



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

**Automatización de pruebas funcionales de software
para optimizar los tiempos de pruebas usando Serenity
BDD en una entidad financiera en el año 2023**

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Carlos Antonio SAN BARTOLOMÉ SANDOVAL

ASESOR

Mg. Armando David ESPINOZA ROBLES

Lima, Perú

2025



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

San Bartolomé, C. (2025). *Automatización de pruebas funcionales de software para optimizar los tiempos de pruebas usando Serenity BDD en una entidad financiera en el año 2023*. [Trabajo de Suficiencia Profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Carlos Antonio San Bartolomé Sandoval
Tipo de documento de identidad	DNI
Número de documento de identidad	72648236
URL de ORCID	https://orcid.org/0009-0009-5449-5460
Datos de asesor	
Nombres y apellidos	Armando David Espinoza Robles
Tipo de documento de identidad	DNI
Número de documento de identidad	08633326
URL de ORCID	https://orcid.org/0000-0001-5671-5927
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Jose Cesar Piedra Isusqui
Tipo de documento	DNI
Número de documento de identidad	25628915
Miembro del jurado 1	
Nombres y apellidos	Cesar Angulo Calderón
Tipo de documento	DNI
Número de documento de identidad	32907109
Datos de investigación	
Línea de investigación	No aplica
Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento

Ubicación geográfica de la investigación	País: Perú Departamento: Lima
	Provincia: Lima Distrito: Cercado de Lima Jr. Carlos Amezaga No. 375 Universidad Nacional Mayor de San Marcos Latitud: -12.0564232 Longitud: -77.0843327
Año o rango de años en que se realizó la investigación	2024
URL de disciplinas OCDE	Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Software

Acta Virtual de Sustentación
del Trabajo de Suficiencia Profesional

Siendo las 20:00 horas del día 10 de enero del año 2025, se reunieron virtualmente los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Mg. Piedra Isusqui José César (Presidente), Mg. Angulo Calderón César Augusto (Miembro) y el Lic. Espinoza Robles Armando David (Miembro Asesor), usando la plataforma Meet (<https://meet.google.com/jtg-ystx-bsz>), para la sustentación virtual del Trabajo de Suficiencia Profesional intitulado: **“AUTOMATIZACIÓN DE PRUEBAS FUNCIONALES DE SOFTWARE PARA OPTIMIZAR LOS TIEMPOS DE PRUEBAS USANDO SERENITY BDD EN UNA ENTIDAD FINANCIERA EN EL AÑO 2023”**, por el Bachiller **San Bartolomé Sandoval Carlos Antonio**; para obtener el Título Profesional de Ingeniero de Software.

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó al Bachiller a dar las respuestas a las preguntas establecidas por los miembros del Jurado.

El Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el Bachiller obtuvo la nota de **17 (DIECISIETE)**.

A continuación el Presidente de Jurados el Mg. Piedra Isusqui José César , declara al Bachiller **Ingeniero de Software**.

Siendo las 20:57 horas, se levantó la sesión.

Presidente

Mg. Piedra Isusqui José César

Miembro

Mg. Angulo Calderón César August

Miembro Asesor

Mg. Espinoza Robles Armando David



CERTIFICADO DE SIMILITUD

Yo, ESPINOZA ROBLES ARMANDO DAVID en mi condición de asesor acreditado con la Resolución Decanal N° 001059-2024-D-FISI/UNMSM y Oficio N° 0014-PTTSP-FISI-2024 del Trabajo de Suficiencia Profesional cuyo título es Automatización de pruebas funcionales de software para optimizar los tiempos de pruebas usando Serenity BDD en una entidad financiera en el año 2023, presentado por el bachiller SAN BARTOLOMÉ SANDOVAL CARLOS ANTONIO para optar el título de Ingeniero de Software CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el 11% de similitud, nivel **PERMITIDO** para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional**.

Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del título.

Lima, 8 de enero de 2025

Firma del Asesor

Nombres y Apellidos del asesor: ESPINOZA ROBLES ARMANDO DAVID

DNI 08633326

DEDICATORIA

Este Trabajo de Suficiencia Profesional
está dedicado a mi familia.

A mis padres por inculcarme valores y
alentarme a perseguir mis metas.

A mis hermanas por su todo su apoyo.

AGRADECIMIENTO

Agradezco a mi asesor, el Mg. Armando David Espinoza Robles, por su orientación en la elaboración del presente Trabajo.

A los profesores de la Facultad de Ingeniería de Sistemas e Informática de la UNMSM por sus enseñanzas que utilice en mi experiencia profesional.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERIA DE SOFTWARE

Automatización de pruebas funcionales de software para optimizar los tiempos de pruebas usando Serenity BDD en una entidad financiera en el año 2023

Autor: San Bartolomé Sandoval, Carlos Antonio
Asesor: Mg. Espinoza Robles, Armando David
Título: Trabajo de Suficiencia Profesional para optar el Título Profesional de Ingeniero de Software
Fecha: Diciembre de 2024

RESUMEN

La automatización de pruebas funcionales basadas en una aplicación web para una entidad financiera es el enfoque de este trabajo de suficiencia profesional, a fin de mejorar la calidad del proceso de pruebas.

La entidad financiera busca mejorar sus procesos en el desarrollo de software, los cuáles consumen mucho tiempo y esto se traduce a costos altos. En búsqueda de dichas mejoras se empezó a impulsar la integración continua y entrega continua, y fomentar más iniciativas que apoyen en estos procesos.

Como solución se implementó la automatización de pruebas funcionales usando Serenity BDD como parte del flujo de pruebas continuas en las aplicaciones web de la División de Riesgos de la entidad financiera.

En los logros obtenidos del presente trabajo, se redujo los tiempos de pruebas funcionales. Antes de la solución, un analista de calidad podía tomar 12 horas en la etapa de pruebas; con la automatización, el tiempo de pruebas funcionales se redujo a 2 horas y sin intervención del analista durante la ejecución.

Palabras clave: Automatización de pruebas, pruebas continuas, Serenity BDD, pruebas funcionales

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS
FACULTY OF SYSTEMS AND INFORMATICS ENGINEERING
PROFESSIONAL SCHOOL OF SOFTWARE ENGINEERING

**Automation of software functional testing to optimize testing times using Serenity
BDD in a financial institution in 2023**

Author: San Bartolomé Sandoval, Carlos Antonio
Advisor: Mg. Espinoza Robles, Armando David
Title: Professional Sufficiency Work to opt for the Professional Title of
Software Engineer
Date: December 2024

ABSTRACT

The automation of functional tests based on a web application for a financial institution is the focus of this work of professional proficiency, in order to improve the quality of the testing process.

The financial institution seeks to improve its software development processes, which are time consuming and result in high costs. In search of such improvements, the company started to promote continuous integration and continuous delivery, and to encourage more initiatives to support these processes.

As a solution, functional test automation was implemented using Serenity BDD as part of the continuous test flow in the web applications of the Risk Division of the financial institution.

In the achievements obtained from this work, functional testing times were reduced. Before the solution, a quality analyst could take 12 hours in the testing stage; with the automation, the functional testing time was reduced to 2 hours and without intervention of the analyst during the execution.

Keywords: Test automation, continuous testing, Serenity BDD, functional testing.

TABLA DE CONTENIDO

RESUMEN	v
INTRODUCCIÓN.....	1
CAPÍTULO I – TRAYECTORIA PROFESIONAL.....	3
CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA .	6
2.1 Empresa – Actividad que realiza	6
2.2 Visión.....	6
2.3 Misión	6
2.4 Organización de la empresa	6
2.5 Área, cargo y funciones desempeñadas	8
2.6 Experiencia profesional realizada en la organización.....	8
CAPÍTULO III – ACTIVIDADES DESARROLLADAS	9
3.1 Situación Problemática	9
3.1.1 Definición del problema.....	10
3.2 Solución	10
3.2.1 Objetivos	10
3.2.2 Alcance.....	10
3.2.3 Etapas y metodología	11
3.2.4 Fundamentos utilizados.....	13
3.2.5 Implementación de las áreas, procesos, sistemas y buenas prácticas.....	27
3.3 Evaluación.....	55
3.3.1 Evaluación económica / Evaluación costo-beneficio.....	55
3.3.2 Interpretación del VAN y del TIR.....	57
CAPITULO IV – CRÍTICA DE LA EXPERIENCIA	59
CAPITULO V – CONCLUSIONES Y RECOMENDACIONES	60
5.1 Conclusiones	60
5.2 Recomendaciones	60
Fuentes de información	62
Glosario	64

ÍNDICE DE TABLAS

Tabla 1: Experiencia laboral.....	3
Tabla 2: Formación académica.....	5
Tabla 3: Formación complementaria.....	5
Tabla 4: Casos de pruebas convenientes y no convenientes para automatizar.....	15
Tabla 5: Ventajas y desventajas de la técnica de pruebas de caso de uso	16
Tabla 6: Comparación entre pruebas automatizadas y pruebas manuales.....	17
Tabla 7: Prerrequisitos de la automatización.....	29
Tabla 8: Herramientas utilizadas en la automatización	30
Tabla 9: Descripción de carpetas y archivos del proyecto de automatización	31
Tabla 10: Comandos de Maven.....	41
Tabla 11: Plugins de Jenkins	48
Tabla 12: Regla de colores de cabeceras en Excel de garantía.....	49
Tabla 13: Comandos básicos de Git	54
Tabla 14: Comparación de tiempo de prueba manual y automatizado para 90 registros	55
Tabla 15: Presupuesto estimado del desarrollo del caso con pruebas manuales	55
Tabla 16: Presupuesto estimado del desarrollo del caso con pruebas automatizadas	56
Tabla 17: Inversión inicial estimada.....	57
Tabla 18: Flujo de caja del primer año	58
Tabla 19: Flujo de efectivo neto proyectado a 5 años	58

ÍNDICE DE FIGURAS

Figura 1: Organigrama de la empresa.....	7
Figura 2: Organigrama de la división según roles	7
Figura 3: Sintaxis de Gherkin.....	22
Figura 4: Reporte de Serenity BDD	23
Figura 5: Incidencia de tipo prueba	24
Figura 6: Incidencia de tipo ejecución de prueba	25
Figura 7: Ejemplo de diseño de sistema de automatización.....	26
Figura 8: Proyecto de automatizaciones de la división	28
Figura 9: Estructura del proyecto de automatización	29
Figura 10: Automatizaciones planificadas	32
Figura 11: Tablero de planificación trimestral	33
Figura 12: Repositorio de la automatización del sistema	34
Figura 13: Escenario en Cucumber	35
Figura 14: Carpeta stepdefinitions.....	36
Figura 15: Clase LoginSteps.java.....	36
Figura 16: Clase LoginPage.java.....	37
Figura 17: Clase RealizarLogin.java	38
Figura 18: Versión del navegador	39
Figura 19: Webdriver del navegador.....	39
Figura 20: Archivo serenity.conf.....	40
Figura 21: Carpeta target	40
Figura 22: Archivo Data-CartaFianza.xls.....	42
Figura 23: Carpeta data.....	42
Figura 24: Escenario en Cucumber con lectura de Excel	43
Figura 25: Archivo LeerCartaFianzaExcel.java	43
Figura 26: Archivo EscribirCartaFianzaExcel.java.....	44
Figura 27: Nuevo método de generación de carta fianza	45
Figura 28: Clase CucumberTestSuite con integración de Jira XRay	46
Figura 29: Incidencia de ejecución de pruebas en Jira	47
Figura 30: Dashboard de Jenkins.....	48
Figura 31: Carpeta ejecutarScripts	49

Figura 32: Script de tubería AgregarDatosGarantia	50
Figura 33: Script de tubería GenerarGarantia.....	51
Figura 34: Configuración de tubería AgregarRegistroGarantiaCalendarizado	51
Figura 35: Configuración de tubería GenerarGarantiaCalendarizado	52
Figura 36: Email con resultado fallido de la ejecución de la automatización	53

INTRODUCCIÓN

En el sector financiero, la competencia se basa en la calidad del servicio brindado a los clientes y su satisfacción, para ello se busca ser más eficiente en sus procesos de manera que puedan entregar valor para garantizar esta calidad. Estas mejoras en sus procesos pueden ser la reducción de tiempos o costos sin perder la calidad de entrega o incluso mejorando la misma.

Este trabajo se enfoca en la automatización de pruebas funcionales como parte de la mejora en el proceso de desarrollo interno de software, reduciendo tiempos de ejecución de las pruebas y mejorando la calidad al reducir posibles errores humanos.

