ESCUELA DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

Tema:

SISTEMA AUTOMATIZADO PARA LA EJECUCIÓN DE PRUEBAS DE *SOFTWARE* FUNCIONALES Y REGRESIVAS UTILIZANDO SELENIUM Y JAVA

Proyecto de investigación previo a la obtención del título de Ingeniero en Sistemas de Información

Línea de investigación:

TECNOLOGÍAS DE LA INFORMACIÓN Y DE LA COMUNICACIÓN

Autor:

Mateo Andrés Quinteros Vivas

Director:

Mg. José Marcelo Balseca Manzano

Ambato – Ecuador

Septiembre 2025

DECLARACIÓN DE AUTENTICIDAD Y RESPONSABILIDAD

Yo: **MATEO ANDRÉS QUINTEROS VIVAS,** con cédula de ciudadanía **1850025162,** autor del trabajo graduación titulado: "SISTEMA AUTOMATIZADO PARA LA EJECUCIÓN DE PRUEBAS DE *SOFTWARE* FUNCIONALES Y REGRESIVAS UTILIZANDO SELENIUM Y JAVA", previo a la obtención del título profesional de **INGENIERO EN SISTEMAS DE INFORMACIÓN**, en la escuela de **HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD.**

1. Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2. Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través del sitio web de la Biblioteca de la PUCE Ambato, el referido trabajo de graduación, respetando las políticas de propiedad intelectual de la Universidad.

Ambato, septiembre 2025

Mateo Andrés Quinteros Vivas
CC. 1850025162

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE AMBATO

APROBACIÓN DEL TRIBUNAL DE GRADO

Tema:

SISTEMA AUTOMATIZADO PARA LA EJECUCIÓN DE PRUEBAS DE *SOFTWARE* FUNCIONALES Y REGRESIVAS UTILIZANDO SELENIUM Y JAVA

Línea de investigación:

TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

Autor:

Mateo Andrés Quinteros Vivas

José Marcelo Balseca Manzano, Ing. Mg. f._____

CC. 1802572915

CALIFICADOR

Edison Fernando Meneses Torres, Ing. Mg. f._____

CALIFICADOR

Teresa Milena Freire Aillón, Ing. Mg. f._____

CALIFICADOR

Darío Javier Robayo Jácome, Ing. Mg. f._____

DIRECTOR ESCUELA DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

Ana Cecilia Parra Ramos, Ab. Mg. f._____

SECRETARIA GENERAL PUCESA (S)

SECRETARIA GENERAL
PROCURADURÍA

Ambato – Ecuador

Septiembre 2025

RESUMEN

El proyecto que se expone en esta memoria es resultado de la necesidad de optimizar el proceso de *testing* en aplicaciones *web* en un entorno donde rapidez y calidad de las aplicaciones que se desarrollan son aspectos determinantes para la competitividad. En este contexto, la automatización de las pruebas de software se ha convertido en una herramienta indiscutible como una de las herramientas para garantizar la eficiencia, disminuir los errores manuales y hacer una cobertura del *testing* más exhaustiva. El presente trabajo tiene importancia para empresas de desarrollo de *software*, les permite implementar soluciones más fiables y sostenibles en sus procedimientos de calidad del *software*. El objetivo general de esta memoria es desarrollar un sistema automatizado para la ejecución de pruebas funcionales y también pruebas regresivas a partir de ejemplos de pruebas para el navegador *web* Selenium y lenguaje Java, para mejorar el control de calidad del *software* y reducir el tiempo dedicado a la validación de funcionalidades del mismo.

La metodología que se propone es la de implementar pruebas que simulan el comportamiento real de los usuarios con el automatizador Selenium WebDriver, acompañado además de algunos *frameworks*, como JUnit y TestNG, enmarcados dentro de un entorno de integración continua. Gracias a este planteamiento, se podrán ejecutar las pruebas de forma automática una vez realizado un cambio de código, garantizando una validación constante y eficiente por parte de la memoria; es de esperar que el sistema contribuya a mejorar la eficiencia en los procesos de pruebas y aumentar la consiguiente fiabilidad del producto final.

Este proyecto se presenta como una solución práctica y escalable para que las organizaciones de desarrollo de software puedan mejorar sus procesos relacionados con el control de calidad, así como en entornos ágiles de desarrollo de *software.*

Palabras clave: automatización de pruebas, selenium, testing funcional, java, pruebas regresivas, calidad de software.

ABSTRACT

This report outlines a project that emerged from the necessity of optimizing the testing process of web applications within an environment where the speed and quality of web applications to be developed are directly determinant factors of competitiveness, among others. As a result, automation of software testing has become an undeniable and necessary practice to ensure efficiency, decrease manual errors, and obtain greater test coverage. This line of work is beneficial to software development organizations that want to leverage a tool that permits more reliable and sustainable practices to be incorporated into their software quality assurance processes (i.e., testing). The report's objective is to develop an automated testing system to execute functional tests and regression tests based on test cases for the Selenium web browser and Java programming language to improve software quality control and minimize time spent validating functionality.

To accomplish this objective, the project proposes to execute tests that simulate real-life user behavior using the Selenium WebDriver automation engine, with JUnit and TestNG frameworks, in a continuous integration environment. Due to the nature of the continuous integration approach, tests will automatically execute once a code change is made and the build has been validated.

The project asserts that the proposed solution will contribute to enhancing the efficiency of software testing practices and further improve the eventual reliability of the final product. The project concludes with a feasible and scalable solution that enables software development organizations to improve their quality control and software testing practices, particularly those working within agile software development frameworks.

Keywords: *test automation, selenium, functional testing, java, regression tests, software quality.*

ÍNDICE GENERAL DE CONTENIDOS

INTRODUCCIÓN

La utilización de la automatización de pruebas de software en el desarrollo de aplicaciones se encuentra plenamente establecida, sobre todo en un entorno donde la rapidez, la precisión y la calidad son factores determinantes para el éxito de los proyectos; las pruebas funcionales y regresivas deben garantizar que las funcionalidades nuevas y viejas de una aplicación se comporten de forma correcta tras cada cambio o actualización del mismo. En ese momento, disponer de herramientas como *Selenium* y un lenguaje sólido como Java facilita la configuración de sistemas automatizados para realizar pruebas que optimicen los tiempos de validación, reduzcan los errores manuales y extiendan la cobertura de los ensayos. El encuentro de sinergia no solo simplifica la identificación y resolución de problemas, sino que también contribuye a proporcionar un software adecuado y eficaz, que se ajuste a las demandas de usuarios cada vez más competitivos y demandantes.

García et al. (2024) En el trabajo orientado a profundizar y ampliar la herramienta de *JUnit* 5 de nombre *Selenium-Jupiter*, esta herramienta permite automatizar la realización de pruebas si existe la presencia del gestor de controladores *Selenium WebDriver*. Este último es un programa software completamente multiplataforma que permite controlar un navegador de forma automática o programada en forma simultánea; es una biblioteca multiplataforma que permite realizar pruebas automatizadas de alto desempeño para aplicaciones web. *JUnit* es uno de los marcos de prueba unificada para Java más comunes; la versión más reciente en la actualidad es la versión *JUnit* 5 que incluye un modelo de programación y extensión al código de llamado *Jupiter*. *Selenium-Jupiter* también es responsable de la automatización del control de controladores y dispositivos, es uno de los marcos de prueba unitaria más comunes en el lenguaje de programación Java; la versión más reciente es *JUnit* 5, que incluye un modelo de programación y de extensión al código también se ocupa de automatizar la gestión de controladores y de permitir su integración con Docker para hacer que pruebas para navegador puedan ser ejecutadas en contenedores. Esto es útil en pruebas de carga o en la monitorización del rendimiento de sistemas con *WebRTC.* Esto permite optimizar la infraestructura

de las pruebas al ser más fácil la configuración y, sobre todo, la eficiencia en testabilidad en entornos de integración continua.

QAwerk (2024) estudia los retos habituales de la automatización de *tests* con *Selenium* y cómo solucionarlos para mejorar su implementación. A pesar de ser un artefacto ampliamente utilizado para la reducción de errores humanos, y contar con pruebas que sean reproducibles, *Selenium* tiene múltiples limitaciones. Los principales retos son la sincronización de ventanas emergentes, la sincronización con objetos que permiten la interacción por defecto y la estabilidad de las pruebas con diferentes combinación de entornos. Se propone mediante las siguientes estrategias de espera explícitas, un *framework* de control de flujo, la incorporación de herramientas complementarias que mejoren la estabilidad de las pruebas. La importancia de la planificación y el mantenimiento de los scripts de las pruebas automatizadas, una mala planificación y el mantenimiento de los scripts puede conducir a falsos positivos y falsos negativos. Concluye el trabajo que, pese a que *Selenium* puede considerarse una herramienta potente para la automatización de las pruebas, es una herramienta que funciona muy bien bajo la correcta implementación y la correcta monitorización.

García (2024) ofrece un completo manual acerca del uso de *Selenium WebDriver* con Java para el desarrollo de test automatizados para pruebas de extremo a extremo. Se tratan puntos como la navegación web automatizada, la interacción con elementos de contenido dinámico, la gestión con los *WebDriver*, el patrón de diseño Page *Object Model* (POM) y puesta en organización en infraestructuras remotas con Docker y servicios de *cloud computing*. El autor expone de manera muy clara la importancia que, sin duda, tiene estructurar el código del test automatizado para lograr que sea reutilizable y escalable; por otro lado, facilita ejemplos reales de que *Selenium WebDriver* se puede implementar juntamente con *JUnit* y con *TestNG* para validar aplicaciones con interfaz web. El análisis a posteriori concluye que una adecuada configuración del usuario de *Selenium WebDriver*, junto con ciertas *periculi-tools*, permite mejorar la calidad y la eficiencia de la actividad de test automatizado.

Calderón et al. (2021) ha realizado un estudio comparativo entre las herramientas para la evaluación de software modelo- basado: *GraphWalker y LeapWork*. El planteamiento del trabajo es el Testeo de Software Basado en Modelo (MBT), técnica que permite la automatización de los casos de prueba funcionales con el objetivo de maximizar la gestión de recursos en el proceso de testeo. *GraphWalker*, una herramienta multiplataforma, consigue describir el comportamiento del sistema con máquinas de estados finitos y atestigua la automatización de pruebas funcionales para aplicaciones de escritorio y de web, *LeapWork* destaca por implementar una interfaz amigable, sin requerir conocimientos previos de programación, a la vez que permite automatizar tareas repetidas. Criterios de comparación, dentro de los cuales se encuentra la facilidad de uso y capacidades de automatización, así como la relevancia de las herramientas en los distintos espacios de los diferentes escenarios de pruebas, son constitutivas de una información útil para la comunidad universitaria y profesional que desea seleccionar herramientas de MBT adaptadas a su propio proyecto de desarrollo.

Mendoza et al. (2020) tuvieron la atención puesta en un contexto que hace uso de la automatización de pruebas, como *Selenium y JUnit*, en una empresa ecuatoriana dedicada al desarrollo de software. Hay que resaltar la importancia de la automatización de algunos procesos para el poder detectar de forma anticipada los fallos que pueda tener el producto, ahorrando costes relacionados con la detección y corrección de fallos y, en última instancia, disponer de un software estable en un entorno de producción. Los autores igualmente definiendo un conjunto indicador de medición de la duración, de la cobertura del código o de la efectividad en la detección de fallos, en entornos de pruebas funcionales de aplicaciones web, su investigación también tenía un interés por estudiar el modo en el que la prueba termina siendo incorporado a los sistemas dentro de CI/CD dada la aceleración del *testing* para obtener calidad del software. La investigación concluye que la automatización en el país va en aumento, aunque también entran en lucha las empresas para acomodarse a esta serie de cambios y la necesariamente formación efectiva de los trabajadores.

Hurtado y otros (2019) realizaron una investigación de un trabajo al que titularon "Automatización de Procesos de Investigación, Vinculación, Prácticas y Pasantías

Preprofesionales en Universidades Ecuatorianas", donde se plantea la automatización de diferentes procesos tanto de tipo escolar, como administrativos en instituciones de educación superior en Ecuador. Para la elaboración del software, se empleó la metodología *eXtreme Programming* (programación extrema), que se encuentra sustentada por valores como comunicación, simplicidad, *feedback* y valor. El software pasó una serie de evaluaciones donde se realizaron evaluaciones de usuarios y de directivos, se implementaron una serie de pruebas funcionales y se sometió al plan de evaluación de calidad web según la metodología Evaluación de Calidad de Sitios Web (ECSW); los resultados muestran una mejora en la eficiencia y calidad de los procesos automatizados, y una serie de beneficios directos a las instituciones educativas.

Descripción del problema

En la mayoría de los entornos donde se trabaja con el desarrollo de *software*, existe un escaso enfoque sistemático en la automatización de pruebas, manifestando un obstáculo significativo para asegurar la calidad del producto. Esta deficiencia puede darse por varios factores como el desconocimiento sobre herramientas de automatización útiles o incluso rechazar cambios en los grupos de trabajo por falta de recursos técnicos. Aplicar estas herramientas permite mejorar sustancialmente la efectividad en la validación de sistemas reduciendo el tiempo necesario para ejecutar pruebas repetitivas y simplificar la detección temprana de errores. Asimismo, se asegura una mayor solidez en los resultados al tratar de minimizar los fallos procedentes de la intervención humana. Esta automatización también se alinea con procesos modernos como metodologías ágiles y *DevOps*, esto ayudando a entregas más confiables y rápidas. Como consecuencia, no solo se aumentará mejores condiciones para el software, sino también la competencia de la organización o de la empresa en el mercado.

Objetivo general

Establecer un sistema automatizado de evaluaciones de *software* fundamentado en *Selenium y Java*, para el incremento de la eficiencia y calidad presentes en los procedimientos de prueba de las aplicaciones web.

