DISEÑO DE UN PROTOTIPO DE METODOLOGÍA INTEGRAL PARA LA AUTOMATIZACIÓN DE ASEGURAMIENTO Y CONTROL DE CALIDAD DE SOFTWARE: OPTIMIZACIÓN DE TIEMPO Y RENTABILIDAD EMPRESARIAL

AUTORES

HUGO GARCIA CLAVIJO

VANESSA BETANCUR GIL

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

INGENIERÍA DE SOFTWARE

MEDELLÍN

2023

TRABAJO PARA OPTAR AL TÍTULO DE INGENIERO DE SOFTWARE

AUTORES

HUGO GARCIA CLAVIJO

VANESSA BETANCUR GIL

TUTORES

INGENIERO CÉSAR DÍAZ RENDÓN

INGENIERO JAIME ERNESTO SOTO

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

INGENIERÍA DE SOFTWARE

MEDELLÍN

2023

Contenido

Glosario 6

Resumen 7

Introducción 8

1. El problema 10

1.1 Descripción 10

1.2 Formulación de la pregunta de investigación 10

2. Justificación 11

3. Objetivos 12

3.1 Objetivo general 12

3.1 Objetivos específicos 12

4. Marco Referencial 13

4.1 Antecedentes 13

4.2 Marco Teórico 17

4.2.1 Tipos de pruebas en el aseguramiento de calidad de software 17

4.2.2 Metodologías de desarrollo de software 18

4.2.3 Herramientas para pruebas de calidad de software 20

4.2.4 SCRUM 20

4.2.5 Ciclo de Vida de Desarrollo de Software Seguro S-SDLC 23

4.2.6 Automatización de Pruebas 30

5. Metodología 40

Actividad 1: Revisión de herramientas disponibles 40

Actividad 2: Evaluación de casos de uso 40

Actividad 3: Diseño de propuesta de metodología funcional 41

6. Resultados 42

Elección de la herramienta de automatización 42

Ejecución de un escenario de prueba automatizada en Robot Framework 45

Archivos Adicionales 50

Tests 53

Keywords 54

Propuesta de Metodología Integral 56

1. Evaluación y Selección 57

2. Codificación de scripts 58

3. Implementación y Recopilación de información 58

4. Integración continua 58

5. Retroalimentación 58

7. Conclusiones 62

8. Recomendaciones 63

9. Bibliografías 64

Tabla de Ilustraciones

Ilustración 1. 26

Ilustración 2. 27

Ilustración 3. 39

Ilustración 4. 43

Ilustración 5. 46

Ilustración 6. 47

Ilustración 7. 48

Ilustración 8. 49

Ilustración 9. 49

Ilustración 10. 50

Ilustración 11. 52

Ilustración 12. 55

Ilustración 13. 57

Ilustración 14. 59

Glosario

Automatización: Proceso de realizar tareas o funciones de manera automática con la ayuda de sistemas informáticos o maquinaria, reduciendo la intervención humana y aumentando la eficiencia.

Calidad de Software: Grado en el que un programa de computadora cumple con los requisitos especificados y satisface las necesidades del usuario, minimizando errores y defectos.

Pruebas: Actividades sistemáticas realizadas para evaluar y verificar el funcionamiento de un producto, proceso o software, con el fin de detectar errores o problemas.

Rentabilidad Empresarial: Capacidad de una empresa para generar ganancias y beneficios en relación con los costos y la inversión realizada, medida como un indicador clave de su éxito financiero.

Eficacia: Capacidad de lograr un resultado deseado o cumplir con un objetivo de manera eficiente y exitosa, generalmente relacionada con la consecución de metas y resultados positivos.

Resumen

Los desafíos que enfrentan las organizaciones en el ámbito del aseguramiento y control de calidad de software, al depender de métodos tradicionales, se centran en la falta de automatización. Esto conlleva problemas como inversiones considerables de tiempo y recursos humanos, retrasos en la entrega de productos, riesgos de comprometer la calidad del software y la posibilidad de errores humanos. Además, la carencia de automatización afecta la detección temprana de defectos, lo que puede resultar costoso y perjudicial para la reputación de la empresa.

Este trabajo surge de la necesidad de diseñar y aplicar un prototipo de metodología integral de automatización para el aseguramiento y control de calidad de software, con el objetivo de reducir los tiempos de ejecución de pruebas y mejorar la rentabilidad empresarial. La investigación examina diversas herramientas utilizadas en pruebas automatizadas, destacando las más eficientes, y recopila información sobre prácticas actuales mediante casos de prueba.

En respuesta a estos desafíos, este proyecto busca mejorar la gestión de proyectos de software, destacando la automatización de pruebas como una solución clave a través del diseño y ejecución de un prototipo de metodología amigable y fácil de aplicar en este proceso.

Introducción

En el dinámico escenario del desarrollo de software, donde la eficiencia y calidad son imperativos ineludibles, los métodos tradicionales de aseguramiento y control de calidad enfrentan desafíos significativos que impactan directamente la temporalidad y la rentabilidad empresarial. La carencia de procesos automatizados en estas fases críticas se traduce en una inversión sustancial de tiempo y recursos humanos, amenazando la puntualidad en la entrega de productos y aumentando la probabilidad de detección tardía de defectos, así como la posibilidad de errores humanos significativos.

Numerosos estudios han enfrentado este desafío empresarial desde diversas perspectivas, abarcando desde compañías de seguros hasta plataformas de pagos en línea, con el objetivo de profundizar en sus análisis y promover una apuesta decidida por la calidad en el desarrollo de software. Las investigaciones que se presentan a continuación resaltan avances significativos y desafíos clave en el ámbito del aseguramiento y control de calidad del software.

Lucero (2022), por ejemplo, centró su análisis en mejorar y optimizar el proceso de calidad de software en una compañía de seguros mediante la automatización de pruebas de regresión para aplicaciones web y APIs. En este proyecto, implementó un framework bajo el marco ágil Scrum, aprovechando enfoques contemporáneos como el Desarrollo Dirigido por Comportamiento (BDD) y patrones de diseño modernos como el Modelo de Objeto de Página (POM).

Al implementar el modelo de automatización de pruebas diseñado para optimizar la gestión de la calidad de software en la central de riesgos Cifin, Huamán (2021) logró una eficiencia del 94.38% en costo de oportunidad y destacó el cumplimiento exitoso en el Time To Market para los clientes más importantes. La automatización de los casos de prueba de los productos core de

CIFIN, reutilizados en regresiones y pruebas de no impacto, resultó en una notable reducción de los tiempos de ejecución.

Chacón (2019), enfocándose en la implementación de la automatización del aseguramiento de la calidad para el puesto de Software Packaging en IBM Costa Rica, proporcionó información valiosa que fortaleció la confianza del cliente y mejoró los tiempos de trabajo. Además, Rueda et al. (2016) subrayan la importancia de la automatización de pruebas para mejorar la eficiencia en el desarrollo de software, destacando la capacidad de las pruebas automatizadas para reducir el tiempo de ejecución y liberar recursos humanos para tareas más complejas y críticas.

Cárdenas (2016), en otro contexto, logró un avance significativo en el proceso de Testing dentro de la empresa de pagos online, mejorando los tiempos de regresión y reduciendo la deuda técnica mediante la eliminación de duplicados en la suite de pruebas de regresión y la automatización exitosa de casos pendientes.

Por último, en línea con la calidad de software según García (2015), la implementación previa de la Integración Continua (IC) en proyectos anteriores ha surgido como una estrategia esencial para dinamizar la validación y verificación en el desarrollo de software. A pesar de la aparente complejidad inicial, la estructuración cuidadosa de dichos proyectos pasados se presentó como un factor determinante para priorizar la automatización

Ante esta problemática, el presente proyecto se erige como un esfuerzo por abordar estos desafíos mediante la creación de estrategias y mecanismos que aúnen eficiencia temporal y mejora sostenible de la rentabilidad en el contexto del aseguramiento y control de calidad de software.

1. El problema

1.1 Descripción

Los métodos tradicionales de aseguramiento y control de calidad de software presentan desafíos en términos de eficiencia temporal y rentabilidad empresarial. La ausencia de automatización en estos procesos resulta en una inversión considerable de tiempo y recursos humanos, lo que puede retrasar la entrega de productos y comprometer su calidad. Esto se traduce en demoras en la entrega de productos, detección tardía de defectos y una elevada probabilidad de errores humanos. En consecuencia, el problema central radica en las repercusiones negativas generadas por la carencia de automatización, afectando la capacidad de las empresas para acceder a recursos básicos y lograr sus objetivos eficientemente.

1.2 Formulación de la pregunta de investigación

¿Cómo puede el diseño e implementación de un prototipo de metodología de automatización en el aseguramiento y control de calidad de software transformar eficazmente la dinámica tradicional de los procesos, incluyendo aspectos clave como las pruebas, la detección de defectos y la resolución de situaciones críticas, no solo optimizando el tiempo involucrado, sino también generando un impacto significativo en la rentabilidad empresarial?

2. Justificación

Esta propuesta para implementar una estrategia integral de automatización en los procesos de aseguramiento y control de calidad del software dentro de las organizaciones se basa en la necesidad imperativa de evolucionar desde enfoques tradicionales hacia soluciones más eficientes y rentables. El objetivo principal de esta iniciativa es la reducción significativa del tiempo necesario para llevar a cabo las actividades de aseguramiento y control de calidad en el desarrollo de productos. Para lograr este propósito, se busca aprovechar la automatización para la ejecución ágil y precisa de casos de prueba, la detección temprana de defectos y la generación de informes detallados, acelerando así la identificación y resolución de problemas en el proceso.

Más allá de la agilización de los procedimientos, esta propuesta se enfoca en la mejora integral de la calidad de los productos al minimizar el riesgo de errores humanos y garantizar una mayor uniformidad en las evaluaciones. Es crucial destacar que la optimización del tiempo en el aseguramiento y control de calidad tiene un impacto directo en la rentabilidad de las organizaciones. Al implementar esta automatización a través de un prototipo, se libera el potencial para reasignar recursos humanos hacia tareas que puedan aportar el mismo o mayor valor añadido, al tiempo que se posibilita la entrega rápida y confiable de productos. Esta transformación contribuye a la reducción de errores en la producción, consolidando una imagen de marca más sólida y fomentando la satisfacción del cliente. La liberación de recursos también ofrece flexibilidad para asignarlos según las necesidades específicas, ya sea para tareas de igual, mayor o menor importancia, garantizando una utilización eficiente de los recursos disponibles.

3. Objetivos

3.1 Objetivo general

Diseñar un prototipo funcional de metodología para la automatización de aseguramiento y control de calidad de software, basado en un marco de trabajo integral. Este prototipo tiene como propósito principal agilizar los procesos de evaluación y la mejora de los productos, reduciendo los tiempos de ejecución y aumentando la rentabilidad de la empresa.

3.1 Objetivos específicos

1. Investigar y seleccionar herramientas apropiadas para implementar un modelo de pruebas automatizadas en el desarrollo de software, considerando una solución ecléctica.
2. Adaptar el marco de trabajo más adecuado para la automatización de casos de prueba, con especial atención a la implicación de casos de uso específicos de la empresa.
3. Diseñar, probar y evaluar un prototipo de metodología integral para la creación y ejecución de pruebas automatizadas en S1ESS, asegurando su integración efectiva en el proceso de entrega continua e integración continua.

4. Marco Referencial

4.1 Antecedentes

En la actualidad, en el ámbito del desarrollo de software, se han implementado diversas estrategias para la automatización de procesos de aseguramiento y control de calidad en proyectos tanto internos como aquellos realizados para empresas externas. La adopción de estas prácticas ha revelado beneficios sustanciales al reducir significativamente los tiempos de ejecución, lo que, a su vez, se traduce en una optimización de costos asociados a dichos procesos.

Lucero (2022), por ejemplo, centró su análisis en mejorar y optimizar el proceso de calidad de software en una compañía de seguros mediante la automatización de pruebas de regresión para aplicaciones web y APIs. En este proyecto, implementó un framework bajo el marco ágil Scrum, aprovechando enfoques contemporáneos como el Desarrollo Dirigido por Comportamiento (BDD) y patrones de diseño modernos como el Modelo de Objeto de Página (POM).