La solución se llevó a cabo usando Serenity BDD que es un marco de trabajo basado en Selenium para automatizar pruebas funcionales de soluciones Web, el cual simula la interacción de un usuario y proporciona reportes con un lenguaje natural.

Adicionalmente se llevó la ejecución de las pruebas automatizadas a tuberías de Jenkins en un servidor de pruebas para ser utilizadas remotamente, liberando recursos del analista de calidad.

En este trabajo se demuestra como la automatización de pruebas funcionales reduce los tiempos en la fase de pruebas de software, reduce costos y mejora la calidad en las entregas.

En el desarrollo de este trabajo, se explora el procedimiento de la implementación de pruebas automatizadas, partiendo de la estructura de su elaboración hasta la adopción del flujo de pruebas continuas.

Al final, el presente trabajo es una guía para que otros equipos u organizaciones que busquen optimizar sus tiempos en sus procesos de pruebas lo tomen en consideración, ya que también prepara al equipo en la evolución de los procesos de software.

A continuación, se detalla la estructura del presente trabajo:

- En el CAPÍTULO I, se detalla la experiencia laboral y formación académica del autor, las cuáles servirán de base para mostrar el perfil del mismo.
- En el CAPÍTULO II, se describe el contexto en donde se llevó el presente trabajo, detallando datos de la organización, así como las funciones desempeñadas por el autor en su rol dentro de la misma.

- En el CAPÍTULO III, se desarrolla la descripción del problema y su solución, detallando los objetivos, alcance, metodología e implementación de dicha solución, y finalmente la evaluación de la misma.
- En el CAPÍTULO IV, se expone la reflexión crítica del autor sobre la experiencia descrita.
- En el CAPÍTULO V, se detalla las conclusiones y recomendaciones del presente trabajo.

CAPÍTULO I – TRAYECTORIA PROFESIONAL

Soy un profesional con más de cuatro años de experiencia en el campo de software, siendo desarrollador por dos años y medio y automatizador de pruebas por dos años, desempeñándome en diferentes consultoras en mi trayectoria.

Inicialmente me desempeñé como desarrollador de software, brindando soluciones a startups en una consultora, teniendo como funciones el desarrollo frontend y backend, y el despliegue de las soluciones al mercado. También tuve la experiencia de desarrollar en una plataforma de desarrollo de código bajo como Outsystems. Posteriormente, dentro de la entidad financiera tuve la oportunidad de cambiar del desarrollo frontend al desarrollo de pruebas automatizadas.

Actualmente me desempeño como automatizador de pruebas de soluciones móvil y API para un producto nuevo impulsado por otra entidad financiera. Soy parte de un grupo cross funcional, el cual plantea y promueve un nuevo lineamiento en la automatización de pruebas e impulsa la adopción de pruebas continuas en el proceso de software.

Tabla 1
Experiencia laboral

Empresa	Cargo	Funciones	Periodo
Globant Cliente: BCP	Test Automation Engineer, SSr	• Automatización de pruebas en web con Serenity BDD y Screenplay. • Automatización de pruebas de API con Karate DSL. • Automatización de pruebas en móvil con Appium. • Integración de reportes con Jira. • Configuración de tuberías en Jenkins. • Pruebas de seguridad con OWASP. • Elaboración de plan de pruebas y ejecución de pruebas.	Abril 2024 – Actualidad

Tata Consultancy Services Cliente: Interbank	IT Analyst (QA)	• Automatización de pruebas en web con Serenity BDD y Screenplay. • Integración de reportes con Jira XRay. • Configuración de tuberías en Jenkins y Azure Pipelines. • Versionamiento con Bitbucket • Pruebas de rendimiento con JMeter - Logros: • Impulsar que los analistas de calidad de los equipos automaticen sus pruebas.	Julio 2022 – Abril 2024
Tata Consultancy Services Cliente: Interbank	IT Analyst (Developer)	• Mantenimiento y desarrollo de features en aplicación web en Angular y aplicación móvil en Ionic.	Enero 2022 – Junio 2022
Readiness IT	Programador Junior	• Desarrollo de web tradicionales y reactivas usando Outsystems bajo el marco de arquitectura Canvas.	Mayo 2021 – Diciembre 2021
Cereza Consulting	Desarrollador Fullstack	• Desarrollo de aplicaciones web en Angular. • Desarrollo de aplicaciones móviles en Ionic. • Desarrollo de aplicaciones backend en NodeJS. • Manejo de estados con Rxjs • Diseño de base de datos relaciones y MongoDB. • Implementación y consumo de API REST y GraphQL • Configuración de ambientes productivos.	Febrero 2020 – Mayo 2021

Nota Elaboración propia

Tabla 2

Formación académica

Programa	Institución	Grado	Periodo
Ingeniería de Software	Universidad Nacional Mayor de San Marcos	Bachiller	2013 – 2021
Diplomatura de Especialización en Desarrollo de Aplicaciones con Inteligencia Artificial	Pontificia Universidad Católica del Perú	Diplomatura	Setiembre 2021 – Marzo 2022

Nota Elaboración propia

Tabla 3

Formación complementaria

Programa	Institución	Tipo	Año
Foundation Agile Tester Extension	Calidad de Conceptos	Curso	Marzo 2024
ISTQB Certified Tester Foundation Level	ISTQB	Certificado	Febrero 2024
Outsystems Associate Reactive Developer	Outsystems	Certificado	Noviembre 2021
Scrum Foundation Professional Certificate	Certiprof	Certificado	Febrero 2020
Inglés Avanzado	ICPNA	Curso	Febrero 2015

Nota Elaboración propia

CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA

2.1 Empresa – Actividad que realiza

La entidad financiera donde se desarrolló la experiencia es una de las principales instituciones financieras del país, brindando productos innovadores y diferentes servicios a millones de clientes. Tiene como prioridad la transformación digital por lo que cuenta con una plataforma digital de productos y servicios para atender sus millones de clientes, y contando con colaboradores especializados para mantener la plataforma.

2.2 Visión

Dicha entidad financiera busca ser la mejor entidad a partir de las mejores personas. Buscando siempre personas motivadas, que disfruten lo que hacen, para que juntas creen soluciones y experiencias innovadoras.

2.3 Misión

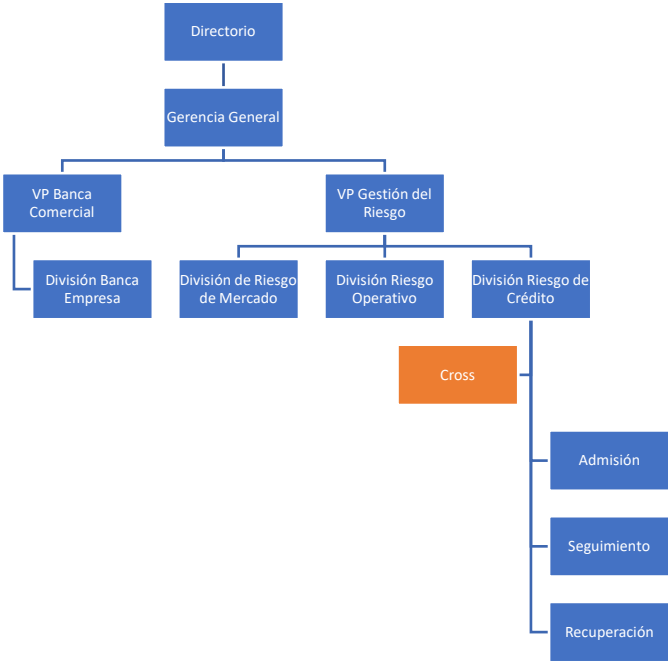
Dicha entidad financiera, hoy, se encuentra acompañando a sus clientes peruanos a cumplir sus sueños. Brindando diferentes productos según las necesidades del cliente, y ofreciendo servicios que tienen al cliente como el centro de todo.

2.4 Organización de la empresa

La entidad financiera cuenta con varias vicepresidencias, las cuales cuentan con divisiones. Se presenta la figura 1 con la información disponible por el autor a la fecha que desempeñe funciones para dicha entidad financiera. Adicionalmente se presenta la figura 2, la cual describe los roles dentro de la división Riesgo de Crédito, visualizando la relación entre negocio y tecnología.

Figura 1

Organigrama de la empresa



Nota Elaboración propia del extracto organizacional de la entidad financiera

Figura 2

Organigrama de la división según roles



Nota Elaboración propia del extracto organizacional de la entidad financiera

2.5 Área, cargo y funciones desempeñadas

El autor del presente trabajo ocupa el cargo de automatizador de pruebas como parte del equipo Cross, puesto impulsado por el líder analista de calidad y por el dueño del producto de la división Riesgo de Crédito para dar apoyo a los cuatro equipos dentro de la división, desempeñando las siguientes funciones:

- Mantenimiento de los scripts automatizados existentes de los equipos.
- Desarrollo de nuevos escenarios automatizados según la criticidad del negocio.
- Apoyar a los analistas de calidad de cada equipo para que comprendan y desarrollen pruebas funcionales automatizadas.
- Impulsar nuevas iniciativas en torno a la automatización de pruebas.

2.6 Experiencia profesional realizada en la organización

El autor desempeñó un papel de soporte para toda la división, teniendo como objetivo el agregar valor con las pruebas continuas. Para ello realizó las siguientes actividades:

- Refactorización del proyecto de scripts automatizados, organizándolo en repositorios por soluciones de cada equipo.
- Elaboración de pruebas automatizadas con Serenity BDD y patrón de diseño Screenplay.
- Alinear los proyectos de automatización de la división, para que todos los equipos manejen una estructura similar, y un mismo lineamiento.
- Agregar archivos externos de data para reducir la manipulación del código y guardarlos como log de resultados de las ejecuciones de las pruebas automatizadas.
- Conexión a Jira XRay para subir los reportes de evidencias de la ejecución de las pruebas automatizadas.
- Configuración de Jenkins en un servidor de pruebas, estructurando el dashboard según los grupos, añadiendo conexión a los repositorios, salida de correos y backups.
- Creación de tuberías para la ejecución de las pruebas automatizadas, y agendarlas para monitorear la estabilidad de los sistemas en el ambiente de pruebas.
- Generación de reportes semanales en Kibana a partir de data obtenida por canales oficiales para obtener métricas de la división usando un script automatizado.

CAPÍTULO III – ACTIVIDADES DESARROLLADAS

3.1 Situación Problemática

Las pruebas funcionales de regresión están presentes en todo desarrollo, nuevo o mantenimiento. Lucero (2022) menciona que, al tomar evidencias de manera manual en las pruebas funcionales, estas toman mucho tiempo y esfuerzo, donde un sprint es insuficiente, reduciendo la cobertura de pruebas y aumentando el riesgo de defectos en el sistema en consecuencia. Las pruebas de software siempre ha sido una actividad de alto consumo de tiempo por más experto que sea el equipo de calidad, las industrias optan por la automatización para reducir el periodo de desarrollo (Kishan, Pavithra, 2021)

En la entidad financiera, dentro de la división de riesgos de crédito se identificó un consumo elevado de tiempo en pruebas funcionales de la web. Una fase de pruebas podía durar entre 8 a 12 horas dependiendo del módulo del sistema en desarrollo.

En una fase inicial la división había iniciado la automatización de pruebas, sin embargo, por temas de tiempo, la iniciativa fue despriorizada. El resultado fue un costo mayor en el tiempo, ya que, con nuevos desarrollos, la cantidad de pruebas aumentó y los tiempos iniciales de la fase de pruebas que oscilaban entre 2 a 4 horas, ahora superaban las 8 horas.

Las aplicaciones de la división no solo eran para uso dentro de la misma, sino que era de apoyo para otras grandes divisiones dentro de la entidad financiera como Banca Persona y Banca Empresa, por lo que el impacto en tiempos retrasaba los desarrollos en otros equipos.

La criticidad de los sistemas era otro factor a considerar, por ello las pruebas eran demandantes y exhaustivas; y con solo un analista de calidad por equipo, el recurso humano podía verse agotado, lo que resultaba en una cantidad mínima de errores humanos en las pruebas, pero existente.

En este punto, el líder analista de calidad propone volver a priorizar la iniciativa de automatización de pruebas ya que había estimado que las fases de pruebas podrían tardar entre 16 y 24 horas en el plazo de un año por el rápido crecimiento de las aplicaciones, considerándolo un número alarmante de horas.

3.1.1 Definición del problema

En la entidad financiera, el desarrollo de software actual incurre en altos tiempos en los ciclos de pruebas manuales, costos elevados y calidad a mejorar de los entregables, lo que afecta a la división de Riesgo de Crédito y otros equipos en las divisiones en Banca Empresa y Banca Persona. Por lo que se va a implementar las pruebas automatizadas.

