleniumhq.org/>

Sommerville, I. (2011). *Software Engineering*. 9th ed. USA: Addison-Wesley.

Tian, J. (2015). *Software quality engineering*. New Jersey: Wiley.

Universidad Técnica Federico Santa María. (17 de 09 de 2012). <http://utfsm.cl/>. Obtenido de <http://wiki.inf.utfsm.cl/index.php?title=Demonios>

Utting, M., & Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann.

- Wang, X., & Xu, P. (2009). Build an Auto Testing Framework Based on Selenium and FitNesse. *International Conference on Information Technology and Computer Science*. Ucraina: IEEE Computer Society.
- Whatls.com. (2017). Obtenido de <http://whatis.techtarget.com/definition/log-log-file>
- Wohlin, C., Höst, M., & Henningsson, K. (2006). 13 Empirical Research Methods in Web and Software Engineering. *Web Engineering*. Springer Science & Business Media.
- Wynne, M., Hellesoy, A., & Tooke, S. (2017). The Cucumber Book: Behaviour-Driven Development for Testers and Developers. USA: Pragmatic Bookshelf.
- Xu, D., Xu, W., Bavikati, B. K., & Wong, W. E. (2012). Mining Executable Specifications of Web Applications from Selenium IDE Tests. *International Conference on Software Security and Reliability (SERE)*. USA: IEEE Computer Society.
- Yin, R. (2003). Case study research. *Design and methods*. Londres.
- Zelkowitz, M. V., & Wallace, D. R. (1998). Experimental models for validating technology. IEEE Computer.