Objetivos específicos

1. Plantear un estudio de las herramientas y técnicas existentes de automatización de pruebas de software tomando como referencia la de *Selenium* y *Java* para la automatización de pruebas de aplicaciones web.
2. Examinar los procedimientos de pruebas existentes, ofreciendo los inconvenientes y oportunidades de mejora a través de la automatización.
3. Desarrollar un sistema básico de pruebas automatizadas mediante *Selenium* y *Java*, integrándolos con herramientas del tipo *TestNG* o *JUnit* incluyendo dentro de un sistema de integración continua.

Metodología

La metodología de desarrollo del tiene un enfoque cuantitativo y experimental y cuya ejecución se articula en cinco etapas de trabajo principales. En primer lugar, un análisis de requisitos para definir las necesidades objetivas importantes así como para establecer el alcance del sistema; en segundo lugar, el diseño de la arquitectura y de los casos de prueba automatizadas; en tercer lugar, el desarrollo del prototipo que integra *Selenium y Java*, las herramientas que se incorporan en el desarrollo del proceso como *JUnit y TestNG* para la gestión de las pruebas; en cuarto lugar, la validación del sistema, a partir de pruebas reales, comparando su eficacia y su precisión a métodos manuales; en quinto lugar, la documentación de los resultados, los beneficios obtenidos en tiempo, en calidad y en cobertura; se aportan las evidencias que se alinean positivamente con el contexto de las pruebas automatizadas y su práctica para los métodos de desarrollo ágiles.

Justificación

La proposición de este sistema conjuntamente con Java está apoyada en la necesidad progresiva de proveer la calidad del software particularmente dentro de un mercado muy competitivo y que cambia constantemente. En este ambiente tecnológico como son la automatización de pruebas se las representa como una táctica clave para respaldar entregas con un alto nivel de rapidez y confiabilidad. Aun así, las pruebas manuales siguen siendo un recurso perfecto en ciertos escenarios específicos. Como por ejemplo, son ideales a lo largo de las primeras

fases de desarrollo, cuando los requisitos aún no están listos y definidos o son propensos a un cambio constante. De la misma forma resultan adecuados para la evaluación de ciertos aspectos subjetivos ya sea en la experiencia del usuario (UX), la accesibilidad general o la experiencia que tiene el usuario con el sistema, donde el análisis humano es fundamental. Por ende, lejos de ser excluyentes, estas pruebas manuales y automatizadas se perfeccionan para asegurar la calidad total de las aplicaciones.

Aunque están sujetas a la errata humana, consumen un tiempo y un esfuerzo extra, y ofrecen limitaciones en la cobertura así como en la repetitividad. Por el contrario, la automatización contribuye a acelerar el proceso, al permitirnos ofrecer tiempos de pruebas mucho más cortos, incrementar la precisión en la detección de errores y obtener el resultado en condiciones repetibles en múltiples entornos y configuraciones.

El uso de *Selenium* para automatizar las pruebas funcionales, se debe a la flexibilidad que demuestra, soportando la ejecución sobre aplicaciones web a través de un conjunto de navegadores; y Java, al tratarse de un lenguaje robusto y ampliamente usado de la industria, ofrece la posibilidad de asegurar una implementación escalable y ajustada. Usando este sistema automatizado se espera ayudar a incrementar la eficiencia operativa y la calidad de los productos ofrecidos; así como la reducción de costes y el plazo de desarrollo, buscando el beneficio de los desarrolladores y de las empresas que buscan soluciones más rápidas y fiables de las pruebas de calidad del *software*. Asimismo con este trabajo se intenta contribuir al desarrollo de metodologías de prueba, en el ámbito local e internacional y fomentar el uso de las herramientas tecnológicas modernas en el ámbito del *software.*

CAPÍTULO I. ESTADO DEL ARTE Y LA PRÁCTICA

1.1. Perfección de la automatización a la hora de desarrollar *software* actual

La automatización de pruebas del software ha evolucionado sustancialmente a lo largo de estos últimos años, conformándose, junto con la integración continua, en uno de los pilares fundamentales dentro del desarrollo ágil. La automatización de pruebas de software, tal y como nos recomiendan Álvarez et al. (2023), ayuda a elevar la calidad y velocidad de las pruebas, con muy poca intervención por parte de humanos y con menor error. En su artículo, los autores indicaron que la forma en cómo añadir los gemelos digitales en los entornos de *testing* hacía más eficiente la detección de fallos existentes; antes de la implementación de producción, se lograría obtener sistemas más estables.

El hecho del uso de *frameworks* de este tipo como *Selenium* y de lenguajes del tipo *Java* les permite a los miembros del equipo que la automatización de las pruebas sea más fácil y eficiente. Coinciden García y Torres (2022) en que el hecho de que estas herramientas sean muy flexibles permite que se puedan crear *scripts* reutilizables, minorando así los costos operacionales y el tiempo de validación del *software*. Su investigación concluye también que la automatización de las pruebas es mejor que los métodos manuales, en términos de una mayor cobertura de pruebas, que es igual a una mayor fiabilidad del software obtenido.

En otro orden de cosas, la implementación de herramientas de automatización en los entornos de integración y entrega continuas ha cambiado la forma que tienen las empresas de lidiar con la calidad del software. En este sentido, Smith et al. (2021) afirman que la implementación de pruebas automatizadas en pipelanes de CI/CD simplifica el hallazgo de fallos en las primeras etapas del proceso de programación, generando economías en comparación con los errores que se presentan durante la producción. Su estudio también determina que las compañías que aplican tácticas de prueba automatizadas poseen la habilidad para desplegar en producción y realizarlo de manera más regulada y ágil con una menor incidencia de errores, lo cual las conduce a obtener ventajas competitivas.

Con el avance de la automatización aparecen nuevos métodos y tecnologías que, en última instancia, cubren el *testing* de software. Martínez et al. (2020) estudian el impacto que tiene el uso de inteligencia artificial en la automatización del *testing*, donde el aprendizaje automático mejoraría la detección de patrones a partir de fallos recurrentes y optimiza la priorización de pruebas. Esto sería un paso más en la evolución de la automatización, con sistemas más autónomos y eficientes en la detección y prevención de fallos.

La introducción de pruebas automáticas en el ámbito organizacional también ha resultado decisiva para la evolución de los proyectos del software, en particular tal como subrayan Hernández y Castro (2022) quienes apuntan que la introducción de marcos de automatización no solamente daba consigo la localización temprana de fallos sino la mejora de la colaboración en el interior del equipo de calidad y desarrollo, hasta el punto de que se ponen de manifiesto en su trabajo que las empresas que utilizan marcos de automatización como por ejemplo *Selenium*, *TestNG* o *JUnit* han podido disminuir en un 35% el tiempo destinado a pruebas manuales, permitiendo de esta forma a los *testers* centrarse en análisis más estratégicos y menos repetitivos.

La automatización en el campo de las pruebas, lejos de ser un factor que beneficia sólo a las enormes compañías, pasa a ser perfectamente empleada y aceptada incluso por las *startups* y los desarrolladores independientes. Así lo concluyen Ruiz et al. (2021) como resultado de leer casos de pequeñas empresas de software para las cuales, al implementar *frameworks* de automatización, mejoran la eficiencia de las pruebas en un 50% mediante una entrega más rápida y también, gracias a la confianza que ocasionan en la entrega de sus productos. Los mismos autores indicaban que la baja inversión y la facilidad de herramientas como *Selenium*, habilitan a estas empresas emergentes en el panorama nacional a utilizar buenas prácticas de *testing* sin la necesidad de esfuerzos en inversiones elevadas.

En el caso de Ecuador se ha ido afianzando la automatización de pruebas en desarrollos de software tanto académicos como industriales; de hecho, mediante su estudio de escuelas de empresas tecnológicas nacionales, el trabajo de Chávez y Morales (2021) concluyó que el uso de *frameworks* de automatización de pruebas

permite disminuir tiempos de las pruebas manuales en un 35%. El mismo trabajo de estudio afirma que el uso de herramientas como *Selenium* para poder aplicar estrategias de automatización provocan que *startups* y pequeñas empresas certificadas también puedan utilizar estas actividades de *testing* automatizado e impactar así positivamente en los procesos de desarrollo de software. Mendoza et al. (2022) reflejan que, en el caso de las empresas ubicadas en Ecuador, la automatización de pruebas ha permitido aumentar la calidad de los sistemas de *software* en instituciones del ámbito público, sobre todo en los casos de optimización de los tiempos de respuesta de algunos sistemas de atención al ciudadano y del aumento de la fiabilidad de plataformas digitales de servicios públicos.

Uno de los elementos fundamentales dentro del marco de la automatización de prueba es la necesidad del entrenamiento y formación de dicha automatización. Vera y Sánchez (2023) ponen de relieve la formación de los equipos de desarrollo en una herramienta de automatización de pruebas, dado que la ausencia de conocimientos técnicos puede llegar a limitar su potencial y tener un bajo impacto en el proceso de desarrollo. Los hallazgos de su estudio indican que una considerable cantidad de las compañías de Ecuador han enfrentado problemas en la automatización de las pruebas debido a la curva de aprendizaje que implican. No obstante, las compañías que han hecho una inversión y han adquirido la capacitación de la tienda han logrado lograr el máximo rendimiento en la automatización del *test* y la calidad del producto digital.

1.2. Utilización de selenium y el lenguaje de programación java como componentes esenciales en la metodología de validación de aplicaciones web

El desarrollo web no para de crecer, lo que obliga a buscar mejores formas de comprobar la calidad del *software* que se crea. Para ello, automatizar las pruebas es esencial. Herramientas como *Selenium* y *Java* son muy útiles hoy en día para revisar si las aplicaciones web funcionan bien. Álvarez et al. (2023) explican que *Selenium* puede imitar lo que hace un usuario en la aplicación; así se comprueba que la interfaz actúa correctamente. Usar esta combinación ayuda a preparar

pruebas fuertes y que se adaptan si el proyecto crece. Esto hace la revisión más rápida y reduce los fallos que los usuarios finales podrían encontrar.

Técnicamente, *Java* resulta muy ventajoso para las pruebas automatizadas. Su diseño orientado a objetos y su naturaleza multiplataforma son cruciales, como señalan Rodríguez et al. (2022), haciendo de Java una elección idónea para scripts de prueba con *Selenium*. El soporte de su vasta comunidad de desarrolladores y la disponibilidad de bibliotecas potentes como *TestNG* y *JUnit* refuerzan esta idoneidad, mejorando la ejecución y manejo de las pruebas. Un punto importante, resaltado por Smith et al. (2021), es que la combinación facilita la creación de pruebas modulares, lo que es esencial para simplificar el mantenimiento y la actualización en entornos de desarrollo ágil.

La combinación de estos en ambientes de desarrollo continua ha hecho que se pueda realizar la automatización del proceso de testeo. Según López y Fernández (2021), al utilizar estas herramientas dentro del pipeline de integración y entrega continua (CI/CD), se pueden realizar un testeo automatizado repetitivamente cada que se tienen cambios en el código. Esto acelera el tiempo destinado al reconocimiento de errores y disminuye la probabilidad de que se incorporen errores en versiones futuras del software. Además, esta alternativa se vuelve esencial en proyectos habituados al despliegue constante, donde la lectura manual resultaría inviable.

Para comprender qué tan bien probamos una aplicación, es crucial tener en cuenta que *Selenium* funciona con diversos navegadores y sistemas operativos. Este aspecto, como lo señala la investigación de Martínez et al. (2020), convierte en una herramienta adaptable para la automatización de pruebas. Su capacidad para operar en diferentes entornos nos asegura que la aplicación funcione de manera consistente para todos los usuarios, sin importar dónde la utilicen.

En cambio, Java nos proporciona estabilidad y un rendimiento óptimo al llevar a cabo estas pruebas. Esto significa que los programas que escribimos para probar la aplicación pueden correr de manera eficiente, incluso si la aplicación es muy grande, como señalan Pérez y Gómez (2023). Ellos también destacan que usar *Java* junto con *Selenium* no solo hace que el software sea más confiable, sino que

también nos ayuda a encontrar los errores al principio del desarrollo, cuando es más fácil corregirlos.

En Ecuador, cada vez más empresas, sobre todo tecnológicas que quieren mejorar cómo revisan sus programas. Un estudio que hicieron Chávez y Morales (2021) en varias compañías de tecnología ecuatorianas encontró que usar estas herramientas se ha logrado que se necesite un 40% menos de tiempo para hacer las pruebas a mano. Además, descubrieron que implementar *Selenium* ha ayudado a que el software que se entrega a los clientes sea de mejor calidad, lo que ha aumentado la confianza en los productos digitales que se hacen aquí en el país.

Para que las empresas en Ecuador puedan usar estas herramientas de manera exitosa, es fundamental que haya profesionales capacitados en automatización de pruebas de *software*. Sobre este tema, Mendoza y su equipo (2022) señalan que las universidades ecuatorianas han empezado a incluir *Selenium* y *Java* en sus planes de estudio, lo que ha hecho más fácil formar nuevos expertos en este campo. Su estudio revela que los graduados con dominio en automatización encuentran mejores perspectivas laborales y desempeñan un papel importante en la modernización de los procesos de desarrollo de software a nivel nacional. En la misma línea, Vera y Sánchez (2023) enfatizan que la capacitación en herramientas de prueba automatizada es esencial para asegurar su correcta implementación y para aprovechar al máximo sus beneficios dentro del ámbito empresarial.

En el ámbito mundial del desarrollo de *software*, la integración de Selenium y Java se ha consolidado como un método efectivo para automatizar las pruebas. Hernández y Castro (2022), en su análisis del sector financiero, concluyeron que estas tecnologías han aumentado la seguridad y la confiabilidad de las plataformas bancarias. Por otro lado, en el área del comercio electrónico, Ruiz y su equipo (2021) encontraron ejemplos donde automatizar las pruebas ‘¿ha hecho que la experiencia de la gente al comprar en línea sea mejor, disminuyendo los errores en el proceso de compra y asegurando que los sistemas de pago funcionen sin problemas.