3.2 Solución

3.2.1 Objetivos

3.2.1.1 Objetivo general

Implementar la automatización de pruebas funcionales usando Serenity BDD y patrón de diseño Screenplay, bajo la metodología SCRUM para optimizar los tiempos de pruebas en el desarrollo de software en la entidad bancaria.

3.2.1.2 Objetivos específicos

- Crear proyectos de automatización de las aplicaciones web, elaborando como base un lineamiento interno.
- Integrar la automatización con Jira XRay para enviar los reportes de resultados de pruebas automatizadas.
- Configurar Jenkins en un servidor de pruebas, creando tuberías para las ejecuciones de scripts automatizados.

3.2.2 Alcance

3.2.2.1 Alcance funcional

El trabajo se limita al equipo de Garantías de la división de Riesgo de Crédito de la entidad financiera, evaluando los escenarios de registro de Carta Fianza y registro de Hipoteca para los casos de una y noventa ejecuciones.

3.2.2.2 Alcance técnico

Las herramientas utilizadas son Serenity BDD como marco de trabajo para el desarrollo de la automatización, el patrón de diseño Screenplay y redacción de escenarios en Cucumber. Se definió Java como lenguaje de programación y Maven como gestor de proyectos. Se usó los navegadores Microsoft Edge y Google Chrome. Los datos de entrada provienen de una hoja de cálculo Excel, los repositorios de versiones se ubican en Bitbucket y se seleccionó Jenkins como servidor de automatizaciones.

3.2.2.3 Alcance organizacional

Los equipos impactados por el desarrollo son el equipo de Garantía, la división de Riesgo de Crédito y algunos equipos de la división de Banca Persona. Dentro de los equipos se encuentran los líderes técnicos y de negocio, desarrolladores y analistas de calidad, donde el principal usuario son los analistas. Los procesos de pruebas se ven alterados, contando con una deuda técnica para casos no automatizados de ser automatizables y nuevos lineamientos de seguridad con respecto a la exposición de credenciales, data sensible y vulnerabilidades del servidor de automatizaciones. Finalmente, se agrega la función de automatizar a los analistas de calidad de la división de Riesgo de Crédito.

3.2.2.4 Alcance geográfico

El desarrollo del trabajo se limita a la ubicación de la entidad financiera al ser un desarrollo interno usado solamente por los propios colaboradores del sitio.

3.2.3 Etapas y metodología

Se detallará las tareas en las fases correspondientes al marco de trabajo Scrum.

- Inicio: En esta fase se presenta al grupo el cual se le asignará puntos de evolución tecnológica según lo acordado con el líder analista de calidad y el dueño del producto, según una evaluación preliminar de automatizaciones existentes en el proyecto de automatizaciones de la división.

- Planificación: Durante esta fase se da lugar a reuniones con el analista de calidad y líder técnico del equipo para refinar el puntaje según la cantidad de automatizaciones obsoletas a refactorizar y nuevas que se requieran. Además de acordar reuniones con el analista para aprender de la automatización.
- Implementación: Esta fase comienza al iniciar el sprint y las tareas se encuentran divididas en fases:
 - Etapa 1: Extraer escenarios correspondientes al equipo desde el proyecto de automatizaciones de la división, refactorizarlas y llevarlas al proyecto de Serenity BDD con patrón Screenplay.
 - Etapa 2: Crear automatizaciones de escenarios críticos faltantes.
 - Etapa 3: Tomar data de archivos externos, como entrada en la automatización y como log de resultado final de la ejecución.
 - Etapa 4: Conectar a Jira XRay para subir los reportes Serenity de la ejecución local de la automatización.
 - Etapa 5: Implementar tuberías en Jenkins, para crear data de entrada y ejecutar la automatización en el servidor de pruebas.
 - Etapa 6: Validar la ejecución agendada por un par de semanas para monitorear el comportamiento de la automatización, envío de notificaciones por correo y levantar observaciones.
- Lanzamiento: Finalmente en esta fase se hace la demostración final y reporte del monitoreo al líder analista de calidad y el analista calidad del equipo y se hace una comparación con el caso de mayor tiempo del equipo. Se realiza la transferencia al analista de calidad, levantando el proyecto, indicando los pasos para ejecutar, detallando los escenarios existentes y una guía para la implementación de nuevos escenarios.

El marco de trabajo usado por la entidad financiera era Scrum. Sin embargo, este presenta un caso particular al ser parte del equipo Cross.

La distribución de puntaje en los tableros se divide en: desarrollo, subsanación de la deuda técnica y evolución tecnológica; los dos primeros se refieren a un flujo regular de desarrollo de software mientras que el último agrupa las tareas de nuevas iniciativas, siendo este el caso para la automatización de pruebas, y estas tareas pueden existir por más de un sprint. Adicional a esto, los equipos celebran las cuatro ceremonias en sprints de dos semanas, y trimestralmente una retrospectiva y planeamiento a nivel de toda la

división; sin embargo, el equipo Cross solo es requerido en la retrospectiva y planeamiento trimestral; solo por el nivel de involucramiento con el equipo correspondiente también fue requerido la participación en las reuniones diarias.

3.2.4 Fundamentos utilizados

3.2.4.1 Fundamentos teóricos

Pruebas de software

Las pruebas de software es el proceso integral en el desarrollo que evalúa y verifica un programa, garantizando su correcto funcionamiento y el cumplimiento de los requisitos establecidos. Siendo las pruebas de gran importancia para detectar errores y asegurar la calidad del software. (Delgado, 2024)

Los tipos de pruebas son diversas, dependiendo del artefacto a probar, la etapa en que se encuentra y lo que se busca probar de este artefacto.

Pesantes (2023) lista algunos tipos de pruebas:

- Prueba unitaria: Prueba que valida el correcto funcionamiento de una unidad de código, la cual puede ser una clase, función o método.
- Prueba de integración: Prueba que valida el comportamiento de la interacción de dos o más componentes sea el esperado.
- Prueba de regresión: Prueba realizada después de un cambio en el sistema para validar que características previas no fueron afectadas por dicho cambio.
- Pruebas de humo: Prueba rápida que busca defectos en los flujos críticos del sistema tras una modificación del software.
- Prueba de aceptación: Prueba que valida que se cumpla un escenario, caso de uso o caso de negocio.
- Prueba de sistema: Prueba que, basada en condiciones específicas, valida el cumplimiento de la funcionalidad del sistema.

También se emplea técnicas de diseño de casos de pruebas para crear una estrategia de pruebas.

Toan (2024) divide las técnicas en tres tipos:

- Técnicas de pruebas de caja negra: Se centran en el comportamiento externo del sistema, ignorando cómo funciona internamente. Las técnicas son:

- Partición equivalente: Separa los valores de entrada en grupos y espera que el sistema responda igual para cualquier valor dentro de un mismo grupo, usualmente los grupos son de valores válidos e inválidos.
- Análisis de valores límites: Evalúa el comportamiento del sistema al ingresar valores que se encuentren en la frontera de los grupos.
- Tabla de decisiones: Cuando existen varias condiciones para determinar el comportamiento del sistema, estas se representan en una tabla para validar el resultado esperado de las combinaciones.
- Pruebas de caso de uso: Evalúa el software desde la perspectiva de un usuario y su interacción con el sistema.
- Diagrama de transición de estado: Usada cuando el sistema tiene diferentes estados, con eventos que desencadenan la transición de un estado a otro.
- Técnicas de pruebas de caja blanca: Conocidas como técnicas de pruebas de estructuras, se centran en examinar el código del software. Las técnicas son:
 - Cobertura de sentencias: Técnica para probar que se haya ejecutado cada línea de código del software.
 - Cobertura de ramas: Técnica que prueba que todas las posibles rutas de ejecución hayan sido exploradas.
 - Cobertura de rutas: Técnica que asegura que todas las rutas posibles han sido ejecutadas.
 - Pruebas de flujo de datos: Técnica que examina como la data viaja a través del código, para validar el uso correcto de las variables.
- Técnicas basadas en la experiencia: Depende de la experiencia y conocimiento del analista de calidad en base a experiencias previas. Las técnicas son:
 - Intuir el error: Se basa en la experiencia del analista de calidad con defectos similares del sistema a probar.
 - Prueba exploratoria: Prueba adaptable, que no sigue un lineamiento fijo y se desarrolla mientras el analista de calidad prueba el sistema.
 - Prueba de usabilidad: Evalúa como el usuario final interactúa con el sistema e identifica los puntos donde decae la experiencia de dicho usuario.

Pruebas de regresión

Las pruebas de regresión son planeadas tras cada desarrollo completado, ya que no solo se debe validar que funcione lo nuevo, sino que también se debe validar que el funcionamiento del sistema no haya sido alterado. Debido a que el desarrollo es incremental, parte de las nuevas pruebas van agregándose al plan de pruebas de regresión, aumentando el tiempo de ejecución de estas pruebas.

En la tabla 4 se observa los casos de pruebas óptimos para automatizar, donde muestra que las pruebas de regresión es uno de estos casos.

Tabla 4

Casos de pruebas convenientes y no convenientes para automatizar

Casos de pruebas convenientes	Casos de pruebas no convenientes
Pruebas que deben ejecutarse en cada parte del sistema	Pruebas que solo se realizan una vez
Pruebas que utilizan múltiples entradas de datos para la misma acción	Pruebas de usabilidad
Pruebas similares en diferentes plataformas	Pruebas que no tienen resultados definidos
Pruebas de regresión	Casos de pruebas que nunca se han probado manualmente
Casos de pruebas críticos del negocio	Sistemas que cambian sus requerimientos frecuentemente
Casos de pruebas que se ejecutan repetitivamente	Ejecución de casos de pruebas particulares
Casos de pruebas que toman mucho tiempo	

Nota Tomado de A Comprehensive Study on Automated Testing with the Software Lifecycle (p. 6) por Ali H., Hamza M. y Rashid T., 2023

Técnica de pruebas de caso de uso

Al probar flujos funcionales de la web, los analistas de calidad hacen uso de la técnica de pruebas de caso de uso interactuando con el sistema tomando la perspectiva del usuario final.

Horina y Oleksandr (2024) describen como en BDD los casos de uso reflejan las acciones y resultados esperados de las características del software y estos son descritos en lenguaje plano con Gherkin.

Debido a ello, la técnica de pruebas de caso de uso es la más compatible en la automatización ya que las propias herramientas describen los escenarios de una característica del software, y con Serenity BDD dichos escenarios se narran tomando de referencia al actor.

En la tabla 5 se muestra las ventajas y desventajas del uso de la técnica de pruebas de caso de uso y evaluando estos puntos se determina que las desventajas no afectan a las pruebas de automatización desarrolladas.

Tabla 5

Ventajas y desventajas de la técnica de pruebas de caso de uso

Ventajas de pruebas de caso de uso	Desventajas de pruebas de caso de uso
Ayuda a entender el sistema con mayor precisión y claridad	Solo cubren requerimientos funcionales, no puede usarse para no funcionales
Representa la secuencia de pasos describiendo la interacción entre usuarios y sistema	Los casos de uso se escriben desde el punto de vista del usuario, dejando de lado escenarios que vienen del propio sistema
Simplifica las complejidades del sistema al enfocarse en una tarea a la vez	
El equipo analista de calidad toma la perspectiva del usuario final, encontrando fácilmente defectos de la experiencia	
Los casos de uso detallan precondiciones, postcondiciones, reglas de negocio, flujo, excepciones, entre otros, que ayudan a diseñar casos de pruebas	

Nota Adaptado de What Is Use Case Testing: [With Examples] por Ahamed I., 2023

Automatización de pruebas

La automatización de pruebas es el apoyo que se da a las actividades de prueba mediante el uso del código, dando a entender que es una herramienta en el proceso de

pruebas y no un fin. Dicho esto, la automatización se ve beneficiada de una buena estrategia de pruebas además de un código bien estructurado con el uso de patrones.

Estas pruebas también dependen de su uso, ya que hay un mayor beneficio en reutilizar las pruebas, ya que debe retornar la inversión de haber elaborado la automatización en las futuras ejecuciones de pruebas. Sin embargo, en el contexto de un ambiente de desarrollo inestable o en deprecación, no habría beneficio en la automatización, sino que hasta sería perjudicial.

Se detallan los beneficios de la automatización de pruebas en la Tabla 6, demostrando como la automatización mejora el proceso de pruebas de software.