En el ámbito tecnológico, hace uso de herramientas de vanguardia como Selenium, Rest Assured, Java, Cucumber y Serenity. Además, aplican dos de las buenas prácticas fundamentales de los principios SOLID para garantizar una arquitectura robusta y mantenible.

La integración exitosa del framework con Jenkins juega un papel crucial, ya que permite la ejecución automática de pruebas manuales y de regresión cada vez que se despliegan historias de usuario en el entorno de pruebas. Este enfoque ha resultado en una significativa reducción en el

tiempo de ejecución, una mayor cobertura de pruebas y la garantía de la calidad necesaria para asegurar la productividad y eficiencia en beneficio de la compañía.

Huamán (2021) demuestra que al implementar el modelo de automatización de pruebas definido para optimizar la gestión de la calidad de software en una la central de riesgos Cifin, evidenció una eficiencia del 94.38% en costo de oportunidad y presentó cumplimiento en el Time To Market para sus clientes más importantes. Con la automatización de los casos de prueba de los productos core de CIFIN al reutilizarlos en las regresiones y pruebas de no impacto, minimiza los tiempos de ejecución a partir de la segunda iteración en adelante en todos los casos. Al implementar la automatización de pruebas con un framework flexible y adaptable, se permitió reutilizar el 100% de los artefactos generados para la ejecución de las pruebas de regresión con flujos repetitivos de inicio a fin.

El planteamiento de escenario y diseño de casos bajo los lineamientos del International Software Testing Qualifications Board (ISTQB) fueron esenciales para una definición adecuada de los escenarios de prueba. La inyección de la automatización de pruebas desde la fase de iniciación de desarrollo para poder identificar errores en etapas tempranas fue posible con ayuda de la estandarización de mapeo de objetos. Esta identificación temprana hace parte de las buenas prácticas de optimización en etapas posteriores significa un esfuerzo menor al realizar pruebas y una reducción en la cantidad de defectos considerable.

De acuerdo con Chacón (2019) en su investigación centrada en la implementación de la automatización del aseguramiento de la calidad para el puesto de Software Packaging en IBM Costa Rica, logra obtener información valiosa que proporcionó una perspectiva amplia sobre los

cambios necesarios. La creación de documentación para validar los procesos de aseguramiento de calidad no solo fortaleció la confianza del cliente, sino que también mejoró los tiempos de trabajo y facilitó las tareas de los colaboradores.

También, el estudio revela la existencia de marcos de automatización de pruebas en IBM Costa Rica, aunque constató que estos no se estaban implementando. La investigación sugirió la necesidad de un marco más adaptado al equipo de Software Packaging para obtener resultados más centralizados. Métodos como la encuesta y entrevista realizadas a los colaboradores demostraron que la gerencia ve favorablemente los cambios en el proceso de aseguramiento de calidad, especialmente si se trata de una herramienta beneficiosa que los colaboradores están dispuestos a utilizar.

Para asegurar que la herramienta cumpliera con los estándares de los clientes de Software Packaging, recopilaron y aplicaron los estándares mediante correos y documentos internos, donde la flexibilidad de la herramienta permitió la modificación de estos estándares según lo necesario, ya fuera para agregar o eliminar alguno.

En sintonía con el trabajo de Rueda et al., (2016), resalta la importancia de la automatización de pruebas para mejorar la eficiencia en el proceso de desarrollo de software. Enfatiza la capacidad de las pruebas automatizadas para reducir el tiempo de ejecución al abordar tareas repetitivas y no críticas, liberando así los recursos humanos para tareas más complejas y críticas. Además, subraya la necesidad de una evaluación cuidadosa del costo y beneficio al elegir entre pruebas manuales y automatizadas, considerando elementos clave como el conocimiento de las herramientas, habilidades de programación y la efectividad en la detección de errores.

En la misma serie de investigación, analiza la situación actual en Axede S.A., donde observa

una carga significativa de tiempo y recursos asociados con las pruebas manuales. Este problema inicial plantea la necesidad de una solución que pueda optimizarse mediante la automatización de casos de prueba reutilizables. Por lo tanto, la investigación se orienta hacia la respuesta fundamental a la pregunta de cómo diseñar una propuesta para la ejecución automatizada del aseguramiento y control de calidad sobre el código fuente, y el estudio se embarca en ofrecer soluciones específicas y adaptadas a la realidad de Axede S.A., transformando el problema inicial en una oportunidad para optimizar y mejorar el proceso de aseguramiento y control de calidad.

En otro contexto, Cárdenas (2016) ha logrado un avance significativo en el proceso de Testing dentro de la empresa de pagos online. En que llevó a cabo un mantenimiento exhaustivo de casos de prueba en el equipo de Payments Methods, eliminando duplicados en la suite de pruebas de regresión. Además, automatizan con éxito 37 casos pendientes, mejorando notoriamente los tiempos de regresión y reduciendo la deuda técnica.

Durante la automatización, se identificaron y aplicaron mejoras, como el estándar de manejo de excepciones, proporcionando descripciones precisas de los momentos en que los scripts generan fallos para análisis y mantenimiento más certeros. También implementó la generación de datos en ambientes alternos para minimizar la contingencia en la ejecución de la suite automatizada.

En cuanto a la dinámica de los equipos de Testing, adoptaron prácticas enriquecedoras y logró aislar las dependencias entre analistas y desarrolladores en casos de prueba con fallos. Aunque, la adaptabilidad de los equipos a estas mejoras generó discusiones sobre los tiempos

para incluirlas como obligatorias en el proceso de automatización, se logró argumentar eficazmente sus beneficios y establecerlas, brindando la oportunidad de considerar más mejoras en el futuro.

Por último, y en línea con la calidad de software según García (2015), la implementación de la Integración Continua (IC) emerge como una estrategia esencial para dinamizar la validación y verificación en el desarrollo de software. Su automatización ha evidenciado una reducción palpable en el tiempo de ejecución y la entrega de información en tiempo real sobre el progreso del proyecto. A pesar de la aparente complejidad inicial, la estructuración cuidadosa del proyecto se presenta como un factor determinante para priorizar la automatización. Con el respaldo de herramientas accesibles y la capacidad de reutilizar conocimientos, la transición hacia la IC se desarrolla de manera gradual, asegurando eficiencia y rentabilidad en el aseguramiento de la calidad del software en el cual la IC se erige como una solución eficaz para optimizar procesos y obtener beneficios notables en el desarrollo de software, especialmente en entornos ágiles.

4.2 Marco Teórico

4.2.1 Tipos de pruebas en el aseguramiento de calidad de software

En el contexto del aseguramiento de calidad de software, diversos tipos de pruebas desempeñan un papel crucial para garantizar el correcto funcionamiento, la usabilidad y el rendimiento de las aplicaciones. Estos métodos de prueba abordan diferentes aspectos de la funcionalidad y la calidad del software, contribuyendo a la entrega de productos confiables y eficaces.

Pruebas Unitarias. Las pruebas unitarias se realizan al inicio del Ciclo de Vida del Desarrollo de Software (SDLC, por sus siglas en inglés) y se centran en evaluar piezas o unidades individuales del software. Cualquier función, procedimiento, método o módulo puede considerarse una unidad que se somete a estas pruebas para verificar su corrección y comportamiento esperado. Estas son ejecutadas por los desarrolladores durante la fase de desarrollo (Loadview, 2020).

Pruebas de Rendimiento. Son un subtipo de pruebas no funcionales, dónde evalúan la velocidad, estabilidad y escalabilidad del software, abordando aspectos como la CPU, velocidad de carga, capacidad para gestionar tráfico máximo y utilización de recursos del servidor. Incluyen pruebas de carga y pruebas de esfuerzo para explorar diversos escenarios y niveles de estrés (Loadview, 2020).

En conjunto, estos tipos de pruebas desempeñan un rol esencial en el aseguramiento de calidad de software, asegurando su funcionalidad, usabilidad y rendimiento. La ejecución adecuada de estas pruebas contribuye a la entrega de productos de calidad, confiables y aptos para satisfacer las necesidades y expectativas de los usuarios.

4.2.2 Metodologías de desarrollo de software

De acuerdo con las definiciones proporcionadas por los autores Maida & Pacienzia (2015) en el contexto de Metodologías de Desarrollo de Software, se presentan las siguientes conceptualizaciones:

Modelo Waterfall (Cascada). El modelo Waterfall organiza el trabajo de manera secuencial, avanzando de arriba a abajo. Cada fase se verifica antes de avanzar a la siguiente, lo que asegura un progreso seguro y preciso. Las fases incluyen análisis de requisitos, diseño de sistema, diseño de programa, modificación, diseño de pruebas, codificación y mantenimiento. Sin embargo, este enfoque no es flexible para cambios y es adecuado cuando los requisitos son claros desde el principio.

Metodología de Prototipo. Esta metodología implica crear un prototipo rápido de software para obtener retroalimentación de los usuarios. Permite detectar fallos técnicos, mejoras y funcionalidad. Aunque mejora el producto final, los cambios durante el proceso pueden incrementar los costos.

Modelo Incremental. Permite probar funcionalidades antes de completar la herramienta y aprovecha mejor el tiempo. Es adaptable y no descarta trabajo previo. Las fases de trabajo se asemejan al enfoque de cascada, pero cada fase agrega una funcionalidad.

Metodología en Espiral. El objetivo es acercarse a las necesidades del cliente. Este enfoque es de alta calidad, ya que el cliente valida y ajusta el proyecto, contempla cuatro fases: planificación del proyecto, análisis del riesgo, desarrollo del prototipo y evaluación del cliente.

Diseño Rápido de Aplicaciones (RAD). Pretende desarrollar software de calidad en poco tiempo, además utiliza prototipos para recibir comentarios de los usuarios, prioriza la velocidad,

aunque aumenta la posibilidad de errores iniciales y enfatiza la retroalimentación de los usuarios para las mejoras.

4.2.3 Herramientas para pruebas de calidad de software

Pruebas funcionales:

- SoapUI
- Selenium

Pruebas de rendimiento:

- LoadRunner
- LoadNinja

Seguimiento de defectos de código:

- SonarQube
- Kiuwan

Pruebas de seguridad:

- Netsparker
- W3af

4.2.4 SCRUM

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. (Rueda Patiño et al., 2016, p.22)

Además de su aplicación en proyectos complejos, Scrum también se utiliza para abordar desafíos específicos en la gestión de proyectos. Se aplica en situaciones donde las entregas al cliente no cumplen con sus necesidades, las entregas se prolongan en exceso, los costes se elevan o la calidad del producto no es aceptable. Scrum también es valioso cuando se busca una mayor capacidad de respuesta ante la competencia, se enfrentan problemas de moral y alta rotación en los equipos, se buscan soluciones a ineficiencias sistémicas o se desea emplear un proceso especializado en el desarrollo de productos.

A su vez, dicha metodología versátil y efectiva lleva a cabo las siguientes ceremonias claves que fomentan la transparencia, la adaptación y la colaboración efectiva entre los miembros del equipo y las partes interesadas.

Ceremonias

Siguiendo la perspectiva de Roche (2023), se detallan los siguientes pasos que conforman la ceremonia de Scrum:

1. **Sprint Planning.** Al inicio del Sprint, se inspecciona el Backlog del Producto junto con el product owner. Durante esta reunión, el equipo selecciona los elementos del Backlog del Producto que abordarán durante el Sprint y se compromete a cumplir el Sprint Goal. También se distribuyen tareas y se establecen metas.
2. **Daily Scrum.** Se realiza diariamente en un tiempo máximo de quince minutos. El equipo se reúne para comunicar su progreso individual basado en la meta del Sprint. Cada miembro responde a tres preguntas: ¿Qué hice ayer para contribuir al Sprint Goal?, ¿Qué

voy a hacer hoy para contribuir al Sprint Goal?, ¿Tengo algún impedimento que me impida entregar?

3. **Sprint Review.** Se dedica a mostrar el trabajo terminado al product owner y otros stakeholders. Esta reunión permite la inspección y adaptación del producto. El enfoque está en el valor comercial entregado, no en juzgar al equipo.
4. **Sprint Retrospective**. Al final del Sprint, se obtiene retroalimentación para mejorar la cultura y el desarrollo del producto. Durante esta reunión, el equipo reflexiona sobre su trabajo, identifica áreas para mejorar y propone acciones para implementar en el próximo Sprint.
5. **Sprint Grooming o Refinement.** La práctica de refinar el Product Backlog asegura su preparación constante. A diferencia de otras reuniones, el product owner dirige esta sesión. Es esencial para asegurar la comprensión directa de los requisitos y eliminar distorsiones involuntarias de información.