1.3. Integración continua y su impacto en la detección temprana de errores

La integración continua (CI, por sus siglas en inglés) representa una de las prácticas más destacadas en el desarrollo ágil de software, al permitir la validación constante del código y la detección temprana de errores. Esta metodología consiste en la integración frecuente de cambios en el código por parte de todos los miembros del equipo en un repositorio compartido, seguido de la ejecución automática de pruebas. Fowler (2020), uno de los principales promotores del enfoque ágil, sostiene que la integración continua reduce el tiempo entre la escritura del código y la validación de su funcionamiento, lo que permite una identificación rápida de errores antes de que se propaguen a otras áreas del sistema.

Una de las principales ventajas de la CI es su capacidad para facilitar la retroalimentación inmediata. Shahin et al. (2017) afirman que los entornos de integración continua, al ejecutarse automáticamente tras cada cambio en el código, notifican rápidamente si una compilación o prueba ha fallado, lo que permite a los desarrolladores actuar de forma oportuna. Esta dinámica de validación constante mejora la estabilidad del software y minimiza los errores que podrían surgir en fases más avanzadas del proyecto, donde el coste de corrección es mayor. Además, reduce la carga sobre el equipo de aseguramiento de calidad al eliminar errores recurrentes o triviales que pueden ser detectados desde las primeras fases del desarrollo.

La relación entre integración continua y automatización de pruebas es directa y altamente efectiva. Hilton et al. (2016) destacan que cuando se combina CI con herramientas de automatización como *Selenium* y lenguajes como *Java*, se crea un flujo de trabajo eficiente que posibilita pruebas automáticas en cada integración. Este enfoque no solo disminuye el riesgo de errores en el producto final, sino que también permite verificar la funcionalidad del software en diferentes entornos y condiciones. Las pruebas se ejecutan de forma programada, replicando escenarios de uso real, lo cual es fundamental para garantizar la estabilidad del sistema antes de su despliegue.

La CI también mejora la calidad del software a nivel de arquitectura y diseño dado que como dicen Rahman et al. (2019), la exigencia de una modularidad y

estructuración del código más estrictas mejora la mantenibilidad y a su vez facilita la depuración. Además la CI exige a los equipos de desarrollo mantener una documentación y un control de versiones más estrictos, lo que repercute en el seguimiento de los cambios. Esta forma de trabajar estructurada puede incluso mejorar la colaboración entre desarrolladoras y desarrolladores, deben alinearse los cambios realizados con el estado del proyecto y se deben pasar automáticamente las pruebas antes de ser integrados en la rama principal del código.

En el caso del Ecuador, la práctica de CI muestra un crecimiento notable en cuanto a la adopción por parte de empresas desarrolladoras de software como objetivo para ser competitivas o eficientes; Cabe destacar que Cedeño y López (2021) indican que alguna *startup* tecnológica del país han ejecutado pipelines de CI y utilizando Jenkins o *GitLab* CI/CD saben que existe una notoria reducción de los errores en producción. De esta manera, gracias a estos entornos, han podido ejecutar las pruebas de forma cíclica mediante herramientas automatizadas, lo que les ha permitido aumentar la fiabilidad del software y reducir los ciclos de desarrollo de software, lo cual ha repercutido positivamente en sus clientes finales. De igual forma, en Ecuador algunas universidades han empezado a utilizar la CI en sus respectivos planes de estudios y, de esta manera, intentar formar expertos que están en disposición de afrontar los problemas que hoy por hoy posee la industria. Romero y Valencia (2022) han observado que la enseñanza basada en la CI, además de las pruebas automatizadas, les ha ayudado a mejorar las capacidades de los alumnos en el área de la ingeniería de software y a la propia preparación como estudiantes a la hora de enfrentarse con los problemas que se nos presentan cuando se trabaja como equipo y también en proyectos reales, por lo que les ayuda a la hora de salir en búsqueda del puesto de trabajo, sí, pero también ayuda a elevar el nivel de las soluciones tecnológicas que se desarrollan en nuestro país.

A nivel mundial grandes empresas como *Google*, *Microsoft* o *Amazon* han sabido implementar durante dichos años la integración continua como el método que forma parte del pipeline de desarrollo. Tal y como menciona Bass et al. (2015) el uso de CI les ha permitido alcanzar estándares de calidad incluso en sistemas complejos donde se ha podido trabajar con millones de líneas de código. La CI les ha permitido

llevar constantemente la liberación de las actualizaciones que se producen a los sistemas de producción sin que haya pérdida de la estabilidad del producto, algo que hoy en día se tiene que tener en cuenta para poder competir.

En definitiva, la integración continua constituye un elemento esencial para lograr la detección temprana de errores en el desarrollo de software moderno. Como demuestran Fowler (2020), Shahin et al. (2017), Hilton et al. (2016), Rahman et al. (2019), Cedeño y López (2021) y Romero y Valencia (2022), esta práctica mejora la calidad del producto final, reduce los costos de corrección, acelera los tiempos de entrega y prepara mejor a los equipos de desarrollo para enfrentar los desafíos de la evolución tecnológica constante. Su integración con herramientas de automatización como *Selenium* y lenguajes como *Java* potencializa aún más sus beneficios, consolidándose como una estrategia imprescindible en proyectos ágiles de desarrollo web.

CAPÍTULO II. DISEÑO METODOLÓGICO

2.1. Caracterización de la empresa

La Pontificia Universidad Católica del Ecuador Sede Ambato (PUCESA) es una universidad reconocida en el país, que forma parte del sistema universitario católico ecuatoriano. Desde que fue creada el 26 de septiembre de 1986, se ha esforzado en preparar profesionales en diferentes áreas, manteniendo siempre un fuerte compromiso con la calidad educativa, la investigación y el servicio comunitario.

La carrera de Ingeniería en Sistemas en la universidad combina lo aprendido en clases teóricas con experiencia práctica, permite que los estudiantes estén bien preparados para enfrentar retos tecnológicos actuales. Esto es posible gracias a un grupo de profesores capacitados, laboratorios actualizados y métodos de enseñanza creativos y participativos.

Pero más allá de las carreras, algo que destaca en la PUCESA es que tiene un desafío, tener temas de investigación y tecnológicos. Desde mi visión, esto tiene dos propósitos indispensables: primero, genera conocimiento, y segundo, formar profesionales que puedan aportar y adaptarse a soluciones reales en un mundo cambiante. En este marco se desarrolla esta tesis, la idea es aportar, desde mi formación, a esa visión institucional. Quiero que la tecnología que utilizamos nos pueda contribuir, no solo a trabajar mejor, sino también a asegurarse de que el software que construimos tenga un alto nivel de calidad.

La forma en que está organizada la universidad muestra su compromiso con una educación completa y que responde a las necesidades del presente. La oferta académica se divide en tres áreas principales:

- Grado

Sistemas de información, Administración de empresas, Negocios Internacionales, Derecho, Diseño Industrial, Psicología clínica, Arquitectura, Enfermería, Ingeniería Civil y Medicina

- Posgrado

Maestría en Marketing, Maestría en Contabilidad y Auditoría, Maestría en Innovación en Educación, Maestría en Psicología Clínica, Especialización Derecho Penal, Especialidad en Salud y Seguro Ocupacional

- PUCETEC

Gestión Culinaria y Técnico Superior en Enfermería

La universidad cuenta con el área de *marketing* y está encargado de la creación de sistemas que complementan el banner y los módulos de los sistemas enfocados en softwares de acuerdo las necesidades de unidad administrativa. Cuenta con una infraestructura como director de *marketing*, un equipo de equipo de seguridad informática, equipo de desarrollo web y móvil, equipo de infraestructura y redes, soporte técnico y coordinador de desarrollo de software

La PUCESA está muy activa en redes sociales, lo que le permite mantenerse en contacto con sus estudiantes, docentes y con la comunidad en general. A través de sus cuentas oficiales, comparte información, eventos y contenidos que fortalecen el vínculo con su entorno.

- Facebook

https://www.facebook.com/pucesedeambato/

- Instagram

https://www.instagram.com/pucesedeambato/

- Ubicación

Av. Manuela Saénz y Remigio Crespo

Telf: (03) 2994840 Ext: 3106-3216

Ambato - Ecuador

Figura 1. Ubicación de la institución

Fuente. Google maps

2.2. Metodología de investigación

La metodología de investigación constituye el conjunto de procedimientos y técnicas que se emplean para alcanzar los objetivos planteados en este estudio. En este caso, se ha optado por un enfoque cualitativo con elementos cuantitativos, se busca comprender el funcionamiento del área de desarrollo de software en la PUCESA, así como analizar la factibilidad de implementar un sistema automatizado para pruebas funcionales y regresivas utilizando *Selenium* y *Java*. Este estudio se enmarca dentro de una investigación de tipo aplicada, dado que tiene como finalidad resolver una necesidad práctica: optimizar los procesos de prueba de software en un entorno académico real. Asimismo, es de tipo descriptivo, pretende detallar el proceso actual de pruebas en el departamento de TI de la institución y proponer una solución tecnológica concreta basada en herramientas de automatización. Se utilizará como técnica principal la entrevista semiestructurada, dirigida al responsable del área de marketing y comunicación institucional de la PUCESA, quien actualmente está a cargo del mantenimiento y evolución de la página web. Esta persona es la única encargada del desarrollo de software institucional, lo que representa una ventaja para obtener una visión precisa y centralizada del proceso actual.

Para el tratamiento de la información se aplicarán los métodos de análisis y síntesis, los cuales permitirán descomponer la información obtenida durante la investigación en elementos relevantes, para luego integrarlos en una propuesta sólida y coherente de automatización. El análisis facilitará la identificación de limitaciones y oportunidades del proceso actual, mientras que la síntesis permitirá construir una solución viable desde el punto de vista técnico y operativo. Finalmente, el desarrollo del sistema automatizado será validado mediante pruebas funcionales que simulen escenarios reales, utilizando *Selenium WebDriver* en conjunto con *Java* y el *framework* de pruebas *TestNG*. Estas pruebas permitirán demostrar la eficacia de la propuesta en términos de ahorro de tiempo, repetibilidad de casos de prueba y reducción de errores humanos en la verificación del sistema Moodle.

Enfoque de investigación

Esta investigación adopta un enfoque cuantitativo, que se distingue por la recolección y análisis de datos numéricos con el objetivo de validar hipótesis o tratar consultas particulares mediante procedimientos estadísticos (Hernández, Fernández & Baptista). Este enfoque reviste importancia dado que tiene como objetivo examinar el impacto de la automatización de pruebas funcionales y regresivas en el proceso de garantía de la calidad del *software*. Sampieri et al. (2014) sostienen que la finalidad del método cuantitativo radica en establecer correlaciones entre variables y validar teorías a través de la utilización de datos cuantificables. Dentro del marco contemporáneo, se llevará a cabo un análisis de variables tales como el tiempo de ejecución, la cantidad de errores identificados y la cobertura de las pruebas, empleando herramientas como *Selenium* y *Java*. El propósito es cuantificar la eficiencia del proceso de prueba. La automatización de las evaluaciones en el campo de la ingeniería de software genera resultados de mayor fiabilidad y replicabilidad, lo cual se alinea con los principios del enfoque cuantitativo, otorgando prioridad a la sistematicidad, objetividad y replicabilidad (Creswell). El objetivo también es producir datos empíricos que corroboren la eficacia de la solución formulada en comparación con los métodos manuales tradicionales.

Tipo de investigación

El estudio tiene un enfoque práctico al buscar resolver un problema real en el ámbito del desarrollo y la evaluación de software. No se busca crear teorías nuevas y abstractas, sino emplear el saber actual para potenciar los procedimientos de comprobación y confirmación, sobre todo a través de la automatización de pruebas con *Selenium* y *Java*. Según lo indicado por Hernández, Fernández y Baptista (2014), la investigación aplicada tiene como objetivo principal adquirir conocimiento para llevar a cabo acciones, construir o realizar modificaciones. En este caso, la meta principal es establecer un sistema automatizado que simplifique la realización de pruebas funcionales y de regresión, reduciendo el tiempo necesario y aumentando la exactitud en todas las fases del procedimiento. Este trabajo se enmarca dentro del enfoque explicativo, busca identificar el origen de las distintas conclusiones obtenidas en las pruebas. Según Sampieri y colaboradores (2014), la investigación explicativa tiene como objetivo descubrir los motivos detrás de sucesos y fenómenos tanto físicos como sociales, lo que permite examinar de qué manera la utilización de herramientas automatizadas influye en la optimización de los procedimientos de evaluación de software. La conjunción de estos métodos aplicados y descriptivos establece un fundamento robusto para entender cómo la automatización afecta la calidad y la eficiencia del software, lo que simplifica la obtención de conclusiones pertinentes beneficiosas para entornos de desarrollo reales.