Tabla 6

Comparación entre pruebas automatizadas y pruebas manuales

Pruebas automatizadas	Pruebas manuales
Ahorra dinero	Cuesta más dinero
Toma menos tiempo	Toma más tiempo
Confiable	No es tan confiable
Reutilizable y con menos esfuerzo humano	La detección de defectos y errores es menor a las pruebas automatizadas
Ayuda a mejorar la calidad de las pruebas y la eficiencia del software	La mejora de calidad es inferior al de las pruebas automatizadas
Lanzamiento temprano	Lanzamiento tardío comparando con la automatización
Al poder usar las herramientas en simultáneo, la cobertura aumenta y las plantillas pueden ser reutilizadas	Las personas pueden agotarse al hacer tareas repetitivas, demandando más tiempo en hacerlas.

Nota Tomado de A Comprehensive Study on Automated Testing with the Software Lifecycle (p. 3), por Ali H., Hamza M. y Rashid T., 2023

Desarrollo Guiado por Comportamiento (BDD)

Los equipos están compuestos por diferentes roles, cada uno comunicándose con su propio léxico, los cuales pueden generar malentendidos con otros roles que no manejen el mismo lenguaje. Es por ello que debe haber un lenguaje en común para la comprensión

de todo el equipo en artefactos importantes, como los criterios de aceptación, el cual se usa de base para la elaboración de pruebas.

BDD busca que los analistas de negocios, desarrolladores de software y analistas de calidad colaboren más estrechamente al poder expresar los requisitos de una manera más comprobable, de una forma que todos puedan comprender fácilmente el negocio. Las herramientas de BDD ayudan a que los requisitos se conviertan en pruebas automatizadas que guían al desarrollador, verificando y documentando el funcionamiento de la aplicación. (Pesantes, 2023)

Horina y Oleksandr (2024) narran como BDD tiene como antecedentes el desarrollo guiado por pruebas (TDD) y el desarrollo guiado por pruebas de aceptación (ATDD), tomando beneficios de ambas metodologías y teniendo una mejor adaptación al enfoque ágil del desarrollo de software.

Una de las herramientas de la metodología de BDD en la automatización es el lenguaje de Gherkin para redactar los escenarios.

Pruebas continuas

Las pruebas continuas llegan como la fase final de la implementación de la automatización de pruebas en este trabajo. Después de ejecutar la automatización de manera local, el siguiente paso es ejecutarlo de manera remota en un servidor de pruebas.

El proceso de automatización de pruebas permite la ejecución continua de las pruebas en cualquier entorno y momento según la configuración y a este proceso se le denomina pruebas continuas. Este proceso tiene como objetivos mejorar los procesos de automatizaciones, disminuir el tiempo de pruebas y entregas del software, y optimizar el esfuerzo de la implementación de pruebas. (Duquino, 2020)

Scrum

Sutherland y Schwaber (2020) describen a Scrum como un marco de trabajo ligero que ayuda a equipos y organizaciones a generar valor. La adopción es simple al solo definir lo necesario para implementar la teoría de Scrum, centrándose más en guiar las relaciones e interacciones entre las personas que en instrucciones detalladas. Entre sus definiciones más importantes se encuentran:

- Equipo Scrum: La unidad base de Scrum, el cual se define en un grupo de personas que unidas se enfocan en el objetivo del producto. El grupo se divide esencialmente en tres roles:
 - Dueño de producto: Es el propietario del producto quien se responsabiliza en maximizar el valor del producto y la gestión del trabajo pendiente del producto.
 - Maestro Scrum: Es el responsable que el equipo lleve Scrum como es debido, orientando a los otros roles a poner en práctica Scrum de la manera más eficaz.
 - Desarrolladores: Son las personas con las habilidades para crear cualquier aspecto de un incremento en cada sprint.
- Eventos de Scrum: Eventos presentes en un Sprint cuya finalidad es asegurar la transparencia de inspeccionar y adaptar los artefactos, y reducir reuniones innecesarias
 - Sprint: Evento de duración fija, el cual empieza un nuevo sprint tras el término del anterior sprint y busca cumplir los objetivos del producto en plazos cortos. Los demás eventos están contenidos en este.
 - Planeación del sprint: Al iniciar el sprint, el equipo se reúne para definir las tareas que pueden realizarse en el sprint y el objetivo que se busca cumplir en dicho sprint.
 - Scrum diario: Reunión diaria que inspecciona los avances hacia el objetivo del sprint y de ser necesario adaptar las tareas planificadas.
 - Revisión del sprint: El equipo presenta en una reunión los resultados del trabajo a los principales interesados y el progreso hacia el objetivo del producto.
 - Retrospectiva del sprint: Última reunión del sprint donde el equipo discute maneras de mejorar la calidad y eficiencia en base a lo análisis del desarrollo del último sprint.
- Artefactos: Representan trabajo o valor, y son diseñados para transparentar la información al tener la misma base. Cada uno es medible con compromisos.
 - Trabajo pendiente del producto: Lista de lo que es necesario para mejorar el producto. Se enfoca en el objetivo de producto y las tareas se crean a partir de cómo se completará dicho objetivo.

- Trabajo pendiente del sprint: Lista que se basa en un reducido grupo de tareas del trabajo pendiente del producto, las cuáles en su conjunto deben enfocarse en un objetivo para el sprint y un plan para entregar el incremento en el plazo determinado.
- Incremento: Es un hito hacia el cumplimiento del objetivo del producto. Cada incremento se adiciona por encima de los incrementos anteriores y debe proveer valor por lo que se define una definición de hecho para saber si se cumplió el trabajo con calidad.

3.2.4.2 Fundamentos metodológicos

Pruebas automatizadas

Ayasta (2022) utilizó el marco de trabajo Serenity BDD para disminuir el tiempo de ejecución de pruebas de software en aplicaciones web. En su proyecto definió los tipos de pruebas de regresión y de humo para etiquetar los escenarios de pruebas. La implementación lo realizó en un repositorio privado, utilizando el patrón Page Object Model y bajo la metodología Kanban. Define que las pruebas consideradas deben cubrir los flujos críticos del negocio y deben ser repetitivas, demandantes y no sufran modificaciones. En su desarrollo, Ayasta define los pasos de las pruebas con Cucumber, usando Java como lenguaje de programación y al final ejecuta las pruebas en Gitlab CI/CD. Como resultado, Ayasta redujo el tiempo de ejecución de pruebas entre un 74% a 99%.

Pesantes (2023) se plantea como hipótesis la implementación de automatización de pruebas de aceptación en un sistema web. Empleó herramientas como Jira para la obtención de los planes de pruebas y sus casos, Cucumber para redactar los escenarios y Ruby como lenguaje de programación. En su análisis, Pesantes toma como variables el número de errores promedio y el tiempo promedio de pruebas, obteniendo resultados favorables según el tamaño del proyecto con resultados de una reducción del 11% a un 39% en tiempos promedio de pruebas y una reducción del 23% a un 28% en errores humanos.

Pruebas de software en Scrum

Pal y Karakostas (2022) resaltan la importancia de las pruebas de software en el marco de trabajo Scrum. Mencionan que, en un enfoque tradicional de desarrollo de software, la fase de pruebas es potencialmente afectada por la fase de desarrollo ya que se ve limitada por el tiempo restante después del fin del desarrollo. Ellos destacan el uso de Scrum, un marco de trabajo ágil, que se alinea con el enfoque ágil actual de las industrias y donde una de las premisas de Scrum es proveer entregas incrementales de calidad por lo que las pruebas están presentes para asegurar la calidad. Adicionalmente se implementa la automatización de pruebas para optimizar la eficiencia en la fase de pruebas. Finalmente, Pal y Karakostas concluyen que adoptar las pruebas ágiles da como beneficios el ahorro de tiempo y dinero, la reducción de la documentación, la flexibilidad y adaptabilidad al cambio, y una manera de retroalimentarse del usuario final.

Delgado (2024) uso Scrum para el desarrollo de su trabajo de automatización de pruebas, siguiendo las fases de inicio, planificación e implementación. Empleó herramientas como un tablero Kanban para mostrar los elementos desarrollados en un sprint, Jira Xray para crear los elementos relacionados con la fase de pruebas y Gherkin para redactar los escenarios, además de cumplir con eventos de Scrum. Delgado concluye que haber implementado Scrum permitió respuestas más rápidas al cambio al facilitar la integración de pruebas al desarrollo además de reducir los tiempos de entregas.

3.2.4.3 Fundamentos técnicos

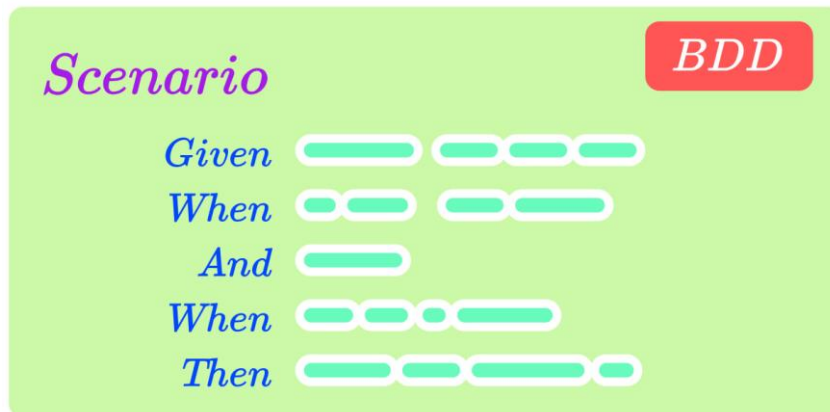
Gherkin

Horina y Oleksandr (2024) definen Gherkin como un lenguaje para articular pasos y escenarios de pruebas. Su sintaxis facilita la representación del comportamiento del software, asegurándose que los escenarios se redacten de manera estandarizada, eliminando ambigüedades y facilitando la comunicación precisa.

La base del lenguaje está en sus palabras clave, mostrándose las más usuales en la figura 3, y dichas palabras también pueden ser llevadas al idioma local de tener la preferencia.

Figura 3

Sintaxis de Gherkin



Nota Tomado de Advantages of Automated Testing of Medical Applications and Information Systems using Gherkin and Behavior-Driven Development (p. 8) de Horina K. y Oleksandr K., 2024

Cucumber

Cucumber es una herramienta que soporta BDD, la cual lee las especificaciones ejecutables escritas en texto plano y validando que el software haga lo que las especificaciones indiquen.

Al seguir las sintaxis de Gherkin, dichas especificaciones consisten en escenarios, donde cada uno es un conjunto de pasos que se conectan con el código y Cucumber verifica su cumplimiento, generando reportes de éxito o fallo de cada escenario. (Cucumber, s.f.)

Serenity BDD

Serenity BDD es una librería de código libre que ayuda a escribir pruebas automatizadas de regresión de manera más limpia y mantenibles. También crea reportes indicando que escenarios o características se han probado y el paso a paso de la ejecución de cada automatización como se ve en la Figura 4.

Figura 4

Reporte de Serenity BDD

Steps	Screenshots	Outcome	
Given Colin has logged onto the application		SUCCESS	660ms
Given Colin has the following items in his cart:			
Sauce Labs Backpack		SUCCESS	684ms
Sauce Labs Bolt T-Shirt			
When he checks out his cart providing his personal details		SUCCESS	4s 022ms
he opens the https://www.saucedemo.com/checkout-step-one.html		SUCCESS	725ms
he provides personal details: <code>CustomerDetails[firstName=Colin, lastName=MacColin, postCode=123-ABC]</code>		SUCCESS	3s 281ms
he enters 'Colin' into 'First Name' field		SUCCESS	949ms
he enters 'MacColin' into 'Last Name' field		SUCCESS	744ms
he enters '123-ABC' into 'Zip/Postal Code' field		SUCCESS	821ms
he clicks on 'Continue' button		SUCCESS	756ms

Nota Reporte de Serenity BDD sobre el resultado de la ejecución de una automatización, indicando el paso a paso con capturas de pantalla. Tomado de Serenity BDD, s.f.

Serenity BDD tiene como base Selenium WebDriver el cual también es una herramienta de código abierto utilizada para el desarrollo de pruebas automatizadas de aplicaciones web. Ambos soportan diferentes navegadores donde se ejecutarán las pruebas, utilizan localizadores para encontrar e interactuar con los elementos web y pueden ser desarrollados en una amplia selección de lenguajes de programación. Entre las mejoras que ofrece Serenity BDD se encuentran, una mayor abstracción que facilita la codificación y mantenimiento, adopción del BDD y generación de reportes.

Serenity BDD promueve el uso del patrón Screenplay el cual está orientado en el usuario como se suele narran los escenarios de BDD empleando herramientas como Cucumber. Además, facilita la integración con herramientas externas de gestión de pruebas como Jira donde puede almacenar los reportes.