Al margen de la automatización de pruebas para el aseguramiento de la calidad en software, Rendón Gutierrez (2022) presentó la investigación “Automatización de pruebas para el aseguramiento de la calidad de Adminfo Vsmart” que constató en la metodología para diseñar e implementar pruebas automatizadas, abordando los requisitos, el flujo del software y los resultados esperados. Dónde logró una implementación y ejecución eficiente mediante el uso de pipelines y servicios en la nube como AWS, evitando la complejidad de la integración de equipos y sistemas.

La herramienta que emplearon fue SonarQube para generar informes detallados de las pruebas, detectar errores y proporcionar imágenes visuales que permitieron identificar y resolver

problemas. La implementación de métricas en el Tablero de Métrica de Grafana reveló una mejora significativa en el cumplimiento de los estándares de calidad en el repositorio destinado a las pruebas automatizadas, incrementando el porcentaje del 56% al 95%. En última instancia, la automatización logró simplificar tareas repetitivas y complejas, aumentando la efectividad y productividad, a la vez que redujo costos y tiempos.

4.2.5 Ciclo de Vida de Desarrollo de Software Seguro S-SDLC

En el proceso de desarrollo de software normalmente se involucra lo que se conoce como ciclo de vida de desarrollo de software, donde se aplican una serie de fases o etapas, con sus respectivas ventajas y desventajas según el paradigma adoptado. Entonces entendemos a **S-SDLC** como el conjunto de principios y pautas de diseño que deben aplicarse en el Ciclo de Vida de Desarrollo de Software (SDLC) con el propósito de identificar, prevenir y solucionar posibles fallos de seguridad en el proceso de desarrollo y adquisición de aplicaciones. El objetivo es obtener software de alta confiabilidad y resistente contra ataques maliciosos, garantizando que el software desempeñe únicamente las funciones previstas, esté exento de vulnerabilidades, ya sean deliberadas o involuntarias, que se hayan incorporado en su ciclo de vida, y asegurando su integridad, disponibilidad y confidencialidad (AWS, 2023).

Hay diversas metodologías y algunas de ellas tienen un enfoque de desarrollo de software seguro. En el presente proyecto se estudian algunos modelos y se realiza un análisis para resaltar sus principales características, que a la vez pueden ser considerados importantes para los fines del presente trabajo investigativo.

McGraw's Seven Touchpoints (Los Siete Puntos de Contacto de McGraw). Gary McGraw (2006) plantea siete principios vitales en el desarrollo de un producto de software seguro los cuales tiene aplicabilidad en el ámbito del desarrollo de software seguro SDLC-S (Requisitos, Diseño, Implementación, Pruebas, Evaluación, Despliegue, Mantenimiento), establecidos en 2004 por el Instituto de Ingeniería Eléctrica y Electrónica (IEEE, por sus siglas en inglés), conocidos como el modelo de Gary McGraw y Cigital. Dicho modelo es considerado uno de los tres pilares de la seguridad de software, ya que une los aspectos técnicos y prácticos del desarrollo y de esa manera realiza énfasis en el empleo de buenas prácticas. Para ello, se recomienda tener en cuenta una revisión externa y los siguientes aspectos:

La revisión del código. Esta práctica puede aplicarse en varios puntos de contacto, pero principalmente en el punto de contacto de "Implementación". Durante la etapa de implementación, los desarrolladores pueden revisar el código en busca de vulnerabilidades de seguridad, como errores de programación, para corregirlos antes de que se conviertan en problemas en la etapa de pruebas y despliegue.

Análisis de riesgos de arquitectura. Esto se relaciona principalmente con el punto de contacto de "Diseño". Durante la fase de diseño, se deben evaluar los riesgos de seguridad de la arquitectura del software y tomar decisiones de diseño que mitiguen esos riesgos.

Pruebas de penetración. Las pruebas de penetración son más relevantes en el punto de contacto de "Pruebas" y pueden utilizarse para simular ataques y evaluar la seguridad del software desde un punto de vista externo.

Pruebas de seguridad basadas en el riesgo. Estas pruebas también son parte del punto de contacto de "Pruebas" y se enfocan en identificar y evaluar riesgos específicos de seguridad en el

software.

La construcción de los casos de abuso. Esta práctica está relacionada con los puntos de contacto de "Requisitos" y "Diseño". Al construir casos de abuso, se están identificando posibles amenazas y vulnerabilidades que deben abordarse desde las primeras etapas de desarrollo.

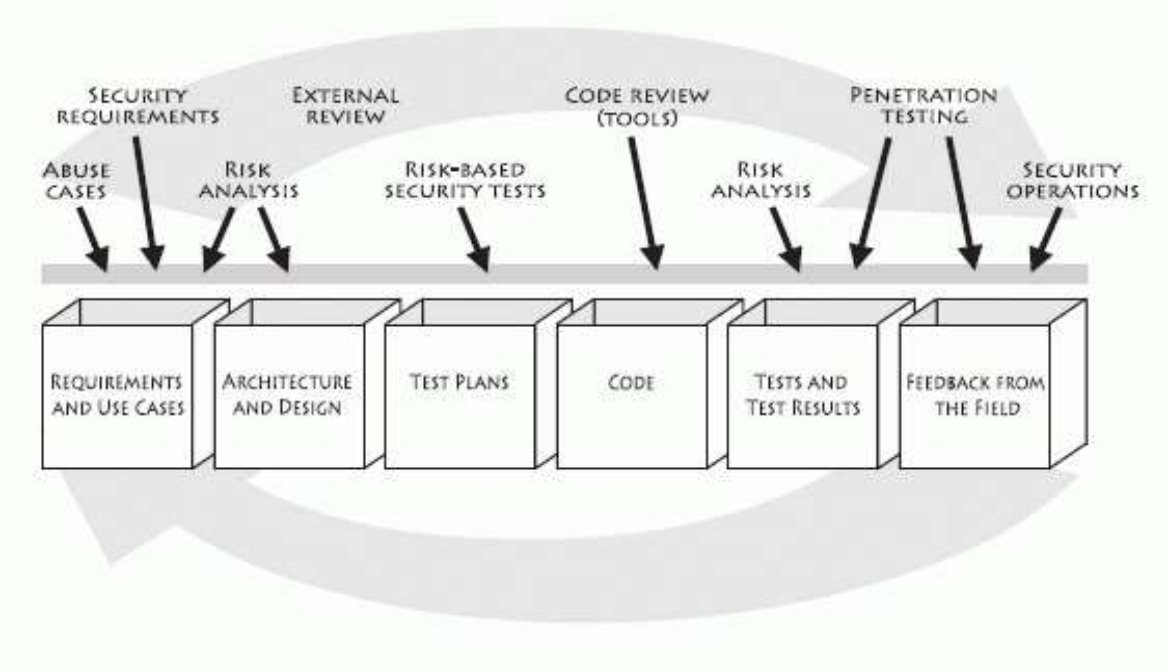
Los requisitos de seguridad. Los requisitos de seguridad son fundamentales en el punto de contacto de "Requisitos". En esta etapa, se definen los requisitos de seguridad necesarios para el software, asegurándose de que estén documentados y que se tengan en cuenta durante todo el ciclo de desarrollo.

Las operaciones de seguridad. Este punto de contacto se refiere a las prácticas de seguridad necesarias en la etapa de "Mantenimiento" y "Despliegue". Incluye la aplicación de parches de seguridad, la configuración segura de servidores y la gestión continua de la seguridad del software en entornos de producción.

Revisión externa. Las revisiones externas son relevantes en múltiples puntos de contacto, ya que involucran a terceros o expertos en seguridad para evaluar el software desde una perspectiva externa. Pueden ser parte de las etapas de "Evaluación" y "Pruebas de penetración".

Ilustración 1.

Los Siete Puntos de Contacto de McGraw

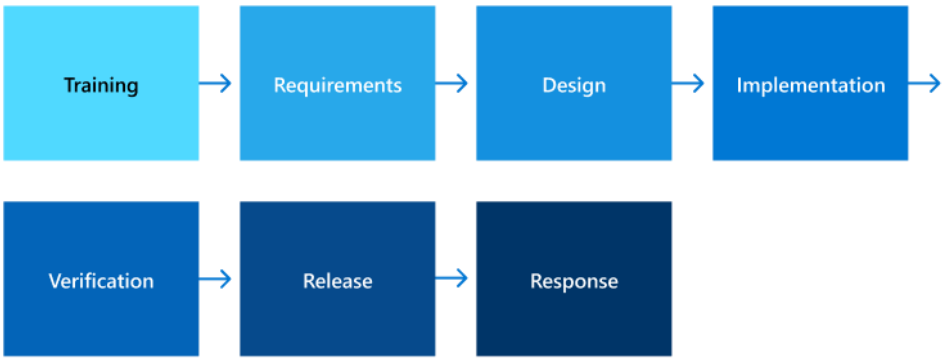


Nota. Adaptado de Seven Touchpoints for Software Security, por Gary McGraw, 2006, Fuente: http://www.swsec.com/resources/touchpoints/.

Microsoft Security Development Lifecycle (SDL). También conocida como *Ciclo de Desarrollo de Seguridad de Microsoft* de Microsoft (2016), la cual es una metodología compuesta por una serie de actividades divididas en las fases que componen la técnica y que están encaminadas al mejoramiento del desarrollo de software.

Ilustración 2.

Ciclo de vida del desarrollo de seguridad de Microsoft (SDL)



Nota. Adaptado de Ciclo de vida de Microsoft Security Development Lifecycle (SDL), por Microsoft Learn, 2023, fuente: https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle.

SDL fue propuesta en 2004 por Microsoft e incluye un proceso de modelado de amenazas, que busca identificar vulnerabilidades a nivel de código. Con el propósito de optimizar el acoplamiento según el tipo de desarrollo, SDL cuenta con dos versiones de ejecución:

SDL versión rígida, enfocada en equipos de proyectos y desarrollo de productos de gran envergadura donde los cambios son mínimos.

SDL versión ágil, cuyos desarrollos son incrementales con aumento en la frecuencia de ejecución y seguimiento de actividades, haciendo énfasis en su seguridad. Recomendada para desarrollos web.

Además, cuenta con una parte diferencial a comparación de las otras metodologías de desarrollo de software seguro, SDL incorpora una etapa de “entendimiento de seguridad” y “aseguramiento”, en la cual se da seguimiento a cada uno de los resultados del proceso.

SDL de Microsoft se enfoca en asegurar que la seguridad sea una consideración fundamental en todas las etapas del ciclo de vida del desarrollo de software, desde la planificación y levantamiento de requerimientos hasta el mantenimiento continuo. A continuación, se enumeran algunas de las características claves de Microsoft SDL:

- Entrenamiento y concienciación: Microsoft proporciona capacitación y concienciación en seguridad a los desarrolladores y equipos de desarrollo para asegurarse de que comprendan los aspectos de seguridad y las mejores prácticas.
- Análisis de riesgos: Se identifican y evalúan los riesgos de seguridad en cada etapa del desarrollo de software, desde el diseño inicial hasta la implementación y las pruebas.
- Revisiones de código y pruebas de seguridad: Se realizan revisiones de código para detectar posibles vulnerabilidades y se llevan a cabo pruebas de seguridad para evaluar la resistencia del software a los ataques.
- Herramientas de seguridad automatizadas: Microsoft utiliza herramientas de análisis estático y dinámico, así como herramientas de prueba de penetración automatizadas para identificar posibles problemas de seguridad.
- Revisión de amenazas y modelado de ataque: Se consideran posibles amenazas y se modelan los escenarios de ataque para comprender mejor las debilidades y fortalezas del software.
- Gestión de incidentes de seguridad: Se establecen procedimientos para gestionar y responder a incidentes de seguridad en el software después de su lanzamiento.
- Documentación y estándares de seguridad: Se crea documentación detallada que

describe los estándares y directrices de seguridad que deben seguirse en el desarrollo de software.