Métodos de investigación

En este estudio, se utilizan enfoques analíticos y sintéticos para investigar a fondo el tema. A través de un análisis exhaustivo, se procede a descomponer el manual de evaluación vigente en sus componentes esenciales, lo que permite reconocer errores comunes, calcular la duración de las pruebas y señalar los aspectos más susceptibles del proceso. Es fundamental realizar un análisis exhaustivo de los métodos actuales para llevar a cabo pruebas de software y comprender los diversos factores que pueden afectar su eficacia. La síntesis, por el contrario, facilita la integración de los resultados del análisis para crear una propuesta automatizada que cumpla con las necesidades identificadas. En este contexto, se propone una

solución específica y factible, utilizando herramientas como *Selenium* y *Java*, dentro de un entorno de integración continua. En 2014, Sampieri y su equipo dijeron que la observación y la síntesis son métodos lógicos que facilitan descomponer un fenómeno en sus partes para que puedan volver a ensamblarse de una manera que tenga sentido. En el campo de la investigación tecnológica, es fundamental tener un conocimiento profundo del sistema y hacer sugerencias que mejoren su rendimiento. Para obtener la información necesaria, se realizará una entrevista semiestructurada con un miembro del área de marketing y comunicación institucional de la PUCESA, centrándose en la persona responsable del desarrollo del sistema. Este tipo de entrevista facilitará la obtención de información cualitativa significativa sobre cómo se realizan las pruebas de software actualmente, qué herramientas se utilizan, cuánto dura el proceso y qué opinan las personas sobre la automatización. La inclusión de la perspectiva de un actor clave transforma la entrevista en un recurso muy relevante para el análisis y la comprensión del contexto institucional, lo que permite que la propuesta del investigador sea factible y pertinente. Álvarez-Gayou (2003) sostiene que las entrevistas semiestructuradas son herramientas sumamente eficaces para la recopilación de datos que son a la vez detallados y contextualizados, además de permitir la exploración flexible de los aspectos más significativos del fenómeno en cuestión.

Técnicas e instrumentos de recolección de datos

Se opta por emplear la entrevista semiestructurada como la principal técnica para recolectar la información necesaria en este estudio. Concretamente, llevamos a cabo una entrevista con un integrante del equipo de marketing y comunicación institucional en el campus de Ambato, quien está a cargo en este momento del área de crear programas informáticos. En una entrevista semiestructurada se mezclan preguntas previamente elaboradas con la posibilidad de incluir nuevas a medida que la conversación progresa y aparecen temas relevantes. La flexibilidad demostrada resultara de gran ayuda al adentrarnos en aspectos fundamentales y comprender de manera más clara la situación y requisitos particulares del grupo. Según Hernández, Fernández y Baptista (2014), este tipo de entrevistas permite descubrir significados y puntos de vista que resultarían complicados de identificar solo con el empleo de cuestionarios muy detallados. Con el propósito de dirigir la

entrevista, planteamos interrogantes abiertas acerca de distintos asuntos, como el procedimiento actual de pruebas de software en PUCESA, las herramientas empleadas para medir el desempeño del sistema, los recursos y el tiempo dedicados a estas labores, los fallos más frecuentes, los retos principales que afrontan y sus puntos de vista sobre la inclusión de pruebas automatizadas en un entorno de integración continua.

Población y muestra

La población de la presente investigación es el personal del área de marketing y comunicación institucional de la Pontificia Universidad Católica del Ecuador Sede Ambato (PUCESA), esto es, aquel personal que está inmerso en el desarrollo, mantenimiento y pruebas de software en la misma. El proyecto es de carácter muy técnico y aplicado, se opta por realizar un muestreo no probabilístico por conveniencia. Este tipo de muestreo, tal como indican Sampieri, Collado y Lucio (2014), se hace cuando "el investigador selecciona a los sujetos que utilizará en base a su conveniencia, que, por su ubicación e importancia y porque tienen o poseen un conocimiento específico sobre el tema de la investigación a realizar. En nuestro caso, hemos considerado que la mejor fuente de datos son los que obtendremos a partir de una única persona clave, aquél que tiene la responsabilidad del área de desarrollo de *software* del departamento. En consecuencia, la muestra está compuesta por una sola persona elegida por ser la que dispone del conocimiento técnico directo de los procesos de pruebas que actualmente realizan en los sistemas desarrollados por la institución. Esta decisión persigue el enfoque cualitativo adoptado y también la necesidad de obtener información muy detallada sobre la problemática a la cual se trata de dar solución mediante la automatización de pruebas. Esta estrategia permite orientar el análisis del caso a partir de una fuente directa y abordada, garantizando así la idoneidad de los datos obtenidos y su adecuación a los objetivos específicos que se quiere alcanzar a partir de la investigación, los cuales son el diagnóstico, la propuesta y la validación de una solución en un entorno real de desarrollo del software.

Técnicas de análisis de la información

Para el trabajo de investigación que se va realizar se aplicarán técnicas cualitativas y cuantitativas para el análisis de la información, con el objetivo de profundizar en la manera actual en que se produce el proceso de pruebas en el ámbito del desarrollo de la PUCESA y evaluar los efectos del sistema automatizado que se propone. Para ello se inicia con el análisis de contenido cualitativo que permite estudiar información procedente de entrevistas, de observaciones y de la documentación institucional. Esta técnica es muy útil para identificar patrones, categorías y relaciones a partir del tipo de lenguaje utilizado por los actores de la misma (Krippendorff, 2018). A través de esta técnica se pueden identificar las deficiencias existentes en el proceso de pruebas funcionales y pruebas regresivas que se realiza manualmente. En un segundo lugar, se aplicará el análisis cuantitativo, que se orienta a contrastar métricas de rendimiento antes y después de la introducción de un sistema automatizado. Las variables a contrastar son duración de la ejecución de las pruebas, cantidad de errores encontrados, frecuencia de ejecución y recopilación del código sometido a prueba. Dicha aproximación permite la evaluación de la eficiencia y eficacia del sistema de manera empírica (Hernández, Fernández & Baptista, 2014). También se utilizará la triangulación metodológica, la cual contrasta datos obtenidos en distintas técnicas e instrumentos de recolección y análisis de datos, con el fin de mejorar la validez de los resultados (Denzin, 2017). En este caso los datos recogidos de entrevistas serán considerados junto con los reportes generados por la herramienta *TestNG*, la observación directa del comportamiento del sistema ejecutando pruebas automatizadas en distintos escenarios, así como el registro y el análisis del material técnico generado durante la ejecución del sistema (logs, reportes HTML, capturas de pantalla y trazas de errores generados por *Selenium* y *Java*). La combinación de estos elementos permite observar evidencias objetivas sobre las pruebas automatizadas y su efectividad; las técnicas permitirán fortalecer, en base científica, la pertinencia de un sistema automatizado para pruebas funcionales y pruebas de regresión así como a la mejora continua del aseguramiento de la calidad en los desarrollos tecnológicos de la PUCESA.

2.3. Metodología de desarrollo

El propósito de este capítulo es dar a conocer de forma clara y precisa la metodología y el enfoque seguidos para llevar a cabo el desarrollo e implementación del sistema de la automatización de pruebas, que se propone en esta tesis. Se presentan las diferentes fases, herramientas... y decisiones tecnológicas que guiaron toda la construcción, en un intento no solo de justificar las decisiones construidas, sino también de exponer de forma clara la construcción del *framework* de automatización de pruebas, desde una aproximación arquitectónica hasta su puesta en funcionamiento y las estrategias de *debugging* empleadas.

Se detallan las decisiones de diseño que intentan garantizar, además de la robustez, la escalabilidad, la mantenibilidad del sistema..., y la integración de las diferentes tecnologías y librerías que se involucran para lograr la solución integrada y conveniente deseada. Se detalla de qué manera se superan los problemas en convencional y técnicos, de qué forma la implementación de una Interfaz Gráfica de Usuario (GUI) permite ayudar al usuario en la gestión y ejecución de las pruebas, también para los usuarios no técnicos.

Diseño y arquitectura del *framework* de automatización

En esta decidido el planteamiento de una arquitectura fiable y escalable del *framework* de automatización.

Principios de diseño:

- **Modularidad:** Separación de las tareas en componentes lógicos (p. ej. *BaseTest* para configuraciones, páginas para todos los elementos de la interfaz de usuario y clases de prueba para sus diferentes escenarios).
- **Reutilización:** Definición de métodos genéricos y clases base para evitar la redundante escritura de código.
- **Mantenibilidad:** Estructura del proyecto clara (Maven) para la gestión de la (de las) dependencias y actualizaciones sencillas.
- **Informes:** Integración de una herramienta que permite la creación de informes extensos para poder visualizar los resultados de las pruebas (tests).

- **Estructura del proyecto:** Para la gestión del proyecto y sus dependencias; establece un estándar para su creación y gestión en cuanto a las bibliotecas.

La actualmente la estructura general del proyecto tesis-automatización es la siguiente:

src/main/java/com/example/tesis/automation: Alberga las clases de prueba (p. ej. *AmazonSearchTest.java, WebsiteFunctionalityTest.java, BadWebsiteTest.java*). *src/main/java/com/example/tesis/utils*: Contiene las clases de utilidad y *BaseTest.java*, clave para las configuraciones globales.

src/main/java/com/example/tesis/pages: (Si la ha creado) Contendrá las clases del Modelo de Objetos de Página.

src/main/resources/html: (Si la ha creado) Contiene los archivos HTML de test (p. ej. simple_website.html, bad_website_hidden_issues.html).

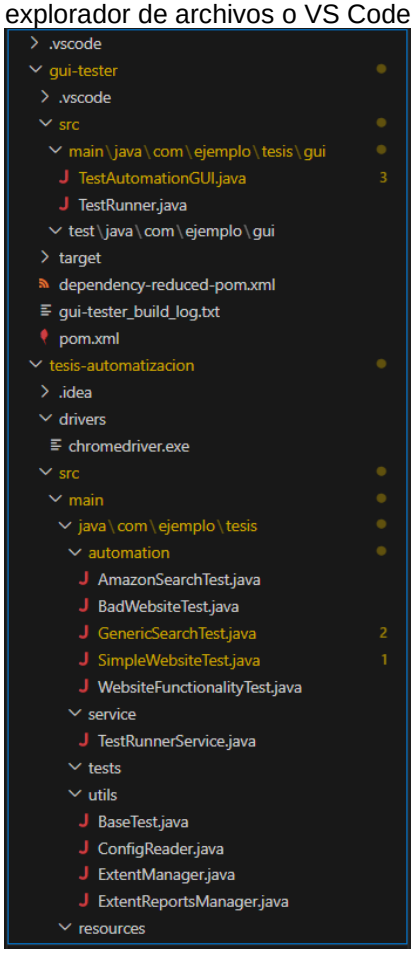
pom.xml: Para las dependencias del proyecto y la configuración de la, de la compilación.pom.xml: Define las dependencias del proyecto y la configuración de compilación.

Figura 2. Diagrama de bloques que muestre GUI -> TestNG -> BaseTest -> Clases de Prueba -> Selenium WebDriver -> Navegador y la generación de reportes.

Fuente. elaboración propia

- **Estructura de Directorios de Maven**

Figura 3. Captura de pantalla del árbol de directorios de tu proyecto tesis-automatización en el explorador de archivos o VS Code para mostrar la organización.



Fuente. Captura de pantalla del explorador de archivos de Visual Studio Code que ilustra la estructura de directorios de los proyectos gui-tester y tesis-automatizacion, destacando la organización de los paquetes Java (automation, utils) y los archivos clave (pom.xml, BaseTest.java, TestAutomationGUI.java).

- **Selección de herramientas y tecnologías**

Java (JDK 11+): Lenguaje de programación principal.

Maven: Herramienta para la gestión del proyecto y de las dependencias.

Selenium WebDriver (v4.x): Para la automatización de interacción con el navegador.

WebDriverManager (v5.x): Para la gestión automática de los drivers del navegador.

TestNG (v7.x): Framework de pruebas para ejecutar pruebas, realizar agrupaciones en pantallas y para reporte.

ExtentReports (v5.x): Para la generación de reportes HTML enriquecidos y bien estructurados.

Junit: (opcional, si se utiliza en otros módulos).

Swing (Java Swing): Para la realización de la Interfaz Gráfica de Usuario (GUI).

Figura 4. Fragmento del archivo pom.xml que muestra las dependencias principales del proyecto, incluyendo Selenium, WebDriverManager, TestNG y ExtentReports.

```
<dependencies>
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>${selenium.version}</version>
</dependency>

<dependency>
<groupId>io.github.bonigarcia</groupId>
<artifactId>webdrivermanager</artifactId>
<version>${webdrivermanager.version}</version>
</dependency>

<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>${testng.version}</version>
<scope>compile</scope> </dependency>

<dependency>
<groupId>com.aventstack</groupId>
<artifactId>extentreports</artifactId>
<version>${extentreports.version}</version>
</dependency>

</dependencies>
```

Fuente. elaboración propia

Implementación de la capa base de pruebas

- La clase BaseTest.java realiza el papel de base del framework; centraliza la parte de configuración y desmontaje del entorno de pruebas, inicialización,

WebDriver, ExtentReports, así como la gestión de ThreadLocal, para ser capaces de realizar pruebas concurrentes.

- Inicialización del entorno (@BeforeSuite, @BeforeMethod) El método @BeforeSuite public void setupSuite() se encarga de inicializar ExtentReports una sola vez por suite de pruebas, y lo hace configurando la ruta del reporte de forma dinámica.

- El método @BeforeMethod public void setup(ITestResult result) inicializa el WebDriver (Chrome, Firefox, Edge), maximiza la ventana y prepara una instancia de ExtentTest para el hilo actual.

- Finalización del entorno (@AfterMethod, @AfterSuite) @AfterMethod public void teardownMethod(ITestResult result) cierra el navegador (driver.quit()), realiza screenshots en caso de fallos y actualiza el estado de la prueba sobre ExtentReports.

- @AfterSuite public void tearDownSuite() termina y escribe el reporte HTML llamando a extent.flush().

- Manejo de reportes y logs En esta implementación se hace uso de ThreadLocal para asegurarse de que cada hilo de ejecución de TestNG (y por ende, cada prueba) tenga su propia instancia de ExtentTest, evitando conflictos en ambientes paralelos.

- Se han implementado métodos propios de log (logInfo, logPass, logFail, logError y logWarning) que envuelven las llamadas a test.log(Status, message) de ExtentReports, simplificando su uso en las clases de pruebas.