Jira

Jira es una herramienta de gestión de proyectos ágiles usada para planificar, seguir, desplegar y mantener software. Se mantiene como la única fuente de verdad en el ciclo de desarrollo de software para que los equipos se mantengan conectados. (Atlassian, s.f.)

Dentro de las herramientas en Jira, estas se categorizan según la necesidad del usuario, y para el caso del presente trabajo se emplea Xray, el cual apoyará en la fase de pruebas de software.

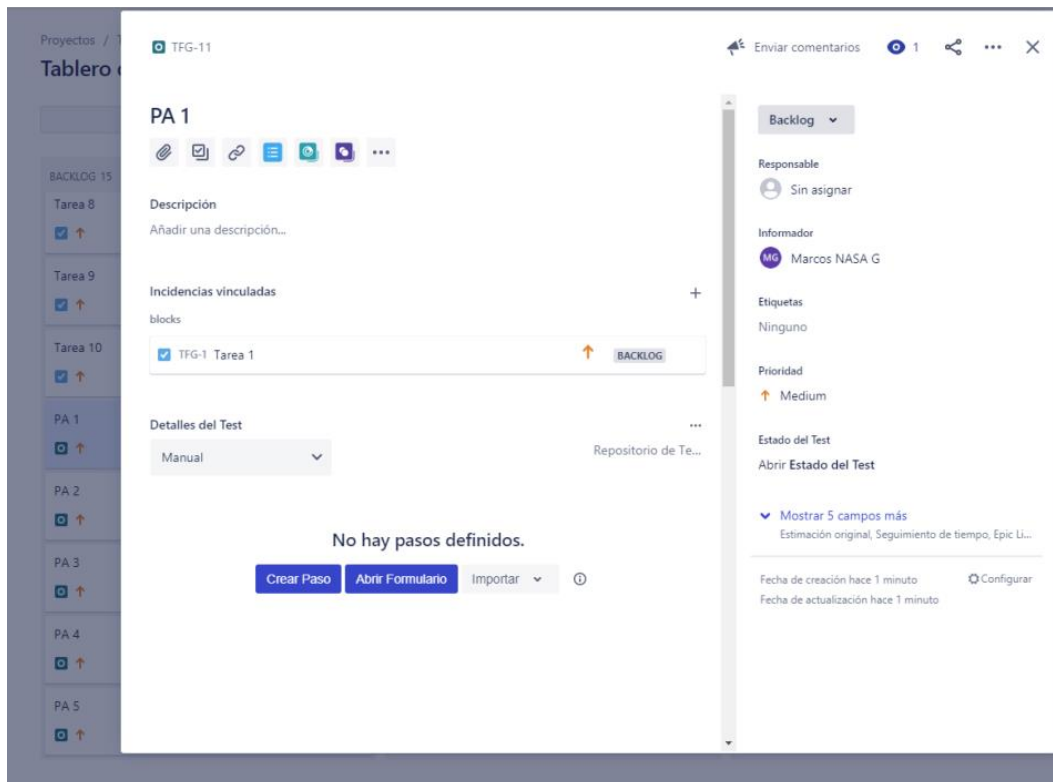
Xray

Perez-Serrano (2020) describe Xray como un plugin desarrollado para Jira el cual ofrece una lista de nuevas incidencias y un tablero de pruebas, donde se agrupan las incidencias de tipo pruebas.

Dentro de las incidencias que se pueden crear se encuentran las pruebas, las precondiciones, los sets de pruebas, el plan de pruebas y la ejecución de pruebas. Mostrando a continuación dos de las incidencias más utilizadas en las figuras 5 y 6 que corresponden a la prueba y ejecución de la prueba respectivamente.

Figura 5

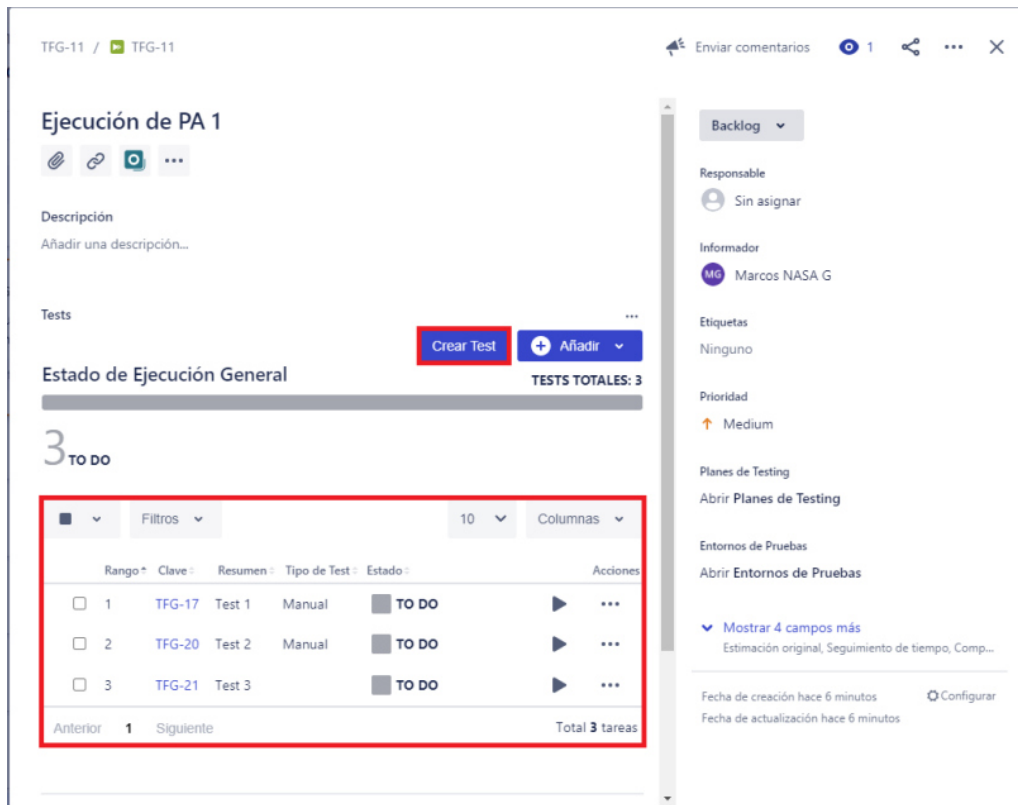
Incidencia de tipo prueba



Nota Tomado de Gestión de Pruebas de Aceptación integrada en una metodología y herramienta para la gestión ágil de proyectos (p. 21) por Perez-Serrano M., 2020

Figura 6

Incidencia de tipo ejecución de prueba



Nota Tomado de Gestión de Pruebas de Aceptación integrada en una metodología y herramienta para la gestión ágil de proyectos (p. 22) por Perez-Serrano M., 2020

Patrón de diseño Screenplay

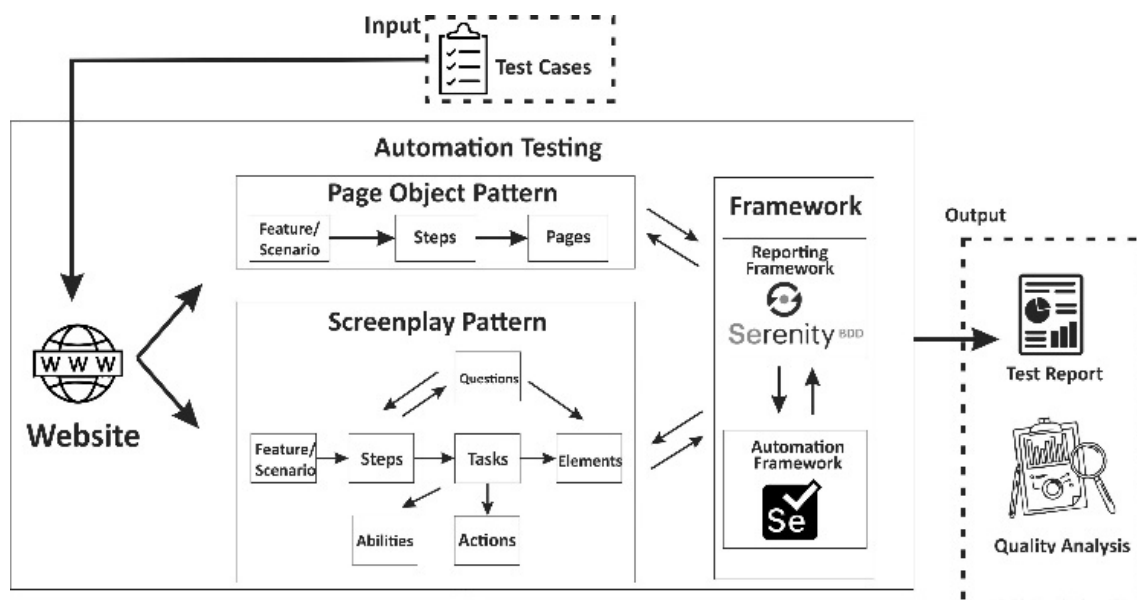
Parte fundamental de la automatización de pruebas es el código, y al referirse al código este siempre tiene que ser construido usando buenos principios los cuales se relacionan a los patrones de diseño.

Dos patrones son los que resaltan en la automatización de pruebas, Page Object Model (POM) y Screenplay, el cual toma como base el patrón POM y le añade otras buenas prácticas utilizadas en el desarrollo de software como los principios SOLID.

Mientras que POM se centra en las páginas, Screenplay separa las acciones de los elementos de las páginas para ser reutilizados, tomando como centro las tareas del usuario, como se ve en la figura 7.

Figura 7

Ejemplo de diseño de sistema de automatización



Nota Tomado de A Comparative Analysis of Quality Page Object and Screenplay Design Pattern on Web-based Automation Testing (p. 2) de Yuniasri D., Badriyah T. y Sa'adah U., 2020

Yuniasri, Badriyah y Sa'adah (2020) concluye que en términos de funcionalidad y efectividad el patrón Page Object tiene una ligera ventaja mientras que el patrón Screenplay es más reutilizable, flexible, entendible y extensible.

Debido a que la utilización será en pruebas de regresión que tienen como característica principal la reutilización, y la automatización debe ser entendida por los analistas de calidad que interactuarán con el proyecto, se optó por Screenplay.

Jenkins

Chavez (2021) define Jenkins como un servidor de automatización de código abierto, utilizado para automatizar cualquier clase de tareas, las cuales se relacionan con el desarrollo, pruebas y entrega o despliegue del software. Pudiendo ser ejecutado en

cualquier ordenador que tenga un entorno de ejecución Java o instalado en paquetes nativos del sistema o Docker.

Entre los ítems que puede haber en Jenkins se encuentran las tareas automatizadas y las tuberías, eligiendo una sobre otra dependiendo de la complejidad de las instrucciones a ejecutar. Además de contar con varios plugins, los cuales ayudan a elaborar procesos complejos como programación de ejecuciones por horarios, ejecuciones parametrizadas, conexión con repositorios de código, envío de correos, generación de backups, entre otros.

Tubería

La tubería o pipeline es una estructura que consiste en una serie de pasos para automatizar procesos, utilizada mayormente en los procesos de pruebas, integración, entrega o despliegue continuos.

Una tubería en Jenkins es un conjunto de plugins que soporta la implementación e integración de tuberías de entrega continua en Jenkins. Estas tuberías de entrega continua automatizan el proceso de obtener el software de un repositorio de versiones y cada cambio en el software pasa por un proceso complejo hasta su entrega. (Jenkins, s.f.)

Se usó las tuberías en este proyecto para la obtención del código desde el repositorio, la creación de datos de entrada y la ejecución de la prueba automatizada.

Bitbucket

Alamo (2022) describe Bitbucket como una herramienta para administrar repositorios basados en Git como sistema de control de versiones. Centraliza el código en un almacenamiento y permite el trabajo colaborativo ofreciendo también la integración con otras herramientas que apoyen en el desarrollo de código de calidad.

3.2.5 Implementación de las áreas, procesos, sistemas y buenas prácticas

En esta sección se detallará el desarrollo en cada fase del proyecto como se describen en el marco de trabajo Scrum. El autor del trabajo fue parte de un equipo cros funcional el cual brindó apoyo al equipo de garantías de la división Riesgo de Crédito. El equipo fue compuesto por el dueño del producto, la maestra Scrum, el líder técnico, dos desarrolladores, una analista de calidad y el automatizador de pruebas.