- Participación de expertos en seguridad: Se involucran expertos en seguridad en el proceso de desarrollo para revisar y evaluar el software desde una perspectiva de seguridad.

Tabla 1.

Principales características de las metodologías S-SDLC en las pruebas

Ítem	McGraw's Seven Touchpoints	SDL
Actividades de pruebas de verificación y validación	Las actividades constructivas se orientan al diseño, la defensa, y la funcionalidad, se plantean cambios a realizar en la especificación de requisitos, diseño o implementación para mitigar o eliminar los riesgos.	Los equipos de desarrollo definen y establecen una lista de las herramientas aprobada por el líder de seguridad del equipo del proyecto; el equipo debe realizar un análisis estático al código fuente
	La prueba de funcionalidad de la seguridad se lleva a cabo con técnicas estándar de pruebas funcionales y pruebas de seguridad, con base en el riesgo a partir de patrones de ataque. Un buen plan de pruebas de seguridad hace ambas cosas.	El software ya es totalmente funcional. Se realizan pruebas al software en el área de la superficie de ataque.

Nota. Esta tabla compara los dos enfoques relacionados con las actividades de pruebas, verificación y validación en el desarrollo de software.

4.2.6 Automatización de Pruebas

La automatización de pruebas se ha convertido en un componente esencial en el proceso de desarrollo de software para garantizar la calidad y eficiencia de las aplicaciones.

Selenium. Según la documentación oficial de Selenium (Selenium.dev, 2023), Selenium es un conjunto de herramientas de código abierto que se utiliza para automatizar pruebas en aplicaciones web. Fue creado originalmente por Jason Huggins en 2004 como una solución interna para automatizar pruebas en un proyecto web, pero posteriormente se convirtió en un proyecto de código abierto ampliamente utilizado. Selenium permite a los desarrolladores e ingenieros de pruebas automatizar interacciones con un navegador web, realizar pruebas de regresión y asegurarse de que las aplicaciones web funcionen de manera consistente en diferentes navegadores y plataformas. Además, proporciona extensiones para emular la interacción del usuario con los navegadores, un servidor de distribución para escalar la asignación de navegadores y la infraestructura para implementaciones de la especificación W3C WebDriver que le permite escribir código intercambiable para los principales navegadores web.

Es importante resaltar que en el núcleo de Selenium se encuentra WebDriver. WebDriver es una interfaz para escribir conjuntos de instrucciones que se pueden ejecutar indistintamente en diversos navegadores.

Componentes de Selenium. Selenium consta de algunos componentes clave:

- Selenium WebDriver: Es la parte central de Selenium y proporciona una API para interactuar con navegadores web. Permite la automatización de acciones como hacer clic en enlaces, completar formularios y extraer información de una página web. Selenium

WebDriver es compatible con múltiples navegadores como Chrome, Firefox, Edge y Safari.

- Selenium IDE (Integrated Development Environment): Es una extensión de navegador que permite la grabación y reproducción de interacciones con la aplicación web. Aunque es una herramienta más simple que WebDriver, es útil para crear scripts de prueba rápidamente.
- Selenium Grid: Esta herramienta permite ejecutar pruebas en paralelo en múltiples navegadores y plataformas de forma simultánea. Es especialmente útil para pruebas de compatibilidad multiplataforma.

Ventajas de Selenium

- Flexibilidad y Multiplataforma: Selenium es compatible con múltiples navegadores y sistemas operativos, lo que garantiza la flexibilidad en la automatización de pruebas.
- Lenguajes de Programación: Puede utilizarse con varios lenguajes de programación, incluyendo Java, Python, C#, Ruby y más, lo que brinda a los desarrolladores una amplia gama de opciones.
- Pruebas en Paralelo: Selenium Grid permite la ejecución de pruebas en paralelo, lo que acelera el proceso de prueba y mejora la eficiencia.
- Comunidad Activa: Selenium cuenta con una comunidad activa de desarrolladores y usuarios que proporcionan soporte y recursos, incluyendo extensiones y bibliotecas adicionales.

Desafíos y/o Consideraciones. A pesar de sus ventajas, la automatización de pruebas con Selenium presenta desafíos significativos. Uno de los desafíos comunes es la estabilidad de las pruebas, ya que los cambios en la aplicación pueden hacer que los scripts de prueba fallen. Según Dima Kovalenko, autor de "*Selenium Design Patterns and Best Practices*", la gestión de elementos web dinámicos es otro desafío importante, ya que los scripts deben esperar a que los elementos estén disponibles antes de interactuar con ellos.

- Estabilidad de las Pruebas: La estabilidad de las pruebas es un desafío constante. Las pruebas automatizadas pueden fallar debido a cambios en la estructura de la página web, en los selectores de elementos o en la lógica de la aplicación. Esto requiere un mantenimiento constante de los scripts de prueba para garantizar que sigan funcionando correctamente.
- Elementos Web Dinámicos: Muchas aplicaciones web utilizan elementos dinámicos que se cargan o cambian después de que la página inicial se haya cargado. Identificar y trabajar con estos elementos puede ser complicado, ya que los scripts de prueba deben esperar a que los elementos estén disponibles antes de interactuar con ellos.
- Mantenimiento Continuo: A medida que una aplicación evoluciona, los scripts de prueba también deben actualizarse. El mantenimiento continuo es una tarea que consume tiempo, ya que los cambios en la aplicación pueden requerir modificaciones en los scripts existentes.

Para concluir, Selenium es una herramienta esencial en la automatización de pruebas de aplicaciones web debido a su flexibilidad, compatibilidad y comunidad activa. Su uso eficiente puede mejorar la calidad del software y acelerar el proceso de desarrollo. Sin embargo, es importante abordar los desafíos y consideraciones asociados para garantizar el éxito en la

automatización de pruebas.

JUnit. Es un marco de prueba unitaria de código abierto diseñado para Java. Fue creado por Kent Beck y Erich Gamma en 1997 como parte del movimiento de Pruebas de Desarrollo Guiado por Pruebas (TDD, por sus siglas en inglés) y ha evolucionado con el tiempo para convertirse en la opción preferida por muchos desarrolladores y equipos de desarrollo de software. JUnit se basa en el principio fundamental de escribir pruebas antes de escribir el código de producción, lo que fomenta la detección temprana de errores y la mejora de la calidad del código.

Características Clave de JUnit. Se destaca por varias características clave:

- Anotaciones y Estructura de Pruebas: JUnit utiliza anotaciones como @Test, @Before, @After, y otras para definir y estructurar pruebas. Las anotaciones facilitan la escritura y organización de pruebas de manera clara y legible.
- Assertions: JUnit proporciona un conjunto de métodos de aserción (por ejemplo, assertEquals, assertTrue, assertNotNull) que permiten verificar si los resultados de las pruebas son los esperados. Esto simplifica la verificación de la lógica de las unidades de código. A continuación, se proporciona información sobre cada aserción:
    - assertEquals: Comprueba si dos valores son iguales.
    - assertTrue y assertFalse: Verifican si una expresión booleana es verdadera o falsa, respectivamente.
    - assertNull y assertNotNull: Comprueban si un valor es nulo o no nulo, respectivamente.
    - assertSame y assertNotSame: Comprueban si dos objetos son idénticos o no idénticos.

- Suite de Pruebas: JUnit permite agrupar pruebas en suites, lo que facilita la ejecución de múltiples pruebas de manera ordenada y automatizada.
- Herramientas de Integración: JUnit se integra con una variedad de herramientas de construcción, entornos de desarrollo y sistemas de automatización de compilación. Algunas de las herramientas populares con las que se integra JUnit incluyen Apache Ant, Apache Maven, Gradle y Eclipse. Esta integración facilita la ejecución de pruebas como parte del flujo de trabajo de desarrollo y la generación de informes detallados.

Ventajas de JUnit. Ofrece varias ventajas:

- Detección Temprana de Errores: Al escribir pruebas antes del código de producción, JUnit ayuda a detectar errores de manera temprana, lo que facilita la corrección de problemas en etapas iniciales del desarrollo.
- Mejora de la Calidad del Código: Las pruebas unitarias garantizan que las unidades de código funcionen como se espera, lo que contribuye a un código más confiable y de mayor calidad.
- Documentación Viva: Las pruebas unitarias actúan como documentación viva, proporcionando ejemplos claros del comportamiento de las unidades de código.

Desafíos y Consideraciones. A pesar de sus ventajas, la implementación exitosa de pruebas unitarias con JUnit puede ser un desafío. La gestión de dependencias, la necesidad de escribir pruebas efectivas y la creación de conjuntos de pruebas exhaustivos son áreas que requieren atención.

Según David Saff en "*JUnit in Action*", la escritura de pruebas efectivas es un arte que

requiere un enfoque cuidadoso y conocimiento de las mejores prácticas. Además, el mantenimiento de pruebas a medida que el código evoluciona puede ser un desafío, ya que las pruebas unitarias deben reflejar con precisión el comportamiento actual del código.

- Escritura de Pruebas Efectivas: Escribir pruebas efectivas es un arte que requiere un enfoque cuidadoso y conocimiento de las mejores prácticas. No todas las pruebas son iguales, y la capacidad de escribir pruebas que revelan comportamientos no deseados en el código es fundamental.
- Mantenimiento de Pruebas: A medida que el código evoluciona con el tiempo, las pruebas unitarias también deben actualizarse para reflejar con precisión el comportamiento actual del código. Esto puede ser un desafío en proyectos de gran envergadura.
- Gestión de Dependencias: En proyectos complejos, la gestión de dependencias puede ser un desafío. Es importante aislar adecuadamente las unidades de código bajo prueba para garantizar que las pruebas sean lo más independientes posible.

Robot Framework. El Robot Framework es un marco de automatización de código abierto que se originó en Finlandia y se ha convertido en una opción popular en la automatización de pruebas y RPA en todo el mundo. Este marco se caracteriza por su sintaxis legible y fácil de usar, lo que lo hace adecuado para usuarios con diversas habilidades técnicas lo que permite integrarlo con prácticamente cualquier otra herramienta para crear soluciones de automatización potentes y flexibles. Robot Framework se basa en el principio de escritura de pruebas y tareas en lenguaje natural, lo que fomenta la colaboración entre equipos técnicos y no técnicos, además. Cabe destacar que este marco cuenta con una sintaxis sencilla y utiliza palabras clave legibles

por humanos. Sus capacidades pueden ampliarse mediante bibliotecas implementadas con Python, Java o muchos otros lenguajes de programación.

Características Clave de Robot Framework. El Robot Framework ofrece una serie de características clave que lo hacen atractivo para la automatización de pruebas y tareas de RPA:

- Lenguaje Basado en Tablas: Robot Framework utiliza una estructura basada en tablas para definir casos de prueba. Estas tablas se componen generalmente de tres secciones principales: encabezados, pasos y datos de prueba. Los encabezados describen el tipo de paso (por ejemplo, "Keyword" para acciones o "Expected" para verificaciones), los pasos describen las acciones a realizar y los datos de prueba proporcionan los valores necesarios para realizar las acciones. Esta estructura tabular hace que sea fácil de entender y mantener las pruebas, incluso para personas sin un profundo conocimiento técnico.
- Amplia Biblioteca de Librerías: Una de las fortalezas de Robot Framework es su amplia gama de bibliotecas de librerías. Estas bibliotecas proporcionan palabras clave predefinidas y funcionalidades específicas para una variedad de aplicaciones y sistemas. Las bibliotecas de librerías están disponibles para tareas comunes, como la automatización de pruebas web (por ejemplo, SeleniumLibrary), pruebas de API (por ejemplo, RequestsLibrary), interacción con bases de datos, automatización de escritorio y muchas otras. Esto facilita la automatización de una amplia gama de casos de uso.