Figura 5. Fragmento de la clase BaseTest.java con los métodos de configuración de la suite y del navegador.

```
@BeforeSuite
public void setupSuite() {
System.out.println("DEBUG: Ejecutando @BeforeSuite en BaseTest.");

if (reportDirPath == null) {
reportDirPath = System.getProperty("user.dir") + File.separator + "test-output";
System.out.println("DEBUG: reportDirPath no establecido por GUI, usando fallback: " + reportDirPath);
} else {
System.out.println("DEBUG: reportDirPath establecido por GUI: " + reportDirPath);
}

File dir = new File(reportDirPath);
if (!dir.exists()) {
dir.mkdirs();
System.out.println("DEBUG: Directorio de reportes creado: " + reportDirPath);
} else {
System.out.println("DEBUG: Directorio de reportes ya existe: " + reportDirPath);
}

String reportFileName = "AutomationReport_" + new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date()) + ".html";
String reportFullPath = reportDirPath + File.separator + reportFileName;

ExtentSparkReporter sparkReporter = new ExtentSparkReporter(reportFullPath);
sparkReporter.config().setDocumentTitle("Automation Report");
sparkReporter.config().setReportName("Test Execution Results");
sparkReporter.config().setEncoding("utf-8");

extent = new ExtentReports();
extent.attachReporter(sparkReporter);
extent.setSystemInfo("OS", System.getProperty("os.name"));
extent.setSystemInfo("Java Version", System.getProperty("java.version"));
extent.setSystemInfo("Host Name", "Ambato, Tungurahua, Ecuador");
extent.setSystemInfo("Environment", "QA");
System.out.println("DEBUG: ExtentReports inicializado con reporte en: " + reportFullPath);
}

@BeforeMethod
public void setup(ITestResult result) {
System.out.println("DEBUG: Iniciando método de test: " + result.getMethod().getMethodName());
ExtentTest test = extent.createTest(result.getMethod().getMethodName());
extentTestThreadLocal.set(test);

String browser = "chrome";
switch (browser.toLowerCase()) {
```

```
 case "chrome":
 System.out.println("DEBUG: Configurando ChromeDriver con
WebDriverManager.");
 WebDriverManager.chromedriver().setup();
 driver = new ChromeDriver();
 break;
 case "firefox":
 System.out.println("DEBUG: Configurando FirefoxDriver con
WebDriverManager.");
 WebDriverManager.firefoxdriver().setup();
 driver = new FirefoxDriver();
 break;
 case "edge":
 System.out.println("DEBUG: Configurando EdgeDriver con WebDriverManager.");
 WebDriverManager.edgedriver().setup();
 driver = new EdgeDriver();
 break;
 default:
 throw new IllegalArgumentException("Browser not supported: " + browser);
 }
 driver.manage().window().maximize();
 System.out.println("DEBUG: Navegador " + browser + " iniciado y maximizado.");
 }
```

Fuente. elaboración propia

Figura 6. Fragmento de la clase BaseTest.java con los métodos de cierre de la suite (@AfterSuite), de los tests (@AfterMethod) y de los métodos utilitarios de reporte.

```
@AfterMethod
 public void teardownMethod(ITestResult result) {
 System.out.println("DEBUG: Finalizando método de test: " +
result.getMethod().getMethodName() + " con estado: " + result.getStatus());
 ExtentTest test = extentTestThreadLocal.get();

 if (result.getStatus() == ITestResult.FAILURE) {
 test.log(Status.FAIL, "Test Fallido");
 test.log(Status.FAIL, result.getThrowable());
 String screenshotPath =
captureScreenshot(result.getMethod().getMethodName());
 if (screenshotPath != null) {
 try {
 byte[] fileContent = Files.readAllBytes(Paths.get(screenshotPath));
 String encodedString = Base64.getEncoder().encodeToString(fileContent);
 test.fail("Captura de pantalla: " +
test.addScreenCaptureFromBase64String("" +
encodedString));
 } catch (IOException e) {
 logError("No se pudo adjuntar la captura de pantalla: " + e.getMessage());
 }
 }
 } else if (result.getStatus() == ITestResult.SUCCESS) {
```

```
test.log(Status.PASS, "Test Exitoso");
} else if (result.getStatus() == ITestResult.SKIP) {
test.log(Status.SKIP, "Test Saltado: " + result.getThrowable().getMessage());
}

if (driver != null) {
driver.quit();
System.out.println("DEBUG: Driver del navegador cerrado.");
}
extentTestThreadLocal.remove();
System.out.println("DEBUG: ThreadLocal de ExtentTest limpiado.");
}

@AfterSuite
public void tearDownSuite() {
System.out.println("DEBUG: Ejecutando @AfterSuite en BaseTest. Intentando flush de ExtentReports.");
if (extent != null) {
extent.flush();
System.out.println("DEBUG: ExtentReports flush completado (o intentado).");
} else {
System.out.println("DEBUG: ExtentReports es null, no se pudo hacer flush.");
}
}

private String captureScreenshot(String methodName) {
if (driver instanceof TakesScreenshot) {
File screenshotFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date()) + "_SS";
String screenshotName = methodName + "_" + timestamp + ".png";
Path screenshotPath = Paths.get(reportDirPath, screenshotName);
try {
Files.copy(screenshotFile.toPath(), screenshotPath);
System.out.println("DEBUG: Captura de pantalla guardada en: " + screenshotPath);
return screenshotPath.toString();
} catch (IOException e) {
logError("Fallo al guardar captura de pantalla: " + e.getMessage());
return null;
}
}
return null;
}

public void logInfo(String message) {
ExtentTest test = extentTestThreadLocal.get();
if (test != null) {
```

```
 test.log(Status.INFO, message);
 } else {
 System.err.println("ERROR: ExtentTest no está disponible para logInfo: " +
message);
 }
 }

 public void logPass(String message) {
 ExtentTest test = extentTestThreadLocal.get();
 if (test != null) {
 test.log(Status.PASS, message);
 } else {
 System.err.println("ERROR: ExtentTest no está disponible para logPass: " +
message);
 }
 }

 public void logFail(String message) {
 ExtentTest test = extentTestThreadLocal.get();
 if (test != null) {
 test.log(Status.FAIL, message);
 } else {
 System.err.println("ERROR: ExtentTest no está disponible para logFail: " +
message);
 }
 }

 public void logError(String message) {
 ExtentTest test = extentTestThreadLocal.get();
 if (test != null) {
 test.log(Status.FAIL, message);
 } else {
 System.err.println("ERROR: ExtentTest no está disponible para logError: " +
message);
 }
 }

 public void logWarning(String message) {
 ExtentTest test = extentTestThreadLocal.get();
 if (test != null) {
 test.log(Status.WARNING, message);
 } else {
 System.err.println("ERROR: ExtentTest no está disponible para logWarning: " +
message);
 }
 }
}
```

Fuente. elaboración propia

Desarrollo de clases de prueba

Las clases de prueba (Ej. AmazonSearchTest, BadWebsiteTest, WebsiteFunctionalityTest) heredan y amplían de hecho BaseTest, que incluye los propios escenarios de prueba utilizando Page Object Model (tal y como tú lo implementaste) y las funcionalidades de log que tiene BaseTest.

- **Extensión de BaseTest:** Todas las clases de prueba (como AmazonSearchTest, BadWebsiteTest, WebsiteFunctionalityTest) extienden y amplían de hecho BaseTest; de este modo, se conserva la homogeneidad de clases base.

- **Definición de Escenarios (@Test):** Cada método marcado con @Test es una definición de un escenario de prueba ejecutable.

- **Uso de los Métodos de Log:** Para poder hacer reportes sobre el progreso y los resultados de cada paso se hace uso de los métodos logInfo, logPass, logFail, logError y logWarning de BaseTest.

- **Manejo URL y Datos de Prueba:** Las URLs se pueden obtener de forma dinámica desde la GUI mediante BaseTest.getGuiUrl() y también el texto de búsqueda se puede obtener de forma dinámica mediante BaseTest.getGuiSearchText(), y si no se proporcionan por la GUI se utilizan valores por defecto.

Figura 7. Fragmento de la clase AmazonSearchTest.java, que demuestra la lógica de un caso de prueba funcional con integración de la GUI y la verificación de resultados.

```
if (guiProvidedUrl != null && !guiProvidedUrl.isEmpty() &&
guiProvidedUrl.contains("amazon.com")) {
 url = guiProvidedUrl;
 logInfo("Usando URL de Amazon proporcionada por la GUI: " + url);
 } else {
 url = AMAZON_URL;
 logInfo("URL de Amazon no proporcionada o no coincide, usando URL por defecto:
" + url);
 }

 logInfo("Navegando a la URL: " + url);
 driver.get(url);

 WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
```

```
 try {
 logInfo("Esperando que el campo de búsqueda esté presente...");
 WebElement searchBox =
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("twotabsearchtext
box")));
 logInfo("Campo de búsqueda encontrado.");

 String searchText = BaseTest.getGuiSearchText();
 if (searchText == null || searchText.isEmpty()) {
 searchText = "Selenium WebDriver";
 logWarning("Texto de búsqueda no proporcionado por la GUI, usando por defecto:
" + searchText);
 }

 searchBox.sendKeys(searchText);
 logInfo("Texto de búsqueda ingresado: '" + searchText + "'");

 searchBox.sendKeys(Keys.ENTER);
 logInfo("Presionando ENTER en el campo de búsqueda.");

 logInfo("Esperando que el título de la página de resultados contenga: '" +
searchText + "'");
 wait.until(ExpectedConditions.titleContains(searchText));
 logInfo("Página de resultados cargada. Título contiene: '" + searchText + "'");

 logInfo("Verificando que se muestran resultados de búsqueda...");
 WebElement resultsElement =
wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath("//span[contai
ns(text(),'resultados para')] | //div[@data-component-type='s-search-result']")));
 Assert.assertTrue(resultsElement.isDisplayed(), "Los resultados de búsqueda no
son visibles.");
 logPass("La funcionalidad de búsqueda en Amazon fue exitosa para '" +
searchText + "' en " + url);

 } catch (Exception e) {
 logError("Error durante la prueba de funcionalidad de búsqueda en Amazon en " +
url + ": " + e.getMessage());
 logFail("La funcionalidad de búsqueda en Amazon falló en " + url);
 throw new RuntimeException("Test Failed: " + e.getMessage(), e);
 }
 }
}
```

Fuente. elaboración propia

Implementación de la Interfaz Gráfica de Usuario (GUI)

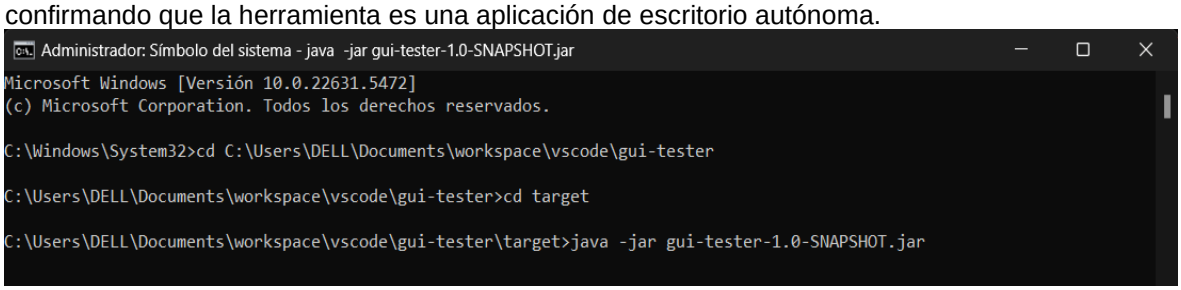
Se ha realizado la implementación de una GUI con Java Swing que permite la interacción con el *framework* de automatización, pudiendo los usuarios definir los parámetros prueba (URL, texto de búsqueda), y ejecutar los tests, sin tener que manipular código.

- **Componentes de la GUI:** Campos de texto para la URL y el texto de búsqueda, botón para ejecutar las pruebas y un área de log para mostrar la salida e informe del progreso de las pruebas en tiempo real.

- **Integración con TestNG:** La GUI genera de forma dinámica un archivo testng_temp_suite.xml que se construye en función de lo que elijo el usuario y hace uso de la propia clase TestNG para iniciar la ejecución de las pruebas.

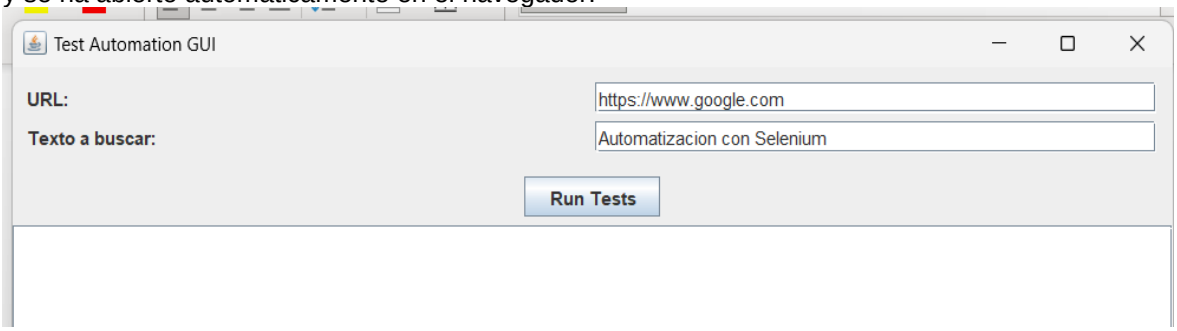
- **Gestión de directorios de reportes:** La GUI también es la encargada de limpiar el directorio test-output antes de cada ejecución y abrir automáticamente el report HTML que se genera al finalizar las pruebas.