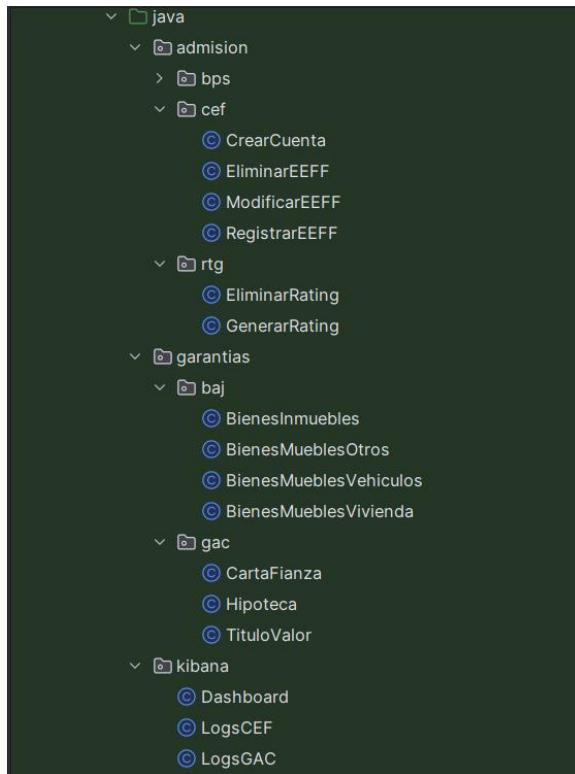
Las fases de Scrum fueron:

3.2.5.1 Inicio

Junto al dueño del producto y el líder analista de calidad se define el equipo que empezará con la automatización de sus pruebas. La decisión se basa según las automatizaciones existentes en IntelliJ mostradas en la figura 8 y automatizaciones existentes en Eclipse, además de la importancia del sistema para la entidad financiera, escogiéndose finalmente al equipo de Garantías.

Figura 8

Proyecto de automatizaciones de la división

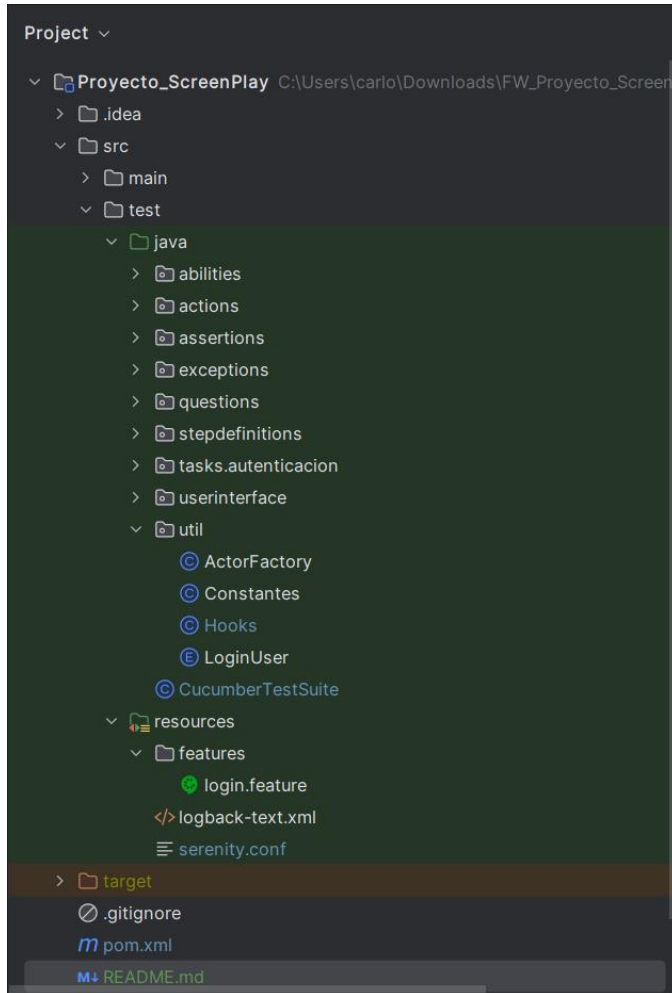


Nota La figura muestra las automatizaciones existentes de la división. Tomada del proyecto automation_junit de la entidad financiera, 2023.

En esta fase también se comparte la estructura base de la automatización como se muestra en la figura 9, la cual fue ideada por un consultor externo para proyectos que usen Serenity BDD y patrón de diseño Screenplay.

Figura 9

Estructura del proyecto de automatización



Nota La figura muestra la estructura de un proyecto de automatización. Tomada del proyecto FW_Proyecto_ScreenPlay de la entidad financiera, 2023.

En el proyecto se encuentra el archivo README.md el cual indica los prerequisites, herramientas utilizadas, descripción de las carpetas y archivos, comando de ejecución y ubicación del reporte.

Tabla 7

Prerequisites de la automatización

Prerequisite	Procedimiento
Instalar Java Development Kit	Descargar e instalar el JDK versión 19 de la página de Oracle.

	Añadir la variable de entorno JAVA_HOME indicando la ubicación del JDK instalado Incluir dentro de la variable de entorno Path la ubicación de la carpeta bin dentro de la ubicación del JDK
Instalar Apache Maven	Descargar el archivo comprimido de la página de Apache Maven y descomprimirlo en cualquier ubicación Añadir la variable de entorno MAVEN_HOME indicando la ubicación del descomprimido Incluir dentro de la variable de entorno Path la ubicación de la carpeta bin dentro de la ubicación del descomprimido
Instalar Git	Descargar el instalador de la página de Git y seguir los pasos
Instalar IntelliJ IDEA Community Edition	Descargar el instalador del IDE de la página de JetBrains e instalar
<i>Nota</i> Tecnologías necesarias para iniciar la automatización. Tomada del archivo README.md del proyecto FW_Proyecto_ScreenPlay de la entidad financiera, 2023.	

Tabla 8

Herramientas utilizadas en la automatización

Herramienta	Descripción
Java	Lenguaje de programación
Maven	Gestor de dependencias
JUnit	Herramienta para pruebas unitarias
Serenity BDD	Herramienta para pruebas basadas en BDD, cuenta con reportería
Cucumber	Redacción de escenarios

Serenity WebDriver	Marco de trabajo base para automatización web
Hamcrest	Librería para controlar aserciones
Logback	Librería para controlar mensajes por consola

Nota Herramientas empleadas en la automatización. Tomada del archivo README.md del proyecto FW_Proyecto_ScreenPlay de la entidad financiera, 2023.

Tabla 9

Descripción de carpetas y archivos del proyecto de automatización

Carpeta o archivo	Descripción
src/test/java/actions	Carpeta que contiene las interacciones que puede realizar un usuario sobre la web
src/test/java/tasks	Carpeta que contiene las distintas tareas que realiza un usuario
src/test/java/questions	Carpeta que contiene las preguntas que realiza el usuario para conocer el estado del elemento web
src/test/java/abilities	Carpeta que contiene las habilidades que puede tener un usuario
src/test/java/stepdefinitions	Carpeta que contiene los pasos de los casos de pruebas
src/test/java/userinterface	Carpeta que contiene los elementos web
src/test/java/assertions	Carpeta que contiene los métodos para realizar validaciones
src/test/java/exceptions	Carpeta que contiene los métodos para controlar las excepciones
src/test/java/util	Carpeta que contiene archivos generales de apoyo como constantes, instanciación de drivers, inicialización del proyecto
src/test/resoruces/features	Carpeta que contiene los archivos tipo feature donde se redactan los casos de pruebas en Gherkin

logback-test.cml	Archivo que contiene la configuración del manejo de mensajes por consola
serenity.conf	Archivo que contiene la configuración del proyecto Serenity

Nota Descripción de la estructura de carpetas. Tomada del archivo README.md del proyecto FW_Proyecto_ScreenPlay de la entidad financiera, 2023.

El comando básico de ejecución por consola es *mvn clean verify* y el reporte generado de la ejecución de la automatización se encuentra en la carpeta target/site/serenity, visualizándose al abrir el archivo index.html.

3.2.5.2 Planificación

Con el equipo de Garantías e intervención del líder analista de calidad, se evaluó la importancia de los flujos a automatizar optando por la automatización de los registros de 10 tipos de garantías, las cuáles se muestran en la figura 10.

Figura 10

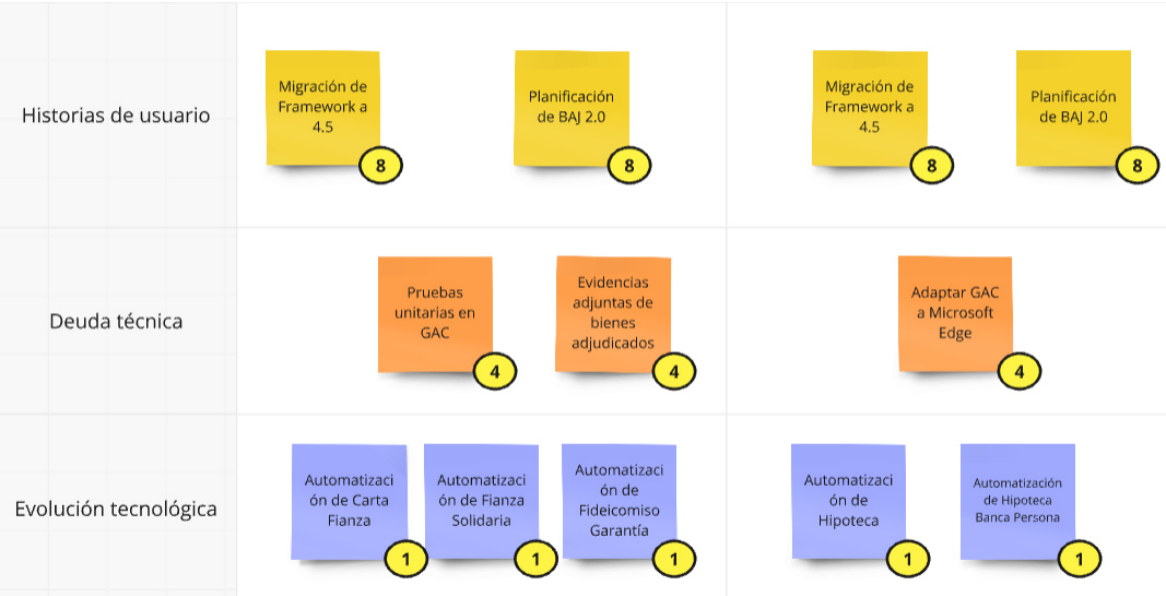
Automatizaciones planificadas

Garantía	Registro	Modificación	Salida	Observaciones	Avance
Carta Fianza	X				Sprint 1
Fianza Solidaria	X				Sprint 1
Fideicomiso Garantía	X				Sprint 1
Hipoteca	X				Sprint 2
Hipoteca Banca Persona	X				Sprint 2
Mobiliaria Sobre Bienes	X				Sprint 3
Mobiliaria Sobre Bienes Banca Persona	X				Sprint 3
Mobiliaria Sobre Dinero	X			CU por CICS 103, 650, 250	Sprint 4
Stand By	X				Sprint 4
Título Valor	X			CU por CICS 830	Sprint 4
Bienes adjudicados					
Bienes inmuebles	X				
Bienes muebles - Vehiculos	X				
Bienes muebles - Vivienda	X				
bienes muebles - Otros	X				
Admisión					
EEFF	X	X	X		
Usuario CEF	X				
Rating	X		X		

Nota La figura muestra las automatizaciones planificadas por el líder analista de calidad. Tomada del archivo Lista de automatizaciones del líder analista de calidad de la entidad financiera, 2023.

Las tareas correspondientes al equipo de Garantía se agregaron al tablero de planificación trimestral del equipo en la sección de evolución tecnológica como se muestra en una adaptación del tablero en la figura 11.

Figura 11
Tablero de planificación trimestral



Nota Adaptación del tablero de planificación trimestral del equipo de Garantías correspondiente al Q2 del año 2023. Elaboración propia.

Por un consenso entre el equipo de desarrollo y de calidad se definió que los pesos de las tareas de evolución tecnológica serían de 1 para no exceder el puntaje máximo de un sprint.

Adicionalmente a las reuniones diarias, se agenda una reunión reiterativa los días martes para la demostración de avances de automatización a todos los analistas de calidad de la división Riesgo de Crédito.