- Reusabilidad y Modularidad: El marco promueve la reusabilidad y modularidad al permitir la creación de bibliotecas personalizadas y la importación de funcionalidades de terceros.
- Extensibilidad: Robot Framework se destaca por su capacidad de extensión. Puedes crear tus propias bibliotecas de librerías personalizadas en Python y utilizarlas en tus pruebas. Esto permite adaptar el marco a las necesidades específicas de tu proyecto. También es posible importar funcionalidades de terceros para ampliar la funcionalidad de tu automatización.
- Reportes y Registros Detallados: Robot Framework genera informes detallados después de ejecutar pruebas, lo que facilita la identificación de problemas y la comprensión de los resultados. Estos informes contienen información sobre pruebas exitosas, fallidas y estadísticas sobre la ejecución de pruebas. Los registros detallados son útiles para el diagnóstico y la depuración de problemas en tus pruebas o tareas de RPA.
- Integración con RPA: A través de librerías y extensiones específicas, Robot Framework se utiliza con éxito en proyectos de Automatización Robótica de Procesos (RPA) para automatizar tareas empresariales repetitivas y basadas en reglas.
- Compatibilidad Multiplataforma: Robot Framework es compatible con múltiples sistemas operativos, lo que lo hace versátil y adecuado para proyectos que requieren la automatización en diferentes entornos.

Ventajas de Robot Framework. Las ventajas de utilizar el Robot Framework incluyen:

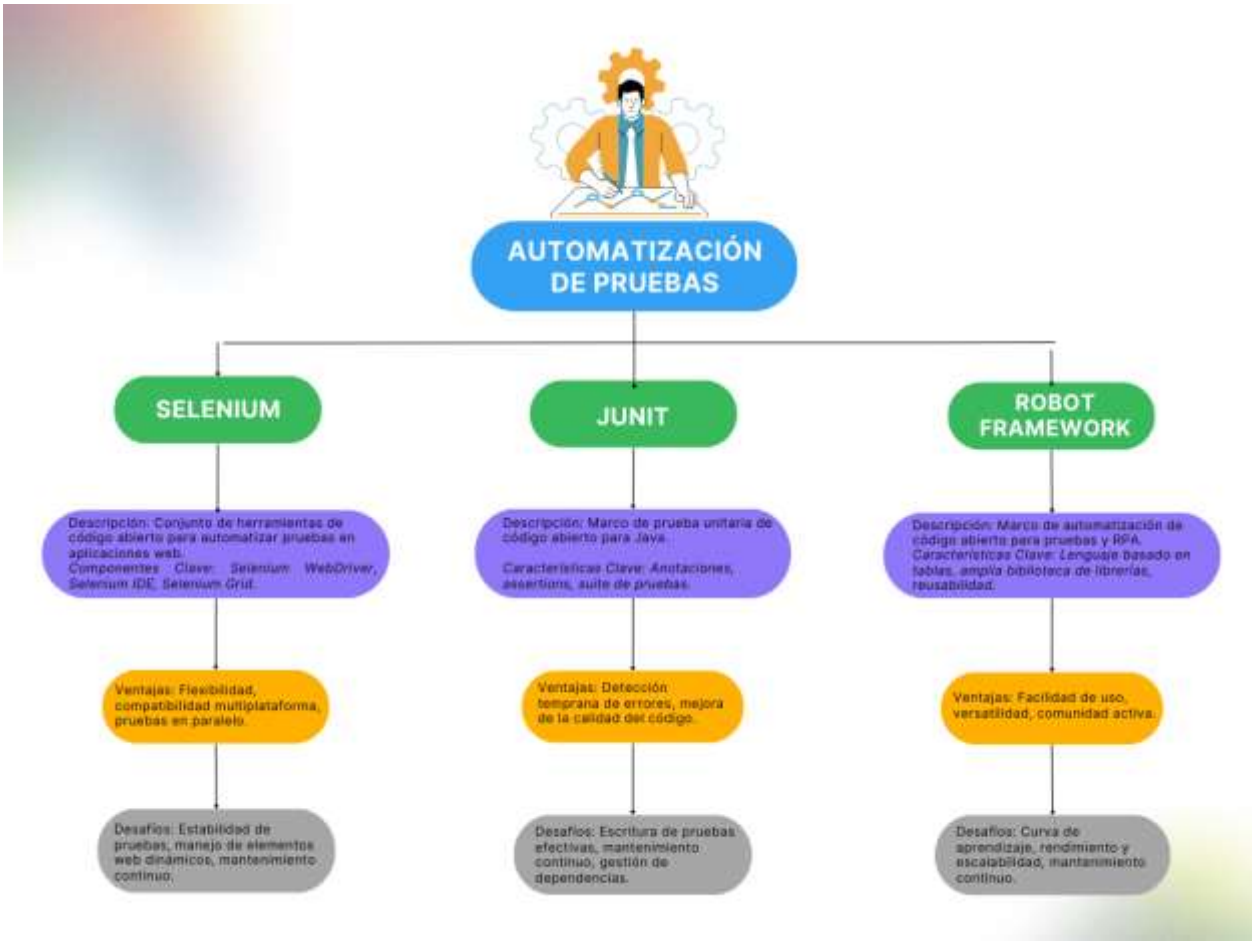
- Facilidad de Uso: La sintaxis basada en tablas y el lenguaje natural hacen que el Robot Framework sea accesible incluso para personas con habilidades técnicas limitadas.
- Amplia Comunidad y Ecosistema: El Robot Framework cuenta con una comunidad activa de usuarios y una amplia gama de librerías y extensiones disponibles, lo que facilita la implementación en diversos contextos.
- Versatilidad**:** La capacidad del Robot Framework para automatizar pruebas de software y tareas de RPA en una amplia variedad de aplicaciones y sistemas lo convierte en una herramienta versátil.

Desafíos y Consideraciones. Si bien el Robot Framework ofrece numerosas ventajas, también plantea desafíos y consideraciones importantes:

- Curva de Aprendizaje: A pesar de su facilidad de uso, el Robot Framework requiere tiempo para aprender y dominar, especialmente al desarrollar bibliotecas personalizadas o realizar tareas de RPA más complejas.
- Rendimiento y Escalabilidad: En proyectos a gran escala, el rendimiento y la escalabilidad pueden convertirse en desafíos. La optimización y el diseño cuidadoso de las pruebas y las tareas de RPA son fundamentales en estas situaciones.
- Integración con Herramientas Externas: Si bien el Robot Framework se integra con numerosas herramientas, puede ser necesario un trabajo adicional para lograr una integración perfecta en entornos específicos.
- Mantenimiento Continuo: Como en cualquier marco de automatización, el mantenimiento continuo es esencial para garantizar que las pruebas y las tareas de RPA sigan siendo efectivas a medida que evoluciona el software y los procesos empresariales.

Ilustración 3.

Diagrama de bloques sobre herramientas de automatización de pruebas



Nota. En este diagrama de bloques se busca comparar las herramientas de automatización de pruebas, resumiendo sus características clave, ventajas y desafíos en el contexto de la automatización de pruebas.

5. Metodología

Es un proyecto de Investigación Aplicada que se enfoca en identificar, estudiar, seleccionar y adaptar herramientas diversas y un marco de referencia adecuado para construir un prototipo de diseño de metodología para la automatización de aseguramiento y control de calidad de software. A través de una investigación aplicada, se abordará y resolverá un problema existente en la gestión de calidad de software, buscando implementar soluciones concretas y efectivas en un entorno empresarial modelado. El proyecto se plantea mediante un conjunto de actividades que permitan alcanzar cada uno de los objetivos específicos:

Actividad 1: Revisión de herramientas disponibles

En esta etapa, se llevará a cabo una revisión minuciosa de las herramientas disponibles para la automatización de pruebas. Se realizará un análisis detallado de las características, capacidades y limitaciones de las herramientas, con el objetivo de identificar aquellas que mejor se adapten a los requisitos específicos del caso de uso.

Actividad 2: Evaluación de casos de uso

Se procederá a evaluar detalladamente los casos de uso relevantes para la empresa. Esta evaluación permitirá determinar cuáles son las herramientas más apropiadas para satisfacer las necesidades específicas de la organización y garantizar una solución completa y efectiva en términos de automatización de pruebas.

Actividad 3: Diseño de propuesta de metodología funcional

Esta actividad se centrará en el desarrollo de una estructura conceptual sólida organizada por pasos o etapas para la implementación de pruebas automatizadas, inicialmente en una versión prototipada en el área de RoadMaps para la empresa S1ESS. La propuesta de metodología requiere identificar un marco de trabajo (framework), para integrarlo y adaptarlo a la propuesta, que abarque desde la creación de los casos de prueba hasta su ejecución, garantizando una cobertura integral de los procesos de desarrollo de software.

6. Resultados

Elección de la herramienta de automatización

Al momento de definir qué herramienta se seleccionará para ejecutar las pruebas automatizadas según el contexto de la compañía o empresa, inicialmente se debe realizar un estudio exhaustivo de los diversos softwares existentes para estas tareas donde es primordial filtrar la cantidad de dichas herramientas a investigar a un rango de al menos dos o tres opciones viables y que se acoplen a los objetivos de la compañía. Es esencial que dichas herramientas al momento de pasar por dicho filtro se consideren algunos aspectos principales como lo son:

- Requisitos específicos del proyecto
- Curva de aprendizaje
- Integración con otras herramientas
- Comunidad y soporte de usuarios.

Con base a lo mencionado anteriormente se diseña un Diagrama de Ponderación con los criterios y pesos más significativos para cada una de las herramientas seleccionadas hasta este punto (Selenium, JUnit y Robot Framework). En la Ilustración 4 encontramos el cuadro de ponderación donde se seleccionan algunos criterios a considerar al momento de elegir la herramienta más idónea con su respectivo valor en porcentajes, cabe resaltar que dichos porcentajes pueden variar según la importancia que la compañía le asigne a cada criterio descrito. Además, encontramos las respectivas casillas para asignar los puntajes según el criterio para cada herramienta; en este escenario los puntajes se asignan de 0 a 9 (siendo 0 el menor y 9

el mayor puntaje posible). Finalmente encontramos la casilla de **"Total"** donde se consigna el valor final representado en porcentajes cuyo cálculo se realiza mediante la fórmula:

Puntuación Total Ponderada = $(\sum_{i=1}^{n}$ *(Puntuación$_i$ X Peso$_i$)) / Peso Total*

Ilustración 4.

Diagrama de ponderación

	Facilidad de Uso (25%)	Flexibilidad (20%)	Capacidad de Mantenimiento (15%)	Escalabilidad (15%)	Soporte de la Comunidad (15%)	Integración con Herramientas (10%)	Total
Selenium	6	8	7	7	9	9	74.5%
JUnit	8	6	8	6	8	8	73%
Robot Framework	8	9	8	9	7	8	82%

Nota. Esta imagen representa la ponderación para la elección de la herramienta de automatización idónea, realizada en la herramienta Lucidchart.

En este proceso de elección del producto más viable (*Ilustración 3*) se encuentra que para Selenium se tiene un 74.5% de idoneidad, para JUnit un 73% y para Robot Framework un 82%. Se llegan a dichos resultados revisando documentaciones sobre cada una de las herramientas primordialmente las consignadas en sus páginas principales y en algunos sitios donde se encuentran comentarios y opiniones (foros) sobre ellas. Como se menciona al iniciar esta sección de resultados esta puntuación no necesariamente implica elegir la herramienta con el puntaje más alto, sino que por el contrario esto sirve como un punto de partida para seleccionar la herramienta más adecuada. Al momento de definir la selección de dicha herramienta en base al

puntaje total es importante tener presente las siguientes consideraciones:

- Requisitos Específicos del Proyecto: Define los requisitos particulares del proyecto, como las características específicas necesitadas, los entornos con los que debe ser compatible, los tipos de pruebas que se planean realizar, entre otros. Es importante resaltar que esto se ajusta y varía según la compañía.
- Curva de Aprendizaje: Se debe evaluar la facilidad con la que el equipo del área de pruebas puede aprender e incorporar la herramienta. Además, se considera la complejidad de la sintaxis, la disponibilidad de recursos de aprendizaje, la familiaridad del equipo con la tecnología, entre otros.
- Integración con Otras Herramientas: Examina la capacidad de la herramienta para integrarse con otras herramientas utilizadas en el entorno empresarial, como lo son herramientas de desarrollo, sistemas de control de versiones, plataformas de CI/CD, entre otras.
- Comunidad y Soporte de Usuarios: Se investiga la comunidad en torno a la herramienta, la disponibilidad de documentación, foros de usuarios, actualizaciones regulares y el nivel de soporte ofrecido por los desarrolladores o la comunidad. Una comunidad de usuarios bien gestionada sirve como un recurso de soporte adicional para los usuarios.