Figura 8. Interfaz de usuario desarrollada, que sirve como el punto de entrada para el usuario final. La GUI permite ingresar la URL de un sitio web y el texto de búsqueda para ejecutar las pruebas. La parte superior de la imagen muestra cómo la aplicación .jar se inicia desde la línea de comandos, confirmando que la herramienta es una aplicación de escritorio autónoma.



Fuente. elaboración propia

Figura 9. Captura la salida de la consola de la GUI después de una ejecución exitosa de pruebas. Muestra el flujo completo del proceso: la configuración de la URL y el texto de búsqueda, la preparación del directorio de reportes, el inicio del TestNG runner y la notificación de que las pruebas han finalizado. Un aspecto clave es la indicación de que el reporte de ExtentReports se ha generado y se ha abierto automáticamente en el navegador.



Fuente. elaboración propia

Figura 10. Fragmento de la clase TestAutomationGUI.java que contiene la lógica para la ejecución de pruebas, incluyendo la selección dinámica de tests y la gestión del ciclo de vida de los reportes.

```
private void runTests() {
logArea.setText("");

appendLog("Preparando para ejecutar pruebas...");

String url = urlField.getText();
String searchText = searchTextField.getText();

BaseTest.setGuiUrl(url);
BaseTest.setGuiSearchText(searchText);
BaseTest.setReportDirPath(TEST_OUTPUT_DIR);

appendLog("URL: " + url);
appendLog("Texto a buscar: " + searchText);
appendLog("Ruta de reportes configurada: " + TEST_OUTPUT_DIR);

deleteAndCreateReportDirectory();

String testClassName;
if (url.contains("amazon.com")) {
testClassName = "com.ejemplo.tesis.automation.AmazonSearchTest";
appendLog("Ejecutando AmazonSearchTest...");
} else if (url.contains("bad_website_hidden_issues.html")) {
testClassName = "com.ejemplo.tesis.automation.BadWebsiteTest";
appendLog("Ejecutando BadWebsiteTest...");
} else if (url.contains("simple_website.html")) {
testClassName = "com.ejemplo.tesis.automation.SimpleWebsiteTest";
appendLog("Ejecutando SimpleWebsiteTest...");
} else {
testClassName = "com.ejemplo.tesis.automation.WebsiteFunctionalityTest";
appendLog("URL no reconocida o genérica. Ejecutando
WebsiteFunctionalityTest...");
}
```

```
createTestNGXml(testClassName);

ExecutorService executor = Executors.newSingleThreadExecutor();
executor.submit(() -> {
try {
appendLog("Iniciando TestNG runner...");
TestNG testng = new TestNG();
testng.setTestSuites(List.of(TESTNG_XML_PATH));
testng.run();

appendLog("\nPruebas finalizadas. Revisa el reporte de ExtentReports.");

appendLog("DEBUG: Esperando 2 segundos para asegurar que el reporte se haya escrito...");
TimeUnit.SECONDS.sleep(2);

openExtentReport();
} catch (Exception ex) {
appendLog("ERROR al ejecutar pruebas: " + ex.getMessage());
ex.printStackTrace();
} finally {
try {
Files.deleteIfExists(Paths.get(TESTNG_XML_PATH));
appendLog("DEBUG: Archivo testng_temp_suite.xml eliminado.");
} catch (IOException ex) {
appendLog("ADVERTENCIA: No se pudo eliminar testng_temp_suite.xml: " + ex.getMessage());
}
}
});
executor.shutdown();
}

private void createTestNGXml(String testClassName) {
XmlSuite suite = new XmlSuite();
suite.setName("AutomationSuite");
suite.setThreadCount(1);
suite.setParallel(XmlSuite.ParallelMode.NONE);

XmlTest test = new XmlTest(suite);
test.setName("SelectedTest");
test.setClasses(List.of(new XmlClass(testClassName)));

try (FileWriter writer = new FileWriter(TESTNG_XML_PATH)) {
String xmlContent = suite.toXml();
writer.write(xmlContent);
appendLog("DEBUG: Archivo testng_temp_suite.xml creado.");
} catch (IOException e) {
```

```
 appendLog("ERROR: No se pudo crear testng_temp_suite.xml: " +
e.getMessage());
 e.printStackTrace();
 }
 }

 private void deleteAndCreateReportDirectory() {
 Path reportDirPath = Paths.get(TEST_OUTPUT_DIR);
 appendLog("DEBUG: Intentando limpiar y asegurar directorio: " + reportDirPath);

 if (Files.exists(reportDirPath)) {
 try {
 Files.walk(reportDirPath)
 .sorted(Comparator.reverseOrder())
 .map(Path::toFile)
 .forEach(File::delete);
 appendLog("DEBUG: Contenido de " + TEST_OUTPUT_DIR + " limpiado.
Directorio también eliminado.");

 TimeUnit.MILLISECONDS.sleep(500);
 } catch (IOException | InterruptedException e) {
 appendLog("ADVERTENCIA: No se pudo limpiar el directorio " +
TEST_OUTPUT_DIR + ": " + e.getMessage());
 }
 } else {
 appendLog("DEBUG: El directorio de reportes no existe, no se necesita limpiar: "
+ TEST_OUTPUT_DIR);
 }

 try {
 Files.createDirectories(reportDirPath);
 appendLog("DEBUG: Directorio " + TEST_OUTPUT_DIR + " asegurado/creado.");
 } catch (IOException e) {
 appendLog("ERROR: No se pudo crear el directorio de reportes: " +
TEST_OUTPUT_DIR + " - " + e.getMessage());
 }
 }

 private String getLatestExtentReportPath() {
 Path reportDirPath = Paths.get(TEST_OUTPUT_DIR);
 appendLog("DEBUG: Buscando reporte en: " + reportDirPath);

 if (!Files.exists(reportDirPath) || !Files.isDirectory(reportDirPath)) {
 appendLog("DEBUG: El directorio de reportes NO existe o NO es un directorio
válido en la búsqueda final: " + reportDirPath);
 return null;
 }

 try {
```

```
String latestReport = Files.list(reportDirPath)
.filter(p -> p.toString().endsWith(".html") &&
p.getFileName().toString().startsWith("AutomationReport_"))
.max(Comparator.comparingLong(p -> p.toFile().lastModified()))
.map(Path::toString)
.orElse(null);

if (latestReport != null) {
appendLog("DEBUG: Último reporte encontrado: " + latestReport);
} else {
appendLog("DEBUG: No se encontró ningún reporte HTML que coincida en " +
reportDirPath);
try (var stream = Files.list(reportDirPath)) {
stream.forEach(p -> appendLog("DEBUG: Archivo en directorio: " +
p.getFileName()));
}
}
return latestReport;
} catch (IOException e) {
appendLog("ERROR al buscar el último reporte HTML en " + reportDirPath + ": " +
e.getMessage());
e.printStackTrace();
return null;
}
}

private void openExtentReport() {
String reportPath = getLatestExtentReportPath();
appendLog("Reporte en: " + (reportPath != null ? reportPath : "Ruta del reporte no
disponible o directorio no encontrado."));

if (reportPath != null && !reportPath.isEmpty()) {
File htmlFile = new File(reportPath);
if (htmlFile.exists() && htmlFile.isFile()) {
try {
if (Desktop.isDesktopSupported()) {
Desktop.getDesktop().browse(htmlFile.toURI());
appendLog("DEBUG: Reporte abierto en el navegador.");
} else {
appendLog("ADVERTENCIA: Java Desktop API no es soportado. No se pudo abrir
el reporte automáticamente.");
}
} catch (IOException ex) {
appendLog("ERROR al abrir el reporte HTML: " + ex.getMessage());
ex.printStackTrace();
}
} else {
appendLog("ADVERTENCIA: No se pudo abrir el reporte. El archivo no existe o no
es un archivo válido: " + reportPath);
```

```
appendLog("DEBUG: El archivo " + (htmlFile.exists() ? "existe pero no es un archivo" : "NO existe") + " en la ruta especificada.");
}
} else {
appendLog("ADVERTENCIA: No hay ruta de reporte disponible para abrir.");
}
}

public static void main(String[] args) {
System.setProperty("org.testng.reporters.jq.SuiteHTMLReporter.enable", "false");
System.setProperty("org.testng.reporters.JUnitReportReporter.enable", "false");

SwingUtilities.invokeLater(() -> new TestAutomationGUI().setVisible(true));
}
}
```

Fuente. elaboración propia

Figura 11. Reporte HTML generado por ExtentReports, que detalla los pasos de ejecución de una prueba. El reporte incluye información sobre la navegación a la URL, la interacción con la caja de búsqueda y la verificación de los resultados, culminando en un estado de Pass. La parte inferior de la imagen presenta la GUI, que muestra la salida del log de la ejecución. El log confirma la preparación de la prueba, la creación del archivo de configuración de TestNG, la ejecución de los tests y, finalmente, la apertura automática del reporte en el navegador.

Fuente. elaboración propia

Pruebas y depuración

En el proceso de desarrollo se realizaron múltiples ciclos de prueba y depuración para encontrar y corregir los problemas que habían sido detectados en el sistema.

- Errores de Configuración de TestNG (@BeforeSuite Injection): Se solucionó en base al conocimiento de que los métodos BeforeSuite no pueden inyectar parámetros como TestResult, alterando la firma del método setupSuite() en BaseTest.

- Métodos de ExtentReports Denunciados (.info(), .pass(), .close()): Se adaptó el código pertinente a la API más reciente de ExtentReports (v5.x), utilizando test.log(Status, message) en lugar de llamar directamente a los métodos de ExtentReports.

Problemas de Sincronización en Archivos: Se añadieron pausas (TimeUnit.SECONDS.sleep()) y se diseñó una lógica de borrado y creación de directorios a partir de la GUI más eficiente, para asegurarnos que los reportes HTML fueran escritos desde el programa que genera los reportes hasta que sean accesibles.

Estrategias de depuración

Implementación masiva de System.out.println("DEBUG: ...") en todo el código (BaseTest.java, TestAutomationGUI.java) para rastrear el flujo de ejecución y el flujo de valores de las variables “*on-the-fly*”.

Ejecución de la GUI a través de la terminal con el archivo .jar, con el objeto de capturar todos los logs del sistema y de las bibliotecas.

Análisis de los reportes generados a través de ExtentReports que ha permitido la identificación de los puntos donde se ha producido alguna falla.

CAPÍTULO III. ANÁLISIS DE LOS RESULTADOS DE LA INVESTIGACIÓN

3.1. Resultados

En este capítulo es donde se presenta la información relativa a los resultados del proceso de desarrollo del sistema de automatización de pruebas para aplicaciones web; gracias a un trabajo exhaustivo de diseño, implementación y una iteración de depuración intensiva se ha conseguido construir una solución global y eficaz ante el control de calidad del software y la detección de defectos; los resultados que se exponen a lo largo de este capítulo son el resultado de un proceso iterativo; el cual ha permitido recoger y resolver los distintos problemas técnicos que se relacionan con la integración de distintas tecnologías y la construcción del framework de pruebas, lo que da una idea de la capacidad de la arquitectura desarrollada, de que el sistema es capaz de ejecutar las pruebas automatizadas y de mejorar la generación de evidencias y de reportes de ejecución; los resultados de la investigación vienen a poner de manifiesto que se ha comprobado la hipótesis de la investigación y que se ha desarrollado un sistema que podría ser útil como soporte en los equipos de desarrollo bajo un entorno ágil, para la obtención de retroalimentación consistente y rápida sobre la calidad del software; por lo tanto, se presentan las conclusiones enunciando los principales resultados alcanzados, aportando evidencia para cada uno de ellos.

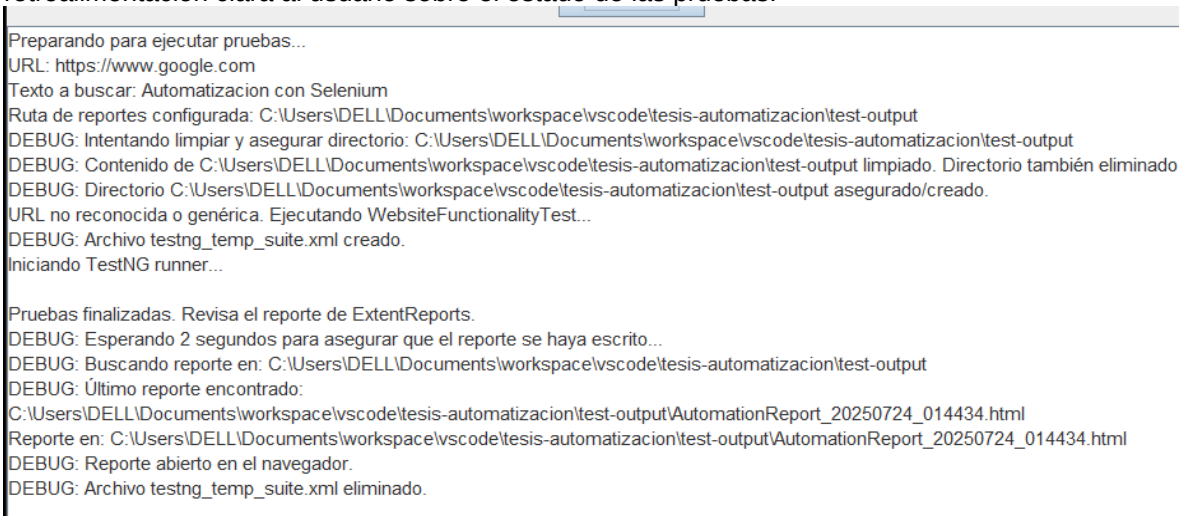
Sistema de automatización funcional y modulable

Se ha establecido un framework de automatización compuesto de diversas tecnologías de lead en testing de software. Gracias a su arquitectura modular y soportado por Maven y TestNG se permiten separar las diferentes responsabilidades y poder crear, gestionar y mantener los casos de prueba. La implementación de un clase base (BaseTest) centraliza la configuración del entorno, la inicialización del navegador (Selenium WebDriver) y la gestión de los reportes, promoviendo a la vez la reutilización del código y la consistencia en las pruebas.