3.2.5.3 Implementación

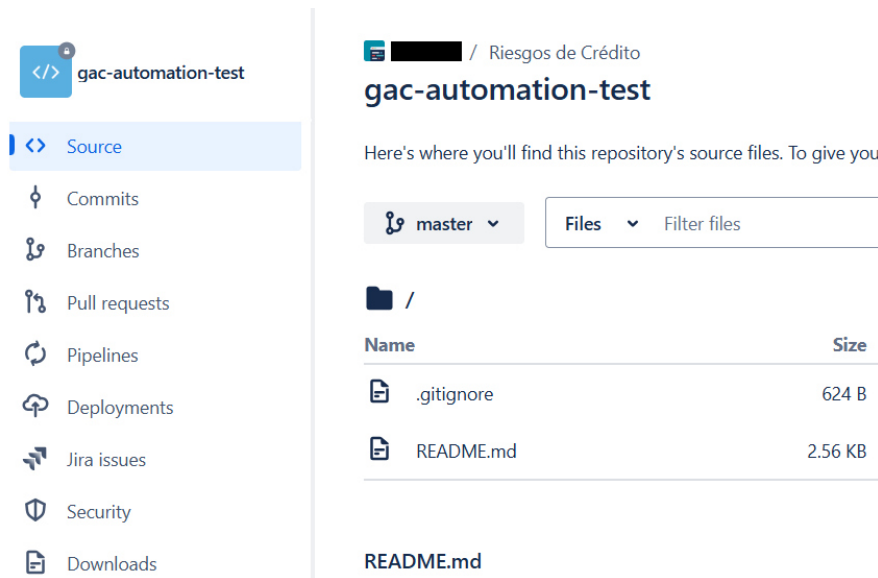
La implementación se dividió en 6 etapas, las cuáles tomaron un sprint o más dependiendo de las tareas a realizar.

Etapa 1

Esta etapa duró 1 sprint, se creó el proyecto correspondiente al sistema GAC como se muestra en la figura 12 y se migró las automatizaciones de carta fianza, hipoteca y titulo valor.

Figura 12

Repositorio de la automatización del sistema



Nota La figura es tomada del espacio de trabajo de Bitbucket de la entidad financiera, 2023.

La creación del repositorio tomó en consideración la nomenclatura del sistema seguido de automation-test. La estructura base de FW_Proyecto_ScreenPlay fue subido a la rama develop y el desarrollo de la automatización se realizó en ramas de feature correspondientes al tipo de garantía, por ejemplo, feature/registro-carta-fianza. Las ramas solo existían durante el desarrollo de la tarea, eliminándolas después de un Pull Request a la rama develop.

Del proyecto de automatización de la división se pudo extraer localizadores y la lógica implementada. Sin embargo, la automatización en JUnit no se encontraba operativa y algunos de los métodos no estaban optimizados, tomando un alto tiempo de ejecución.

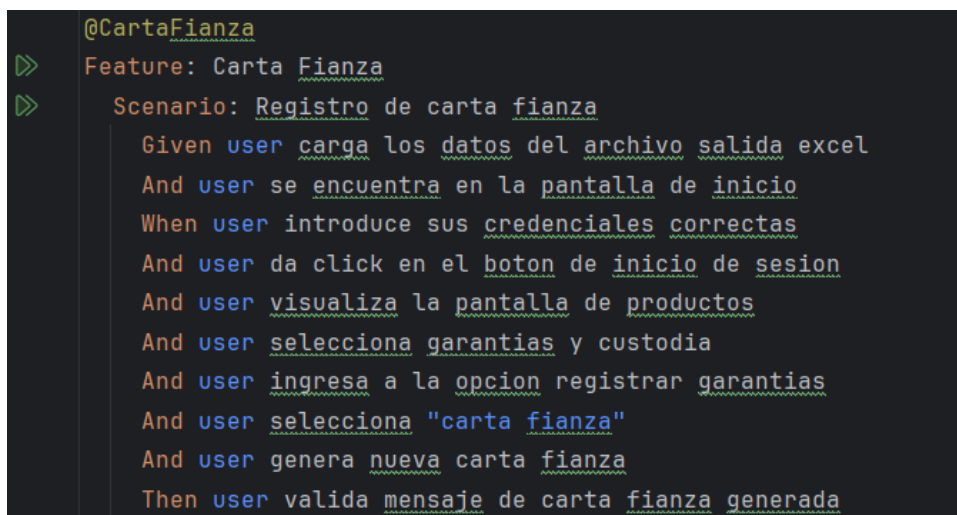
Etapa 2

Esta etapa duró 4 sprints. En esta etapa se realizó las automatizaciones en el orden que se determinó en la planificación. Los pasos que se siguieron para crear la automatización fueron los siguientes.

Paso 1: Crear los archivos feature de Cucumber en la carpeta features, en cada archivo se describe los escenarios con Gherkin como se muestra en la figura 13.

Figura 13

Escenario en Cucumber



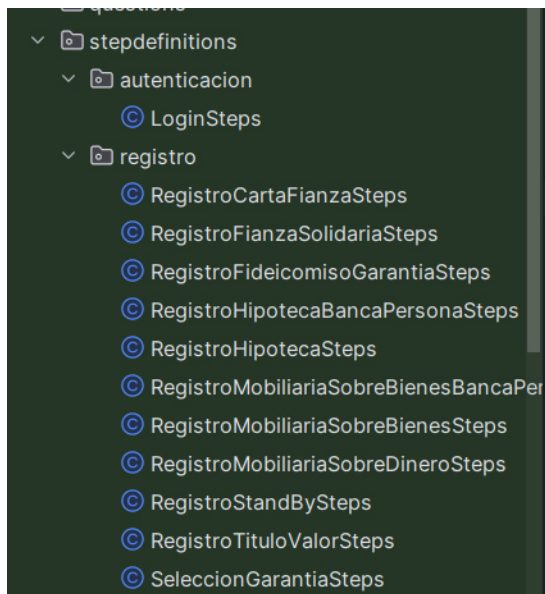
```
@CartaFianza
Feature: Carta Fianza
  Scenario: Registro de carta fianza
    Given user carga los datos del archivo salida excel
    And user se encuentra en la pantalla de inicio
    When user introduce sus credenciales correctas
    And user da click en el boton de inicio de sesion
    And user visualiza la pantalla de productos
    And user selecciona garantias y custodia
    And user ingresa a la opcion registrar garantias
    And user selecciona "carta fianza"
    And user genera nueva carta fianza
    Then user valida mensaje de carta fianza generada
```

Nota La figura muestra el escenario de registro de carta fianza. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Paso 2: Crear las clases Steps en la carpeta stepdefinitions según la característica como se ve en la figura 14, dentro de las clases steps se implementa los métodos por cada paso los cuáles deben contener solo una tarea o acción de ser posible como se ve en la figura 15.

Figura 14

Carpeta stepdefinitions



Nota La figura muestra las clases Steps donde se definió el código de los pasos del escenario. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Figura 15

Clase LoginSteps.java

```
public class LoginSteps {

    @And("{actor} se encuentra en la pantalla de inicio")
    public void se_encuentra_en_la_pagina_de_login(Actor actor) throws Exception {
        try {
            actor.attemptsTo(
                Navegar.aLoginPage()
            );
        } catch (Throwable e) {
            ExceptionHandler.Error(e);
        }
    }

    @When("{actor} introduce sus credenciales correctas")
    public void inicia_sesion_con_credenciales_validas(Actor actor) throws Exception{
        try {
            actor.attemptsTo(
                RealizarLogin.Como(LoginUser.VALID_USER)
            );
        } catch (Throwable e) {
            ExceptionHandler.Error(e);
        }
    }
}
```

Nota La figura muestra los métodos de los pasos de inicio de sesión. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Paso 3: Crear clases Page en la carpeta userinterface donde se encuentran los localizadores según la página o módulo como se ve en la figura 16.

Figura 16

Clase LoginPage.java

```
public class LoginPage extends PageObject {

    1 usage
    private static Target usernameInput = Target.the( targetElementName: "Campo usuario").located(By.id("inpuser"));
    1 usage
    private static Target passwordInput = Target.the( targetElementName: "Campo clave").located(By.id("inp_clave"));
    1 usage
    private static Target dropdownLocator = Target.the( targetElementName: "Selector de ambiente").located(By.name(""));
    1 usage
    private static Target loginButton = Target.the( targetElementName: "Botón Inicio de sesión").located(By.id("Acep
    1 usage
    private static Target moduleLink = Target.the( targetElementName: "Link de garantías y custodia").located(By.lin

    1 usage
    public static Target getUsernameInput() { return usernameInput; }
    1 usage
    public static Target getPasswordInput() { return passwordInput; }
    1 usage
    public static Target getLoginButton() { return loginButton; }
    1 usage
    public static Target getDropdownLocator(){ return dropdownLocator; }
    7 usages
    public static Target getModulo(){ return moduleLink; }
```

Nota La figura muestra las variables que representan a los localizadores de los elementos web. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Paso 4: Crear clases en la carpeta tasks para las tareas, las cuáles se definen como una secuencia de acciones como se ve en la figura 17 y serán empleadas en los pasos de que tengan más complejidad.

Figura 17

Clase RealizarLogin.java

```
1 usage
public class RealizarLogin implements Task {

    @Override
    public <T extends Actor> void performAs(T t) {

    }

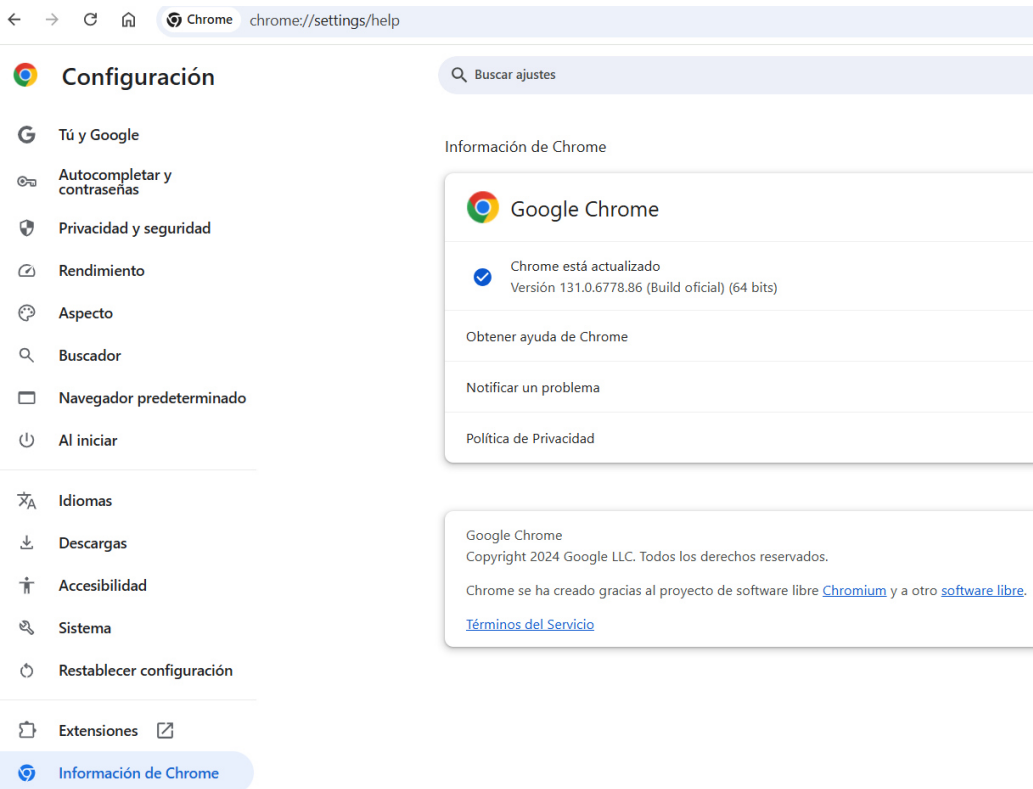
1 usage
    public static Performable Como(LoginUser loginUser){
        return Task.where( title: "{0} inicia sesion como " + loginUser,
            Enter.Text(LoginPage.getUsernameInput(), loginUser.username),
            Enter.Text(LoginPage.getPasswordInput(), loginUser.password),
            SendKeys.of(Keys.ARROW_DOWN).into(LoginPage.getDropdownLocator())
        );
    }
}
```

Nota La figura muestra la Tarea de inicio de sesión. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Paso 5: Validar que el driver del navegador a usar sea compatible con el navegador. Para ello se debe ir la Configuración del navegador y buscar la sección de Información del navegador o donde se visualice la versión del navegador como se ve en la figura 18 y descargar el webdriver oficial correspondiente como se ve en la figura 19 para colocarlo en la carpeta webdriver.windows del proyecto y cuya ruta debe estar definida en las propiedades de serenity como se ve en la figura 20.

Figura 18

Versión del navegador



Nota La figura muestra la información de Chrome el cual se encuentra en la versión 131. Tomada de Google Chrome, s.f.