En base al puntaje obtenido en la ponderación y considerando que la compañía maneje tecnologías de desarrollo como .NET y Blazor encaminadas al desarrollo de aplicaciones web y el uso de bibliotecas como RequestsLibrary para interactuar con APIs y servicios web la elección más idónea en este escenario sería Robot Framework, el cual permitiría la automatización de

pruebas de servicios RESTful. Robot Framework como herramienta de automatización otorga flexibilidad y una alta capacidad para integrarse y acoplarse con una gran variedad de herramientas y tecnologías de desarrollo. Como implica la integración de herramientas entre sí es importante apoyarse de la documentación de la página oficial para tener claro las librerías y comandos necesarios para su comunicación y configuración.

Ejecución de un escenario de prueba automatizada en Robot Framework

Dentro de este apartado de resultados presentamos un ejemplo, el cual es bastante conciso el cual proporciona resultados que permiten observar lo potente que es la herramienta Robot Framework y la automatización de pruebas, en este ejemplo se simula el proceso de registro de usuario en la página **https://demo.nopcommerce.com/** utilizando el navegador Google Chrome. El objetivo de esta prueba es validar el proceso de registro de un nuevo usuario en el sitio web indicado. La prueba constó de tres etapas: abrir el navegador web, llenar el formulario de registro y validar el registro; dentro de la sintaxis de Robot Framework encontramos que las etapas o casos de prueba se identificaran con la palabra reservada **“Keywords”**.

Ilustración 5.

Librería "SeleniumLibrary" y variables usadas

```
*** Settings ***
Library      SeleniumLibrary


Load in Interactive Console
*** Variables ***
${urlPage}              https://demo.nopcommerce.com/
${chrome}               chrome
${btnRegister}          css:div.header-links a.ico-register
${formRegister}         css:div.page-body div.form-fields
${rbGender}             css:input[name="Gender"]
${rbGenderM}            //*[@id="gender-male"]
${rbGenderF}            //*[@id="gender-female"]
${txtFName}             //*[@id="FirstName"]
${txtLName}             //*[@id="LastName"]
${selDBD}               css:select[name="DateOfBirthDay"]
${selDBM}               css:select[name="DateOfBirthMonth"]
${selDBY}               css:select[name="DateOfBirthYear"]
${txtEmail}             //*[@id="Email"]
${txtCompany}           //*[@id="Company"]
${chkNewsLetter}        //*[@id="Newsletter"]
${txtPass}              //*[@id="Password"]
${txtConPass}           //*[@id="ConfirmPassword"]
${btnSaveRegister}      //*[@id="register-button"]
${msgVer}               css:div.registration-result-page div.result
```

Nota. Antes de iniciar la codificación de las fases en las que se llevará a cabo la prueba, se realiza la importación o llamada de la librería "SeleniumLibrary". A continuación, se declaran variables esenciales para la ejecución de la prueba, como se muestra en la imagen.

Luego de realizar el proceso anterior se procede a codificar cada una de las etapas de la prueba, en este caso se siguieron las siguientes etapas.

Etapa 1: Abrir Navegador Web

Ilustración 6.

Codificación de la Etapa 1

```
Abrir navegador web
    [Documentation]    Se está abriendo el navegador web chrome

    Open Browser    ${urlPage}  ${chrome}
    Maximize Browser Window
```

Nota. En esta etapa, el navegador web Google Chrome se abrió exitosamente y se maximizó la ventana del navegador.

Etapa 2: Llenar Registro

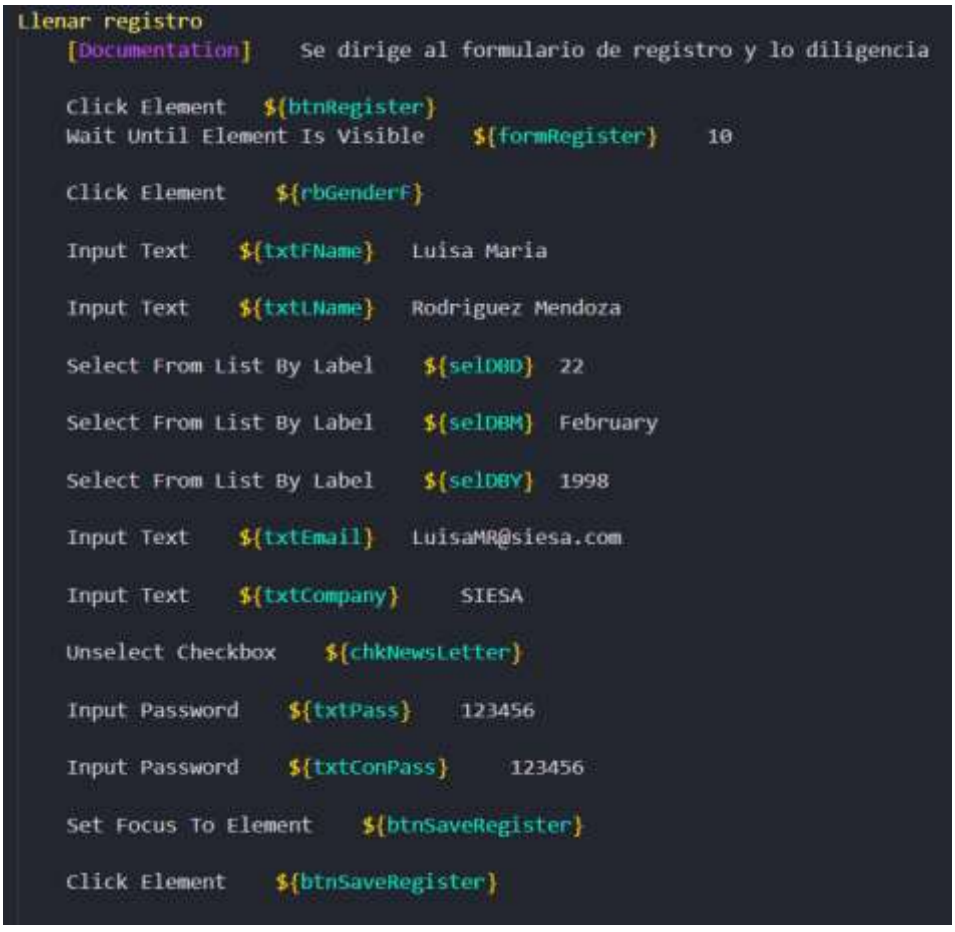
En la segunda etapa, se completó el formulario de registro en el sitio web. Se realizaron las siguientes acciones:

- Se hizo clic en el enlace "Registro" en la barra de navegación.
- Se esperó a que el formulario de registro estuviera visible.
- Se seleccionó el género "Femenino".
- Se ingresó el nombre "Luisa María" en el campo de nombre.
- Se ingresó el apellido "Rodríguez Mendoza" en el campo de apellido.
- Se seleccionó la fecha de nacimiento: Día 22, Mes "Febrero" y Año 1998.
- Se ingresó la dirección de correo electrónico "LuisaMR@siesa.com".
- Se ingresó la empresa "SIESA" en el campo de la empresa.

- Se deseleccionó la casilla de suscripción al boletín.
- Se ingresó la contraseña "123456" en el campo de contraseña.
- Se confirmó la contraseña "123456".
- Se estableció el foco en el botón "Guardar Registro".
- Se hizo clic en el botón "Guardar Registro".

Ilustración 7.

Codificación de la Etapa 2



```
Llenar registro
    [Documentation]    Se dirige al formulario de registro y lo diligencia

    Click Element    ${btnRegister}
    Wait Until Element Is Visible    ${formRegister}    10

    Click Element    ${rbGenderF}

    Input Text    ${txtFName}    Luisa Maria

    Input Text    ${txtLName}    Rodriguez Mendoza

    Select From List By Label    ${selDOD}    22

    Select From List By Label    ${selDOM}    February

    Select From List By Label    ${selDOY}    1998

    Input Text    ${txtEmail}    LuisaMR@siesa.com

    Input Text    ${txtCompany}    SIESA

    Unselect Checkbox    ${chkNewsLetter}

    Input Password    ${txtPass}    123456

    Input Password    ${txtConPass}    123456

    Set Focus To Element    ${btnSaveRegister}

    Click Element    ${btnSaveRegister}
```

Nota. La imagen demuestra la segunda etapa dónde se completó el formulario de registro en el sitio web mediante una serie de acciones anteriormente mencionadas.

Etapa 3: Validar Registro

Ilustración 8.

Codificación de la Etapa 3

```
Validar registro
    [Documentation]    Validando que el registro haya quedado guardado.

    TRY
        Wait Until Element Is Visible    ${msgVer}  10
    EXCEPT
        Fail    ERROR! NO SE GUARDÓ EL REGISTRO!
    END
```

Nota. En la etapa final, se validó que el registro se hubiera completado con éxito. Se esperó a que un mensaje de confirmación estuviera visible. Si el mensaje no se mostraba en un plazo de 10 segundos, se consideraba que la prueba había fallado.

Resultado de la Prueba: Éxito

Ilustración 9.

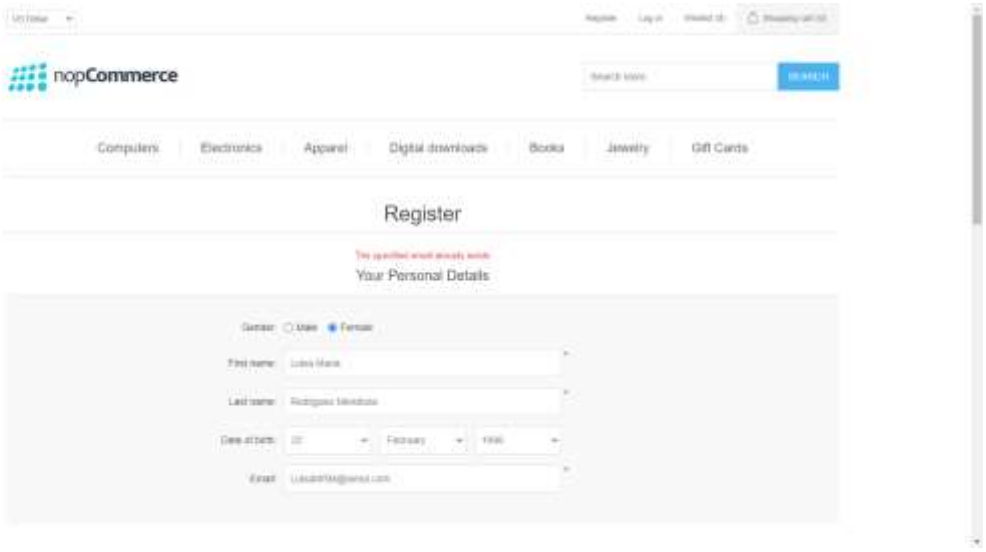
Resultado exitoso al ejecutar la prueba automatizada

Nota. La prueba de registro se completó con éxito. El formulario de registro se llenó correctamente, y se confirmó que el registro se realizó con éxito al mostrar el mensaje de confirmación.

Resultado de la Prueba: Error

Ilustración 10.

Resultado sin éxito al ejecutar la prueba automatizada



Nota. La prueba de registro NO se completó exitosamente. El formulario de registro se llenó con algún(os) datos únicos ya existentes, y se confirmó que el registro NO se realizó con éxito al mostrar el mensaje de error.

Archivos Adicionales

El archivo "log.html" generado por Robot Framework es un informe detallado de la ejecución de tus pruebas. Este archivo es una de las salidas estándar que se generan después de ejecutar un conjunto de pruebas y proporciona información importante para la revisión de los resultados de tus pruebas automatizadas.

El archivo "log.html" contiene varios elementos clave:

Resumen de la Suite de Pruebas

Proporciona un resumen de la suite de pruebas que se ejecutó, incluyendo estadísticas generales como el número de casos de prueba, el número de casos de prueba exitosos y fallidos, la duración de la ejecución, entre otros.

Lista de Casos de Prueba

Los casos de prueba individuales se presentan en una lista con detalles sobre cada caso de prueba, incluyendo su nombre, estado de ejecución (éxito o falla), tiempo de ejecución, enlaces a más detalles y más.

Registro Detallado

Un usuario puede hacer clic en un caso de prueba específico en el archivo "log.html" para visualizar un registro detallado de esa ejecución. Esto incluye información sobre las palabras clave ejecutadas, los argumentos utilizados, los resultados de las palabras clave, y otros detalles de la ejecución.