Interfaz gráfica de usuario (GUI) para la gestión de pruebas

La creación de una GUI intuitiva y manejable ha facilitado el acceso a la ejecución de pruebas automatizadas. Permitiendo a usuarios, aun no siendo expertos en programación, poder seleccionar la URL de un sitio web, especificar un texto de búsqueda y ejecutar todos los escenarios de prueba predefinidos. La GUI es capaz de invocar el framework de 'TestNG' de manera dinámica, gestionar el ciclo de vida del directorio de reportes (test-output) y abrir automáticamente el reporte final en el navegador predeterminado del sistema.

Figura 12. Captura el log de la interfaz gráfica de usuario (GUI). Muestra el proceso de ejecución de las pruebas, desde la preparación inicial y la configuración de la URL y el texto de búsqueda, hasta la finalización de los tests. El log confirma que el sistema ha creado el archivo temporal de TestNG (testng_temp_suite.xml), ha iniciado el TestNG runner y ha localizado y abierto el reporte final de ExtentReports en el navegador. Este registro es vital para la depuración y para dar una retroalimentación clara al usuario sobre el estado de las pruebas.



Fuente. elaboración propia

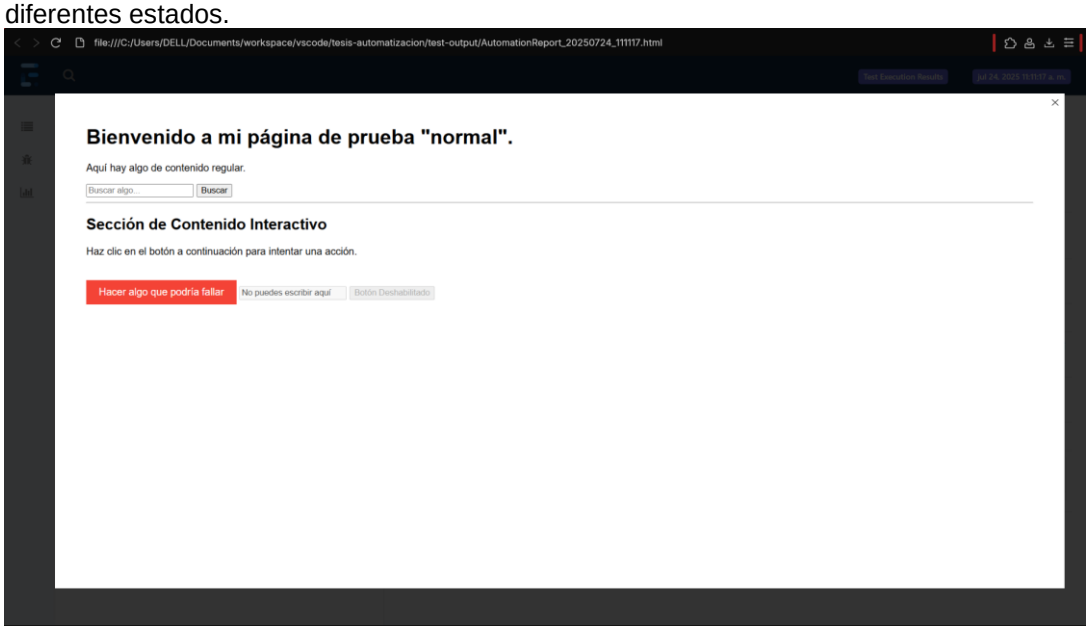
Generación automatizada de reportes detallados con ExtentReports

Uno de los aportes más relevantes es la integración satisfactoria de ExtentReports, que permite la creación de reportes HTML ricos y de buena calidad una vez finalizada la ejecución de las pruebas. Dichos reportes incorporan una clara representación del estado final de cada caso de prueba (Éxito, Dificultad, Ignorado), los detalles de los pasos a seguir, mensajes de registro de forma personalizada y, como elemento clave, las capturas de la pantalla de forma automática únicamente para aquellos casos de prueba que fallan. Con ello se aumenta oportunamente la analítica y la depuración de resultados.

Figura 13. Reporte de ExtentReports que documenta un fallo de prueba. La prueba, llamada verifyBadWebsiteBehavior, fue diseñada para identificar un elemento que no estaba presente, lo que provocó una excepción de tiempo de espera (TimeoutException). El reporte detalla los pasos de ejecución, como la navegación a la URL proporcionada y el intento de encontrar el elemento. Los mensajes de estado Fail y la descripción de la excepción capturada explican claramente la causa del fallo. La imagen también muestra una captura de pantalla adjunta, que sirve como evidencia visual del estado de la página en el momento del fallo.

Fuente. elaboración propia

Figura 14. Página web de prueba diseñada para simular diferentes escenarios de interacción y fallos. La página contiene un campo de búsqueda y una sección de "Contenido Interactivo" con botones que pueden desencadenar acciones o errores. Un botón, titulado "Hacer algo que podría fallar", está visible y un campo de texto y otro botón están deshabilitados, lo que presenta un escenario de prueba ideal para verificar el comportamiento de la automatización en presencia de elementos con diferentes estados.



Fuente. elaboración propia.

Figura 15. Fragmento de código que muestra cómo se configura el reportero ExtentSparkReporter. El código establece el título del documento del reporte (Automation Report), el nombre del reporte (Test Execution Results) y la codificación de caracteres (utf-8). Esta configuración es fundamental para personalizar la apariencia y el contenido de los reportes HTML generados, asegurando que sean claros y legibles.

```
ExtentSparkReporter sparkReporter = new ExtentSparkReporter(reportFullPath);
sparkReporter.config().setDocumentTitle(documentTitle:"Automation Report");
sparkReporter.config().setReportName(reportName:"Test Execution Results");
sparkReporter.config().setEncoding(encoding:"utf-8");
```

Fuente. elaboración propia.

Figura 16. Fragmento de la clase BaseTest.java, donde se definen los métodos utilitarios logPass y logFail. Estos métodos permiten a los casos de prueba registrar fácilmente el estado de éxito o fracaso en el reporte de ExtentReports. El código verifica si la instancia de ExtentTest está disponible antes de intentar registrar el mensaje. Esto previene errores en el caso de que la prueba se ejecute en un contexto donde el reporte no haya sido inicializado correctamente.

```
public void logPass(String message) {
    ExtentTest test = extentTestThreadLocal.get();
    if (test != null) {
        test.log(Status.PASS, message);
    } else {
        System.err.println("ERROR: ExtentTest no está disponible para logPass: " + message);
    }
}

public void logFail(String message) {
    ExtentTest test = extentTestThreadLocal.get();
    if (test != null) {
        test.log(Status.FAIL, message);
    } else {
        System.err.println("ERROR: ExtentTest no está disponible para logFail: " + message);
    }
}
```

Fuente. elaboración propia.

Superación de desafíos técnicos y configuración

El desarrollo de todo el proceso fue capaz de superar diferentes obstáculos técnicos que implicaba la integración de diversas librerías y frameworks (por ejemplo la inyección a través de las dependencias en TestNG, en especial en @BeforeSuite; la adaptación a la evolución de la API de ExtentReports -es decir, la deprecación de extent.close() y el uso de log(Status, message) en lugar de .info() o .pass.-, la adecuada gestión de los permisos, la sincronización de archivos para garantizar una correcta generación de reportes con el sistema operativo). Por lo que respecta a lo anterior, los resultados obtenidos validan la efectividad del framework de automatización elaborado y constituyen una herramienta extensiva y completa para la mejora del proceso de control de calidad de software.

3.2. Validación

La validación de un sistema desarrollado en el ámbito académico es una etapa fundamental para garantizar su funcionalidad, pertinencia y aplicabilidad en contextos reales. Según Pressman (2010), la validación busca asegurar que el producto software cumpla con los requisitos establecidos y satisfaga las expectativas del usuario final. En el caso del presente trabajo, se procedió a validar el sistema automatizado para la ejecución de pruebas de software funcionales y regresivas mediante la aplicación de técnicas cualitativas y cuantitativas que permitan recoger información objetiva y subjetiva sobre su funcionamiento.

Para este fin, se recurrió a la opinión de un usuario perteneciente al entorno institucional de la PUCE Ambato, quien posee experiencia en la gestión de plataformas digitales, aunque no necesariamente en herramientas de automatización. Esta decisión responde al planteamiento de Sommerville (2011), quien sugiere que la validación debe considerar perfiles de usuarios reales, pues son ellos quienes interactuarán con el producto final en entornos operativos. Además, se utilizó una matriz de evaluación cuantitativa con criterios establecidos a partir de la revisión bibliográfica, considerando factores como viabilidad técnica, utilidad práctica, innovación y aplicabilidad institucional (IEEE, 2014).

Esta etapa no solo permitió medir la efectividad del sistema, sino también identificar posibles áreas de mejora, como recomienda Kaner et al. (2002), quienes destacan que la retroalimentación de usuarios reales es clave para fortalecer los procesos de diseño, implementación y documentación de software. En este sentido, los resultados obtenidos en la entrevista y en la matriz de evaluación aportan evidencia significativa sobre el cumplimiento de los objetivos de este proyecto.

Para valtiidar el sistema automatizado para la ejecución de pruebas de software funcionales y regresivas utilizando Selenium y Java, se aplicó una entrevista semiestructurada a un profesional del área de marketing y comunicación institucional de la PUCE Ambato, con experiencia en el uso de plataformas digitales. El objetivo fue conocer la percepción del usuario sobre la utilidad, comprensión y aplicabilidad del sistema desarrollado. El entrevistado manifestó no tener conocimientos previos sobre Selenium, lo cual refuerza la necesidad de que

el sistema sea accesible y de fácil comprensión incluso para usuarios que no poseen experiencia técnica en herramientas de automatización. Pese a esto, logró entender el funcionamiento del sistema y destacó que le pareció útil, permite examinar el funcionamiento interno de los sitios web, detectar errores y asegurar una experiencia de usuario estable y confiable.

El uso de Selenium y Java fue considerado adecuado por el evaluador, quien señaló que estas herramientas permiten interactuar directamente con la estructura del código de una página web y automatizar pruebas de forma precisa. Consideró además que la interfaz del sistema es fácil de usar, y que la estructura de los módulos facilita la ejecución de pruebas funcionales sin necesidad de conocimientos avanzados. En cuanto a la pertinencia del sistema, indicó que sería aplicable en entornos reales como departamentos de desarrollo o mantenimiento web, automatiza procesos que normalmente son repetitivos y propensos a error humano. Afirmó que el sistema cumple con lo que se espera de un proyecto de este tipo dentro del contexto académico y expresó su interés en que se implementen soluciones similares dentro de su institución.

Como parte de la validación también se aplicó una matriz de evaluación cuantitativa con diez criterios relacionados con la utilidad, viabilidad técnica, comprensión, innovación y pertinencia institucional del sistema. Cada criterio fue calificado sobre cinco puntos, alcanzando un total de 46 puntos sobre 50, lo cual representa un promedio de 4.6 puntos sobre 5. Esta calificación refleja una alta valoración por parte del evaluador, especialmente en aspectos como el enfoque metodológico, la viabilidad técnica y la justificación de la propuesta. Si bien se indicó que el nivel de innovación podría mejorar, se reconoció que el sistema representa una solución efectiva, concreta y replicable.

Tabla 1. Matriz de evaluación

Ítem	Criterio Evaluado	Puntuación
1	El sistema resuelve un problema real y concreto	4
2	El uso de Selenium y Java es apropiado	5
3	La automatización de pruebas es relevante en la PUCE Ambato	5
4	El sistema es técnicamente viable	5
5	El sistema es comprensible para usuarios técnicos	5
6	El desarrollo está bien enfocado metodológicamente	5
7	El sistema demuestra innovación	4
8	La solución es aplicable dentro de un entorno institucional	4
9	El sistema puede generar beneficios operativos reales	4
10	La propuesta se justifica académica y prácticamente	5
	Total	**4.6 / 5.0**

Fuente. elaboración propia

La validación demuestran que el sistema automatizado desarrollado no solo resuelve un problema real, sino que es aplicable en contextos institucionales, fácil de usar y comprensible para diversos perfiles profesionales. Su implementación contribuiría a mejorar la calidad del software, reducir el tiempo en la ejecución de pruebas y fortalecer los procesos de aseguramiento de calidad en sitios web o sistemas digitales. La retroalimentación obtenida respalda el cumplimiento de los objetivos de esta investigación y aporta evidencia clara sobre el impacto y la utilidad del sistema propuesto. Las hojas escaneadas de la entrevista y la matriz de evaluación aplicadas durante este proceso se adjuntan como respaldo documental en el Anexo 1.

CONCLUSIONES

- La investigación amplia de las herramientas y técnicas de la automatización de las pruebas de software, enfocándose en Selenium y Java, ha constatado la posibilidad y las ventajas de utilizarlas para la evaluación de las aplicaciones web. Se ha evidenciado que Selenium + Java no solo forma una combinación potente y polivalente, sino que se apoya en un ecosistema maduro de librerías y frameworks como TestNG y WebDriverManager. Dichos frameworks, a su vez, permiten la creación de scripts de prueba potentes y escalables con la capacidad suficiente para interactuar con los elementos de la interfaz de usuario.

- El examen de los procedimientos de prueba tradicionales ha puesto de manifiesto los inconvenientes de la ejecución manual que, entre otros, están el tiempo prolongado de ejecución, la posibilidad de producirse errores humanos o la dificultad para reproducir complejos escenarios. En este sentido, la automatización se establece como una solución directa a estos inconvenientes, ofreciendo una gran posibilidad de poder avanzar en la mejora en la eficiencia y en la calidad. El sistema propuesto ha demostrado cómo la automatización de las pruebas de software puede demostrar una reducción del tiempo de ejecución, una mayor cobertura de pruebas de software, la posibilidad de obtener reportes generados de forma automática y continua y, por tanto, la posibilidad de identificar defectos mucho antes y en condiciones más fiables.