Capturas de Pantalla

Si el usuario ha configurado la captura de pantalla en los casos de prueba, el archivo "log.html" puede incluir capturas de pantalla que se tomaron durante la ejecución de las pruebas. Estas capturas de pantalla son útiles para identificar visualmente problemas en la interfaz de usuario.

Vista de Historial

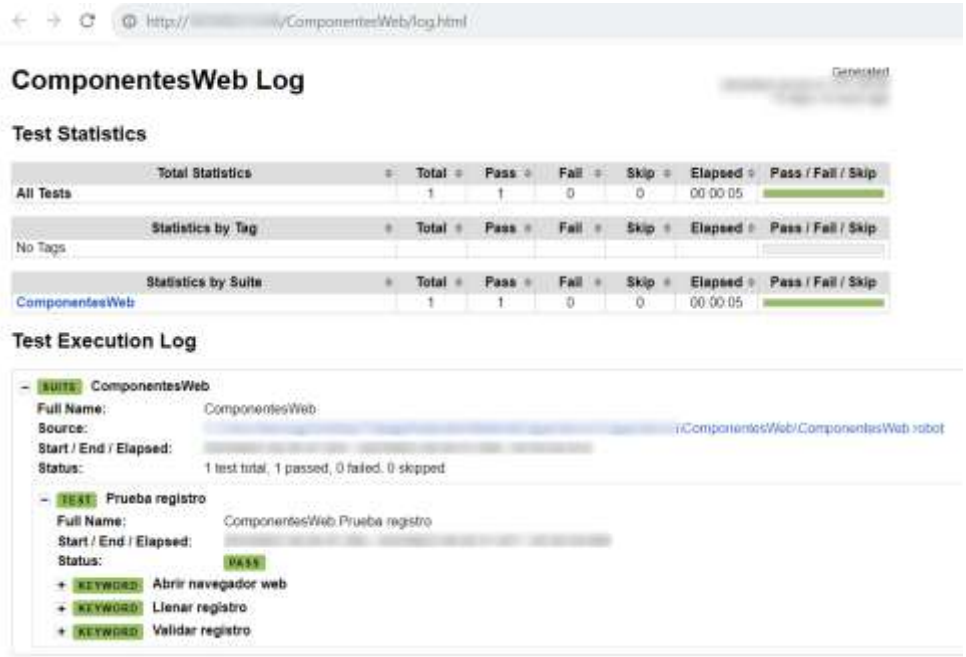
En algunos informes, se presenta un historial de ejecuciones anteriores, facilitando la comparación de resultados a lo largo del tiempo.

Estadísticas y Gráficos

Dependiendo del informe específico, se pueden encontrar estadísticas y gráficos que resumen la ejecución de pruebas de una manera más visual.

Ilustración 11.

Archivo "log.html" generado por Robot Framework



Nota. El archivo "log.html" es una herramienta valiosa para revisar los resultados de las pruebas, identificar problemas y comprender el rendimiento del conjunto de pruebas.

Dentro de este archivo de logs donde se muestran los datos de las pruebas de forma detallada encontramos desglosados los apartados de “Test” y “Keywords”, en los cuales se muestra la información de manera detallada sobre el estado de las pruebas ejecutadas, dentro de estos apartados se puede ver información importante como lo es:

Tests

Proporciona una lista de todos los casos de prueba ejecutados en el conjunto de pruebas. Para cada caso de prueba, se muestra información importante, como:

- ***Name.*** El nombre del caso de prueba.
- ***Status.*** El estado de ejecución del caso de prueba, que puede ser "PASS" (éxito) o "FAIL" (falla).
- ***Message.*** Un mensaje breve que describe el resultado o la razón de la falla si un caso de prueba falló.
- ***Tags.*** Etiquetas asociadas con el caso de prueba, que pueden ayudar a categorizar y filtrar los casos de prueba.
- ***Start Time.*** La hora de inicio de la ejecución del caso de prueba.
- ***End Time.*** La hora de finalización de la ejecución del caso de prueba.
- ***Elapsed Time.*** El tiempo transcurrido durante la ejecución del caso de prueba.

Además, realizando clic en el nombre de un caso de prueba en esta lista se puede acceder a un informe detallado que incluye información sobre palabras clave, argumentos y resultados.

Keywords

Suministra una vista general de las palabras clave utilizadas en los casos de prueba y cómo se ejecutaron. Para cada palabra clave, se muestra información como:

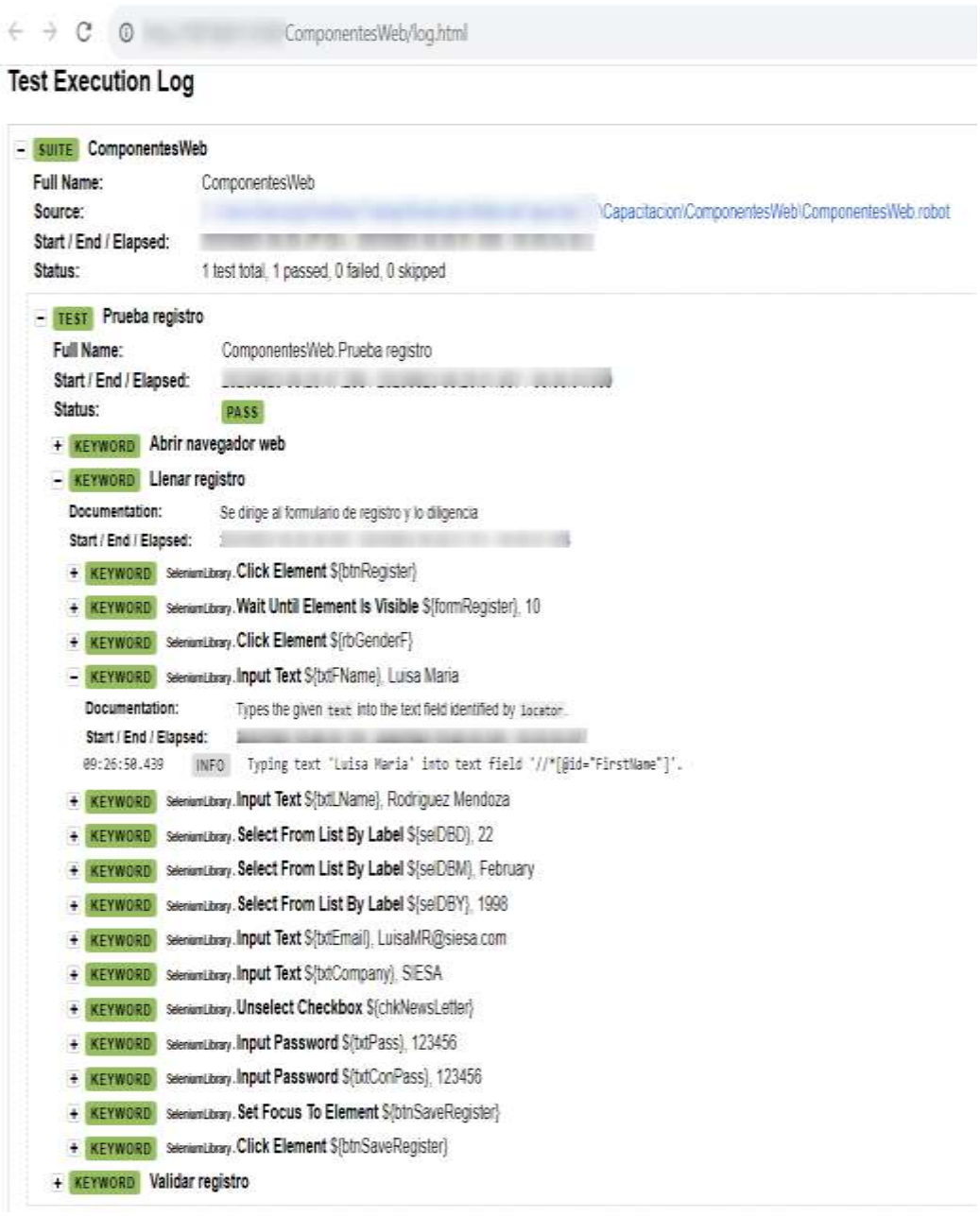
- ***Keyword*.** El nombre de la palabra clave utilizada en el caso de prueba.
- ***Status*.** El estado de ejecución de la palabra clave, que puede ser "PASS" (éxito) o "FAIL" (falla).
- ***Message.*** Un mensaje breve que describe el resultado o la razón de la falla si una palabra clave falló.
- ***Start Time.*** La hora de inicio de la ejecución de la palabra clave.
- ***End Time.*** La hora de finalización de la ejecución de la palabra clave.
- ***Elapsed Time.*** El tiempo transcurrido durante la ejecución de la palabra clave.

Al igual que con la sección *Tests*, se puede hacer clic en el nombre de una palabra clave para acceder a información más detallada sobre la ejecución de esa palabra clave en el contexto del caso de prueba.

Estos apartados son útiles para comprender rápidamente la ejecución de cada una de las pruebas y para acceder a información detallada si es necesario. Usualmente, suelen ser utilizados para identificar problemas, examinar el flujo de ejecución de tus pruebas y obtener una visión general de cómo se comportaron tus casos de prueba y palabras clave.

Ilustración 12.

Apartados Test y Keywords en archivo "log.html" generado por Robot Framework



Propuesta de Metodología Integral

La metodología actual constituye una propuesta derivada de la revisión exhaustiva de los diversos puntos abordados y documentados en el marco teórico. Además, el diagrama de ponderación proporciona una guía para identificar y segmentar cinco etapas cruciales dentro del modelo propuesto. Estas etapas son:

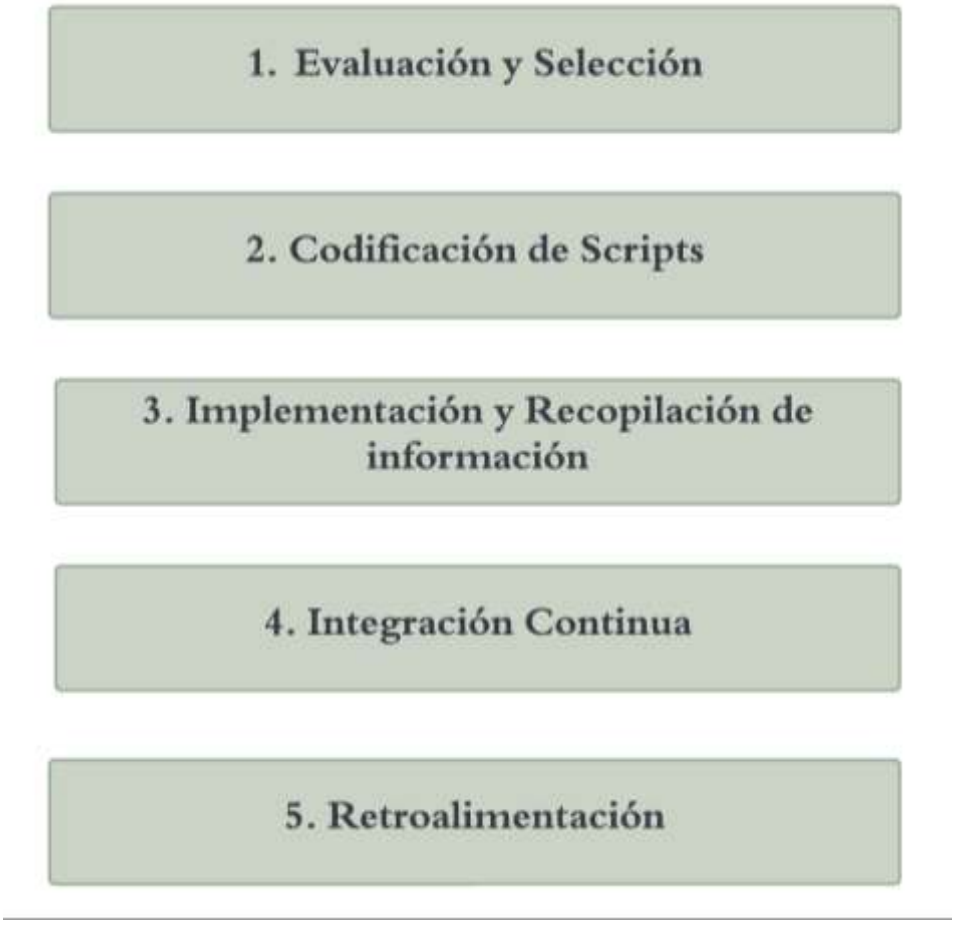
1. Evaluación y Selección
2. Codificación de scripts.
3. Implementación y recopilación de información.
4. Integración continua.
5. Retroalimentación.