- Se logró crear con éxito un sistema de pruebas automatizadas utilizando la biblioteca Selenium junto con el lenguaje de programación Java; así como con la implementación de algunas herramientas como TestNG para la organización de la suite de pruebas y ExtentReports para la generación de los informes. Asimismo, el sistema cuenta con una interfaz gráfica (GUI) que permite la interacción del usuario, es decir, la selección dinámica de tests y una visualización en tiempo de la ejecución de las pruebas. La implementación de lógica de reportería de ExtentReports y la posibilidad de

adjuntar las capturas de pantalla de la ejecución en caso de fallos aseguran que cada ejecución de la prueba es totalmente trazable.

RECOMENDACIONES

- El ecosistema de automatización se sugiere que sea mantenido de manera periódica y actualizado para aprovechar nuevas funcionalidades de librerías como las anteriormente discutidas, por ejemplo, Selenium, TestNG, ExtentReports. Esto puede incluir la revisión de versiones, la integración de nuevos WebDriver, la evaluación de herramientas complementarias que mejoren la estabilidad y/o el rendimiento de las pruebas; incluso puede incluir revisar el proceso de configuración del entorno de desarrollo, así como las dependencias y sus versiones, todo lo cual permite, por ejemplo, replicar y colaborar con el resto de los equipos de desarrollo.

- Se plantea como una opción acoplar la automatización de pruebas como un paso mínimo dentro del ciclo de vida de desarrollo del software, en especial para las pruebas de regresión, de humo y de verificación de funcionalidades críticas; lo que se puede hacer para quitarle provecho a los beneficios de adoptar una automatización de pruebas a la verificación de funcionalidad del software. Una práctica que se podría desarrollar es el implementación de una estrategia de pruebas de acuerdo con la cual se extienda una automatización para los casos de prueba que más errores/dificultades presenten o los test de los casos de prueba que se cuente que pueden llegar a necesitar una existencia muy amplia para convertirse en caso de prueba. Este soporte de automatización permitirá a los evaluadores dejar fuera de sus tareas aquellas tareas manuales y repetitivas para dejarles enfocarse en el despliegue de pruebas de exploración o de situaciones más complejas que requieren juicio humano.

- Con el fin de lograr mejorar este aspecto todavía más, al mismo tiempo el sistema también sería mucho más interesante y útil si se migrase la interfaz gráfica de usuario (GUI) con la que se implementó el prototipo a alguna tecnología más sólida y escalable como, por ejemplo, a una aplicación web o a una robusta línea de comandos. Con esto podría, por ejemplo realizarse la integración continua (CI/CD) con Jenkins o GitLab de forma que el proceso

de integración fuera mucho más simplificado. Otra opción sería la de ampliar el propio sistema para que pueda soportar pruebas paralelas, de manera que se pudiese obtener valores bajos para los tiempos de ejecución de los grandes lotes de pruebas o de las suites de pruebas. Otra posibilidad sería la de gestionar de forma más avanzada los datos de las pruebas a través, por ejemplo, de la lectura de datos externos (CSV, Excel) o a través del uso de una base de datos que permita desacoplar los datos de prueba del propio código, y así facilitar que las pruebas sean más mantenibles.

BIBLIOGRAFÍA

Álvarez-Gayou, J. L. (2003). Cómo hacer investigación cualitativa: Fundamentos y metodología. Paidós Educador.

Apache Software Foundation. (s.f.). Apache Maven (Versión 3.x). Recuperado de https://maven.apache.org/

Balestrini, M. (2002). Cómo se elabora el proyecto de investigación. BL Consultores.

Bernal, C. A. (2010). Metodología de la investigación: para administración, economía, humanidades y ciencias sociales (3.ª ed.). Pearson Educación.

Beust, C., & Suleiman, H. (2007). Next generation Java testing: TestNG and advanced concepts. Pearson Education.

Bonilla, E., & Rodríguez, P. (2005). Más allá del dilema de los métodos: la investigación en ciencias sociales. Grupo Editorial Norma.

ExtentReports. (s.f.). ExtentReports for Java (Versión 5.x). Recuperado de https://www.extentreports.com/docs/versions/5/java/

Freeman, A. (2011). Pro ASP.NET MVC 3 Framework. Apress.

Google Inc. (s.f.). ChromeDriver [Software]. Recuperado de https://chromedriver.chromium.org/

Goucher, A. & Riley, D. (2011). Beautiful Testing: Leading Professionals Reveal How They Improve Software. O'Reilly Media.

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). Metodología de la investigación (6.ª ed.). McGraw-Hill.

Jones, C. (2012). Software Engineering Best Practices. McGraw-Hill Education.

Kaner, C., Falk, J., & Nguyen, H. Q. (2002). Testing Computer Software (2nd ed.). Wiley.

Martínez, M. (2006). La investigación cualitativa etnográfica en educación. Trillas.

Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley Professional.

Mozilla Foundation. (s.f.). GeckoDriver [Software]. Recuperado de https://github.com/mozilla/geckodriver

OpenJS Foundation. (s.f.). Java Development Kit (JDK) (Versión 11+). Recuperado de https://openjdk.java.net/

Page, A., Johnston, K., & Rollison, B. (2008). How we test software at Microsoft. Microsoft Press.

Pressman, R. S. (2010). Ingeniería del software: Un enfoque práctico (7ª ed.). McGraw-Hill.

Rodríguez, G., Gil, J., & García, E. (1999). Metodología de la investigación cualitativa. Aljibe.

Rouse, M. (2019). Selenium Definition. TechTarget. Recuperado de: https://www.techtarget.com/

Sabino, C. (1992). El proceso de investigación. Panapo.

Software & Systems Engineering Standards Committee of the IEEE Computer Society. (2008). IEEE standard for software and system test documentation. IEEE.

Software Freedom Conservancy. (s.f.). Selenium WebDriver (Versión 4.x). Recuperado de https://www.selenium.dev/documentation/webdriver/

Software Freedom Conservancy. (s.f.). WebDriverManager (Versión 5.x). Recuperado de https://bonigarcia.dev/webdrivermanager/

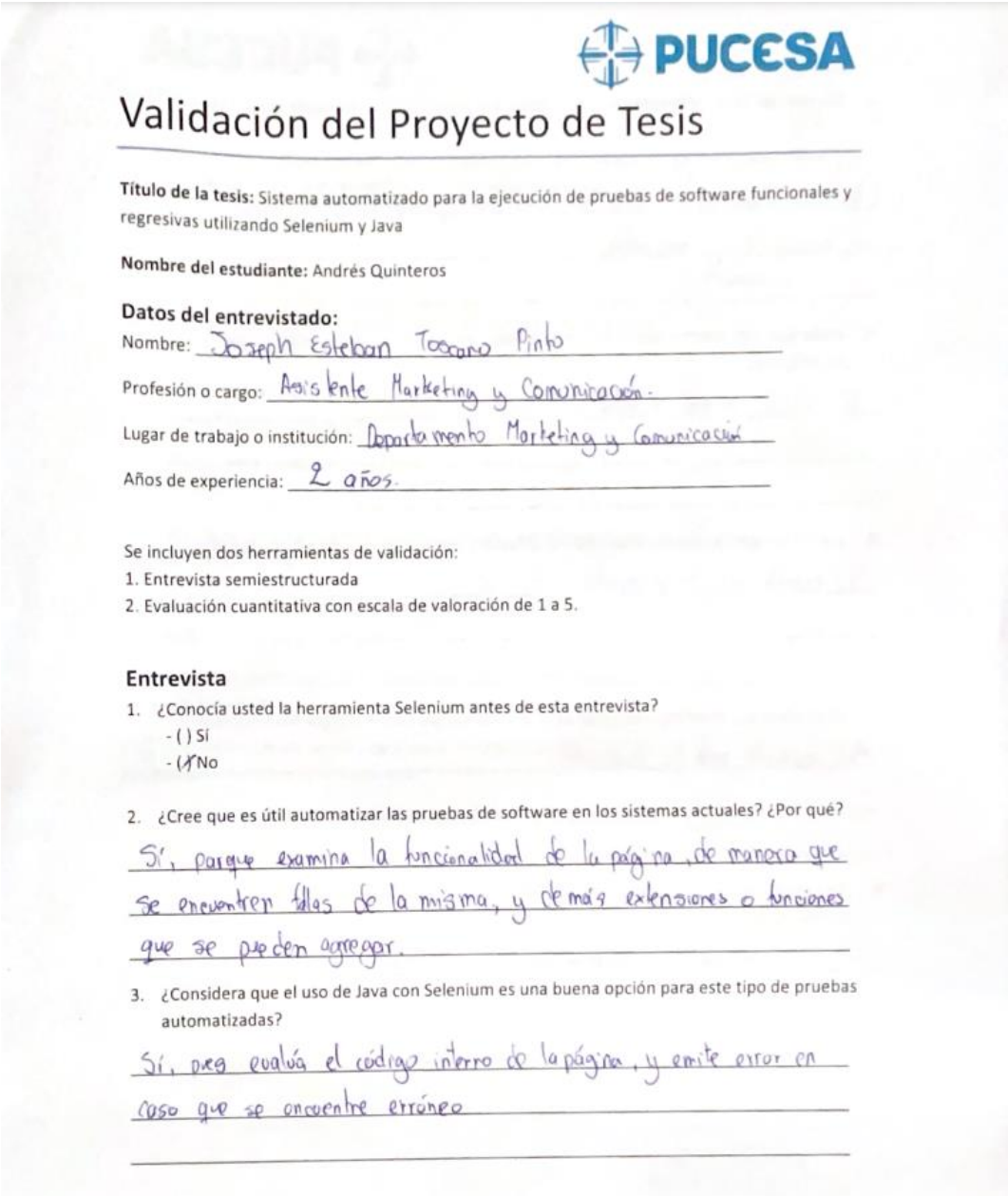
Sommerville, I. (2011). Ingeniería del software (9ª ed.). Pearson Educación.

Tamayo, M. (2006). El proceso de la investigación científica. Limusa.

Yin, R. K. (2014). Estudio de caso: diseño y métodos (5.ª ed.). Sage Publications.

ANEXOS

Anexo 1.



PUCESA

Validación del Proyecto de Tesis

Título de la tesis: Sistema automatizado para la ejecución de pruebas de software funcionales y regresivas utilizando Selenium y Java

Nombre del estudiante: Andrés Quinteros

Datos del entrevistado:

Nombre: Joseph Esteban Toscano Pinto

Profesión o cargo: Asistente Marketing y Comunicación.

Lugar de trabajo o institución: Departamento Marketing y Comunicación

Años de experiencia: 2 años.

Se incluyen dos herramientas de validación:

1. Entrevista semiestructurada
2. Evaluación cuantitativa con escala de valoración de 1 a 5.

Entrevista

1. ¿Conocía usted la herramienta Selenium antes de esta entrevista?
   - ( ) Sí
   - (X) No

2. ¿Cree que es útil automatizar las pruebas de software en los sistemas actuales? ¿Por qué?

   Sí, porque examina la funcionalidad de la página, de manera que se encuentren fallas de la misma, y demás extensiones o funciones que se pueden agregar.

3. ¿Considera que el uso de Java con Selenium es una buena opción para este tipo de pruebas automatizadas?

   Sí, pues evalúa el código interno de la página, y emite error en caso que se encuentre erróneo

PUCESA

4. Después de conocer el sistema desarrollado, ¿le parece fácil de entender y usar? ¿Por qué?
   - (1) Sí
   - ( ) No

Es una herramienta fácil de entender y emplear, pues la interfaz es accesible y amigable.

5. ¿Cree que este sistema puede ayudar a detectar errores antes de poner en funcionamiento un software?

Sí, debido a que evalúa el código de página.

6. ¿Le parece que la tesis tiene una buena aplicación práctica dentro del área de tecnología?

Sí, cumple con lo esperado

7. ¿Considera que el objetivo principal de la tesis se cumple con el sistema desarrollado?

Sí, cumple con lo esperado

8. ¿Le gustaría ver este tipo de sistemas aplicados en su lugar de trabajo o en entornos reales de desarrollo?

Sí, cumple con lo esperado.

PUCESA

Matriz de Evaluación

Por favor, califique cada afirmación con una escala del 1 al 5, donde: 1 = Muy en desacuerdo, 2 = En desacuerdo, 3 = Neutral, 4 = De acuerdo, 5 = Muy de acuerdo.

Ítem	Calificación (1 a 5)
1. El sistema automatizado responde adecuadamente a una necesidad real en el contexto universitario.	4
2. La elección de Selenium y Java como herramientas es pertinente para la automatización de pruebas.	5
3. El enfoque técnico del sistema es comprensible y está bien fundamentado.	5
4. El sistema demuestra una correcta aplicación de conocimientos en ingeniería de software.	5
5. La metodología empleada en el desarrollo del proyecto fue clara y adecuada.	4
6. La propuesta aporta valor al proceso de pruebas funcionales en entornos académicos.	5
7. El desarrollo muestra un nivel de innovación dentro de las posibilidades del entorno universitario.	5
8. El sistema puede ser comprendido y replicado por otros estudiantes o técnicos.	4
9. El proyecto está alineado con las necesidades tecnológicas de la PUCE Ambato.	4
10. La propuesta tiene sustento académico y práctico suficiente para ser implementada.	5

Firma del estudiante

Firma del entrevistado

Pontificia Universidad Católica del Ecuador | Sede Ambato

DEPARTAMENTO DE MARKETING Y COMUNICACIÓN