Es importante resaltar que esta propuesta busca presentar un enfoque estratégico para llevar a cabalidad la automatización, aseguramiento y control de la calidad, con el objetivo de optimizar el tiempo destinado al proceso de pruebas aumentando así por ende la rentabilidad y eficiencia empresarial.

En la siguiente ilustración encontramos las etapas que tendrá la metodología, posteriormente se presentarán una breve descripción de cada una.

Ilustración 13.

Etapas de la metodología propuesta



1. **Evaluación y Selección**

En esta etapa encontraremos todo lo relacionado a los subprocesos para evaluar las necesidades, la selección de herramientas y la formación del personal involucrado en el proceso de pruebas.

2. Codificación de scripts

En esta etapa encontraremos todos los subprocesos para identificar, codificar y priorizar los casos de prueba, además de desarrollar y validar los scripts de prueba.

3. Implementación y Recopilación de información

En esta etapa encontraremos todos los subprocesos para desplegar la solución de las pruebas automatizadas, realizar pruebas piloto y recopilar la información producto de la ejecución de las pruebas planificadas.

4. Integración continua

En esta etapa encontraremos los subprocesos para implementar el entorno de integración continua y configurar notificaciones automáticas.

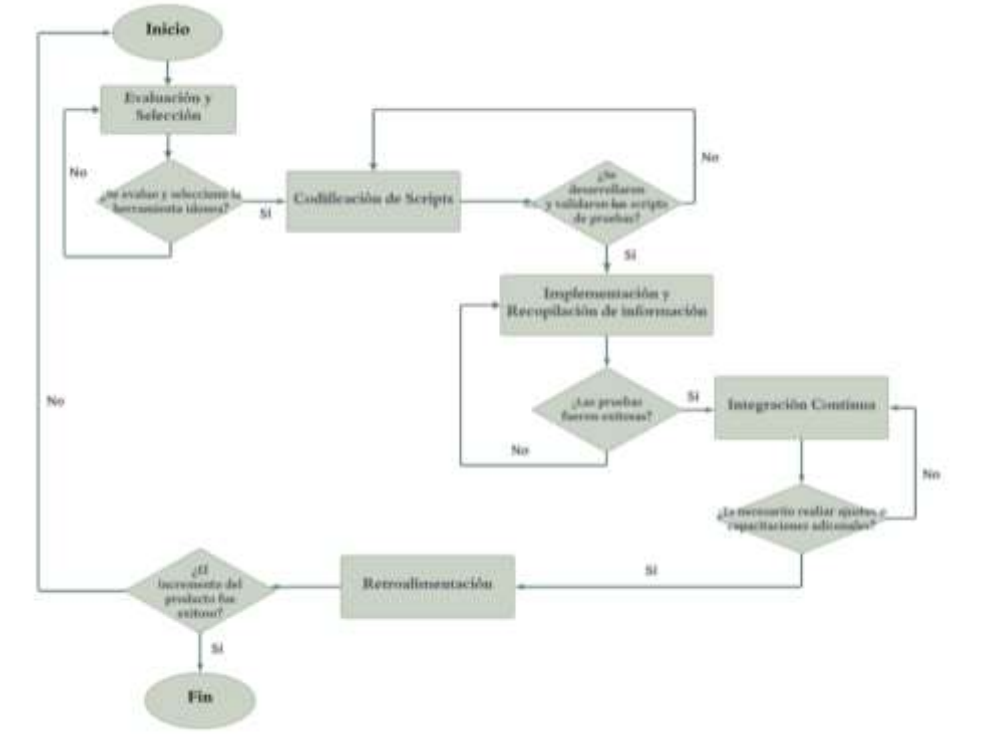
5. Retroalimentación

Esta etapa es el componente crucial de nuestro modelo de implementación de pruebas automatizadas, donde se recopilan y analizan datos sobre el rendimiento del proceso de pruebas. La información obtenida se utiliza para evaluar la eficacia de la implementación, identificar áreas de mejora y realizar ajustes continuos para optimizar aún más el tiempo y la rentabilidad empresarial.

A continuación, se presenta el diagrama de flujo de trabajo provisional para la metodología propuesta. Este diagrama detalla diversas etapas, cada una de las cuales implica una serie de decisiones o métricas condensadas en forma de pregunta. Como se mencionó anteriormente, estas preguntas resumen varios procesos más específicos, los cuales variarán según la compañía donde se pretenda implementar este modelo. Es importante destacar que esta propuesta modelada es de naturaleza genérica, y cada compañía tiene la flexibilidad de realizar los ajustes que considere pertinentes.

Ilustración 14.

Diagrama de flujo de las etapas para la metodología integral



Nota. En el diagrama de flujo anterior, se destacan preguntas tentativas que encapsulan procesos

más detallados en cada etapa. Estas preguntas están vinculadas a bloques de decisión, permitiendo el avance a la siguiente etapa en caso de un resultado positivo. En el caso de un resultado negativo, el avance a la siguiente etapa no será posible.

En el flujo de trabajo presentado, la conclusión se alcanzaría al finalizar satisfactoriamente la etapa de *Retroalimentación*, la cual es la última del proceso. Un resultado positivo en esta fase indicaría que la empresa ha logrado un incremento satisfactorio en el producto, culminando con éxito el ciclo de pruebas y permitiendo la continuación hacia nuevos procesos.

Es crucial tener en cuenta que los resultados esperados o generados por la aplicación de la metodología propuesta son genéricos y pueden variar según la naturaleza del proyecto y las particularidades de la organización. Es esencial medir y evaluar de manera constante cada indicador o resultado obtenido para realizar ajustes y mejoras continuas en la metodología de pruebas automatizadas.

En términos generales, los resultados anticipados al implementar esta propuesta, especialmente en una etapa inicial y basándonos en una simulación en Robot Framework, incluirían:

1. Reducción de Costos: Significativa disminución de los costos asociados con las pruebas manuales, liberando recursos financieros y de personal para otras iniciativas críticas.

2. Aumento de la Calidad del Software: La implementación de pruebas automatizadas, utilizando la metodología propuesta, resultaría en una detección temprana de defectos, contribuyendo a una mejora sustancial en la calidad del software.

3. Aceleración del Tiempo de Entrega: Eliminación de cuellos de botella asociados con pruebas manuales prolongadas, lo que permitiría entregas más rápidas y frecuentes, acelerando el tiempo de desarrollo.

4. Mejora Continua: Establecimiento de retroalimentación continua al final del flujo propuesto mediante métricas de calidad y rendimiento de las pruebas, facilitando ajustes proactivos para una mejora continua.

7. Conclusiones

La relevancia de las pruebas en el desarrollo de software se evidenció de manera sólida a través de los resultados obtenidos en la implementación de Robot Framework para verificar el proceso de registro. Estos resultados motivaron la formulación de una propuesta para una metodología integral que abarca las etapas de evaluación y selección, codificación de scripts, implementación y recopilación de información, integración continua y retroalimentación. La fase final, la retroalimentación, se destaca como crucial para concluir los procesos de manera satisfactoria. Es importante señalar que la principal ventaja para las empresas radica en la optimización del tiempo, ya que la automatización de tareas rutinarias libera a los analistas de pruebas para dedicar más tiempo a pruebas de mayor complejidad, dejando que las pruebas automatizadas se encarguen de las rutas más básicas.

Adicionalmente, la eficiencia en la ejecución, la reproducibilidad confiable y el ahorro de costos a largo plazo emergen como aspectos clave que resaltan la importancia de la automatización en este contexto. La capacidad de lograr una amplia cobertura de pruebas, obtener retroalimentación rápida, escalar eficientemente y mantener una documentación clara y activa son factores adicionales que subrayan la trascendencia de la automatización en el aseguramiento de la calidad del software. La integración continua, la mayor cobertura de código, la facilidad para las pruebas de regresión y la escalabilidad continua de las pruebas representan un horizonte prometedor para el desarrollo y la evolución del proceso de registro, y, por extensión, para el éxito general de proyectos de software.

8. Recomendaciones

Se recomienda realizar pruebas adicionales de este prototipo de metodología en el área de RoadMaps ya que actualmente es una versión beta, con la finalidad de asegurar que el proceso al cual se realiza la prueba dentro de la compañía siga siendo exitoso en futuras versiones del sitio web o aplicativo.

La velocidad de carga del sitio web o aplicativo puede afectar el tiempo de espera necesario para la validación del proceso al completarlo. Es importante considerar este factor en pruebas posteriores.

Adicionalmente, en la selección del servidor para implementar las herramientas de automatización, es esencial considerar que este servidor deberá ser compartido por todos los proyectos dentro de la empresa. Este planteamiento puede generar conflictos al proporcionar servicios directos a clientes que tienen restricciones debido a políticas de seguridad, las cuales prohíben la conexión de servidores que no formen parte de su red.

Se sugiere mejorar la claridad en la comunicación respecto a la designación de asesores de trabajo de grado por parte del área correspondiente de la universidad. Esto para situaciones futuras, será beneficioso al proporcionar una pronta conexión entre los estudiantes y asesores, facilitando un inicio oportuno y eficaz del trabajo durante el semestre.

9. Bibliografías

Cárdenas, O. L. (2016). *AUTOMATIZACIÓN DE CASOS DE PRUEBA PARA MEJORAR EL PROCESO DE CALIDAD DE SOFTWARE*. http://hdl.handle.net/11371/738

Chacón, J. G. (2019). Automatización de Aseguramiento de Calidad, de Encapsulación de Software. *Revista de La Facultad de Ingenierías y Tecnologías de La Información y Comunicación.*, *1*(5), 1–17.

García, A. M. (2015). *LA INTEGRACIÓN CONTINUA Y SU APORTE AL ASEGURAMIENTO DE LA CALIDAD EN EL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE*. http://hdl.handle.net/11349/8276

Huamán, E. C. (2021). *Modelo de automatización de pruebas para optimizar la gestión de la calidad de software en la empresa Cifin*. https://hdl.handle.net/20.500.12867/4708

Lucero, J. M. (2022). *Implementación de un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros*. https://hdl.handle.net/20.500.12672/18814

Rueda, A. A., Cruz Mosquera, H. F., Londoño Rojas, J. A. (2016). *PROPUESTA DE AUTOMATIZACIÓN DE CASOS DE PRUEBA PARA ASEGURAMIENTO DE CALIDAD EN EL DESARROLLO DE SOFTWARE*. https://repository.uniminuto.edu/handle/10656/5647

Maida, E. & Pacienzia, J. (2015). *Metodologías de desarrollo de software.* PONTIFICIA UNIVERSIDAD CATÓLICA ARGENTINA SANTA MARÍA DE LOS BUENOS AIRES. https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf.

Roche, J. (2023). Las 5 ceremonias Scrum: claves para la gestión de procesos. Deloitte. Tomado de: https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html

McGraw (2006). *Seven Touchpoints for Software Security.* Software Security, Building Security In. Tomado de: http://www.swsec.com/resources/touchpoints/

Microsoft. (2023). What are the Microsoft SDL practices? Microsoft. Recuperado de: https://www.microsoft.com/en-us/securityengineering/sdl/practices

Microsoft Learn (2023). *Microsoft Security Development Lifecycle (SDL).* Microsoft Learn. Recuperado de: https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle

Selenium.dev. (2023*). Selenium documentation.* Selenium. Tomado de: https://www.selenium.dev/documentation/en/

Robot Framework Foundation. (2016). *Robot Framework User Guid*e. Robot Framework. Tomado de: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Bechtold et. al. (2023*). JUnit 5 User Guide*. JUnit 5 User Guide. Recuperado de: https://junit.org/junit5/docs/current/user-guide/

Consejos técnicos. (2020). Tipos de pruebas de software: diferencias y ejemplos. loadview. Tomado de: https://www.loadview-testing.com/es/blog/tipos-de-pruebas-de-software-diferencias-y-ejemplos/