Figura 19

Webdriver del navegador

Stable		
Version: 131.0.6778.85 (r1368529)		
Binary	Platform	URL
chrome	linux64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/linux64/chrome-linux64.zip
chrome	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/mac-arm64/chrome-mac-arm64.zip
chrome	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/mac-x64/chrome-mac-x64.zip
chrome	win32	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/win32/chrome-win32.zip
chrome	win64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/win64/chrome-win64.zip
chromedriver	linux64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/linux64/chromedriver-linux64.zip
chromedriver	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/mac-x64/chromedriver-mac-x64.zip
chromedriver	win32	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/win32/chromedriver-win32.zip
chromedriver	win64	https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.85/win64/chromedriver-win64.zip

Nota La figura muestra la lista de chromedriver de la versión correspondiente al navegador. Tomada de Google Chrome s.f.

Figura 20

Archivo serenity.conf

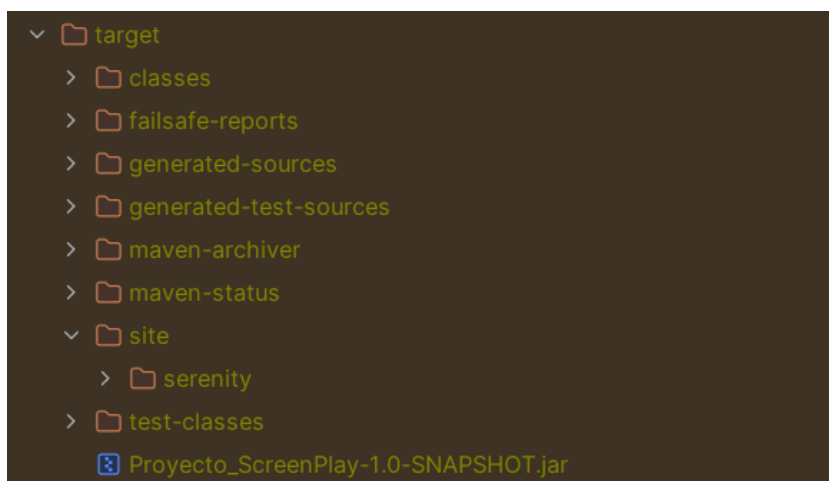
```
drivers {
  windows {
    webdriver.chrome.driver = "src/test/resources/webdriver/windows/chromedriver.exe"
    webdriver.gecko.driver = ""
    webdriver.edge.driver = "src/test/resources/webdriver/windows/msedgedriver.exe"
    webdriver.ie.driver = "src/test/resources/webdriver/windows/IEDriverServer32.exe"
  }
  linux {
    webdriver.chrome.driver = ""
  }
}
```

Nota La figura muestra la ubicación de los webdriver dentro del proyecto de automatización. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Paso 6: Ejecutar la sentencia *mvn clean verify -Dwebdriver.driver=Chrome -Dcucumber.filter.tags="@CartaFianza"* en la terminal y ver la generación de la carpeta target como se ve en la figura 21, ubicándose el reporte en *site/serenity/index.html*. La interpretación de los comandos de la sentencia se ve en la tabla 10.

Figura 21

Carpeta target



Nota La figura muestra la carpeta target, resultante de la ejecución de la automatización donde se guarda lo necesario para visualizar el reporte. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Tabla 10

Comandos de Maven

Comando	Función
clean	Limpia la carpeta target, cada ejecución acumula los archivos que fueron necesarios para los reportes anteriores.
verify	Construye el proyecto y ejecuta las pruebas
-Dwebdriver.driver	Selecciona el webdriver a utilizar para levantar el navegador preferido
-Dcucumber.filter.tags	Filtra las pruebas a ejecutar según el o los tags. Los tags preceden al Feature, Scenario o Example.

Nota Explicación de los comandos de la sentencia de ejecución de la automatización.
Elaboración propia.

Con la finalización de esta etapa se cumple el objetivo de crear proyectos de automatización de las aplicaciones web, elaborando como base un lineamiento interno

Etapa 3

Esta etapa duró dos sprints. El objetivo de la etapa era poder ingresar los datos externamente mediante el uso de una hoja de cálculo en Excel. Los valores deben ser leídos de las columnas correspondientes y al finalizar la ejecución de la automatización se debe guardar el resultado en la última columna como se muestra en la figura 22.

La sintaxis del archivo Excel es Data- seguido del tipo de garantía y su ubicación es en resources como se muestra en la figura 23.

Figura 22

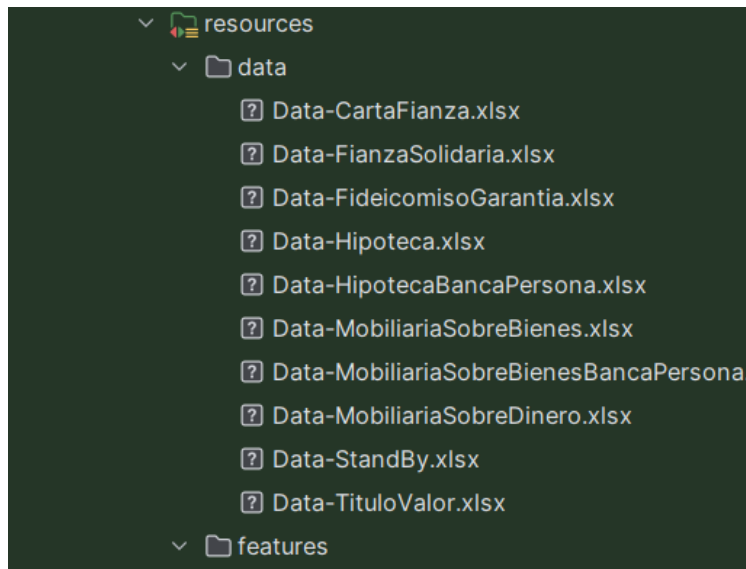
Archivo Data-CartaFianza.xls

A	B	C	D	E	F	G	W	X
CU	Tienda	Tipo Cobertura	Tipo Documento	Rango	Moneda	Estado	Comentario	Resultado
00	100	Específica	REPROGRAMACIONES REACTIVAS	RANGO 1	Soles	VIGENTE	Prueba automatizada	Garantía N° 00
00	100	Específica	REPROGRAMACIONES REACTIVAS	RANGO 1	Soles	VIGENTE	Prueba automatizada	CU no existe o no válido
00	100	Específica	REPROGRAMACIONES REACTIVAS	RANGO 1	Soles	VIGENTE	Prueba automatizada	Fecha de vigencia debe ser posterior a fecha actual
00	100	Específica	REPROGRAMACIONES REACTIVAS	RANGO 1	Soles	VIGENTE	Prueba automatizada	Garantía N° 00

Nota La figura muestra algunos de los valores de entrada usadas en la automatización. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Figura 23

Carpeta data



Nota En la figura se muestra las hojas de cálculo en Excel de los tipos de garantía de la automatización. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Adicionalmente a la creación de las hojas de cálculo, se añadió un paso para la lectura del Excel como se ve en la figura 24, así como los métodos de lectura y escritura como se ven en las figuras 25 y 26.

Figura 24

Escenario en Cucumber con lectura de Excel

```
1 @CartaFianza
2 Feature: Carta Fianza
3 Scenario: Registro de carta fianza
4     Given user carga los datos del archivo excel carta fianza
5     And user carga los datos del archivo salida excel
6     And user se encuentra en la pantalla de inicio
7     When user introduce sus credenciales correctas
8     And user da click en el boton de inicio de sesion
9     And user visualiza la pantalla de productos
10    And user selecciona garantias y custodia
11    And user ingresa a la opcion registrar garantias
12    And user selecciona "carta fianza"
13    And user genera nueva carta fianza
14    #Then user valida mensaje de carta fianza generada
```

Nota En la figura se muestra la lectura del archivo Excel como primer paso. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Figura 25

Archivo LeerCartaFianzaExcel.java

```
public class LeerCartaFianzaExcel implements Task {
    2 usages
    public static LeerCartaFianzaExcel leer() { return instrumented(LeerCartaFianzaExcel.class); }

    public <T extends Actor> void performAs(T actor) {
        actor.attemptsTo(
            LeerExcel.conNombre( nombreArchivo: "Data-CartaFianza")
        );
        Sheet sheet = actor.recall( key: "excel");

        //Lista que guardará los datos del archivo .csv
        List<CartaFianza> cartaFianzaList = new ArrayList<>();

        int numRows = sheet.getPhysicalNumberOfRows(); // Obtiene el número de filas en la hoja de cálculo
        for (int i = 1; i <= sheet.getLastRowNum(); i++) {
            Row row = sheet.getRow(i);
            if(row.getCell(23) == null || row.getCell(23).getCellType() == CellType.BLANK) {...}
            // se hace un mapeo de las columnas con las variables del objeto CartaFianza
            // se crea una CartaFianza y se añade a la lista cartaFianzaList
        }
        actor.remember("cartaFianzaList", cartaFianzaList); // Guarda la lista datos
    }
}
```

Nota La figura muestra la tarea usada para leer el Excel, mapeando las filas no procesadas a una lista de objetos del tipo de garantía y guardando la lista con el actor. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Figura 26

Archivo EscribirCartaFianzaExcel.java

```
import ...

4 usages
public class EscribirCartaFianzaExcel implements Task {

    2 usages
    private final int numeroFila;
    2 usages
    private final String mensaje;

    no usages
    public EscribirCartaFianzaExcel(int numeroFila, String mensaje) {
        this.numeroFila = numeroFila;
        this.mensaje = mensaje;
    }

    public static EscribirCartaFianzaExcel en(int numeroFila, String mensaje) {
        return instrumented(EscribirCartaFianzaExcel.class, numeroFila, mensaje);
    }

    @Override
    public <T extends Actor> void performAs(T t) {
        String nombreArchivo = "src/test/resources/data/Data-CartaFianza.xlsx";
        EscribirExcel.conNombre(nombreArchivo, numeroFila, numeroCelda: 23, mensaje);
    }
}
```

Nota La figura muestra la tarea usada para guardar el mensaje de respuesta en el archivo Excel. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Posteriormente se hace una modificación al paso final, comentándolo y modificando la lógica del paso de generar la garantía como se muestra en la figura 27, haciendo uso de todas las filas que se deben procesar y guardando el resultado.

Figura 27

Nuevo método de generación de carta fianza

```
@And("{actor} genera nueva carta fianza")
public void genera_nuevo_titulo_valor(Actor actor) throws Exception {
    try {
        List<CartaFianza> cartaFianzaList = actor.recall(key: "cartaFianzaList");
        int cantidadFilas = cartaFianzaList.size();
        int iterador = 1;
        for (CartaFianza cartaFianza : cartaFianzaList) {
            actor.attemptsTo(
                GenerarCartaFianza.Con(cartaFianza),
                Check.whether(Visibility.of(SecurePage.getGuaranteeMessage()))
                    .andIfSo(
                        EscribirCartaFianzaExcel.en(cartaFianza.getNumeroFila(),
                            Text.of(SecurePage.getGuaranteeMessage()).answeredBy(actor))
                    )
            );
            if(iterador < cantidadFilas) {
                new RegistroCartaFianzaSteps().ingresa_a_la_opcion_registrar_garantias(actor);
                new SeleccionGarantiaSteps().selecciona_garantia(actor, garantia: "carta fianza");
            }
            iterador ++;
        }
    } catch (Throwable e) {
        ExceptionHandler.Error(e);
    }
}
```

Nota En la figura se muestra como el paso de generar nueva carta fianza procesa una lista de cartas fianza. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Etapas 4

Esta etapa duró un sprint. Para el desarrollo de esta etapa se requirió previamente las credenciales necesarias para conectar a Jira XRay las cuáles se agregaron en los métodos de conexión y envío de reporte de la automatización como se ve en la figura 28. Adicionalmente se agrega dos tags al archivo feature, un tag antes de Feature indicando el código de incidencia de ejecución de pruebas y un segundo tag antes del escenario indicando el código de incidencia de la prueba.

Figura 28

Clase CucumberTestSuite con integración de Jira XRay

```
@RunWith(CucumberWithSerenity.class)
@cucumberOptions(
    plugin = {"json:target/build/cucumber.json"},
    stepNotifications = true,
    features = "src/test/resources/features",
    tags = "@CG-██████"
)

public class CucumberTestSuite {

    @BeforeClass
    public static void beforeAll() {
        setXrayEnvironmentVariables();
    }

    @AfterClass
    public static void afterAll() {
        JiraXrayOperation.importCucumberResult( cucumberJsonFile: "/target/build/cucumber.json");
    }

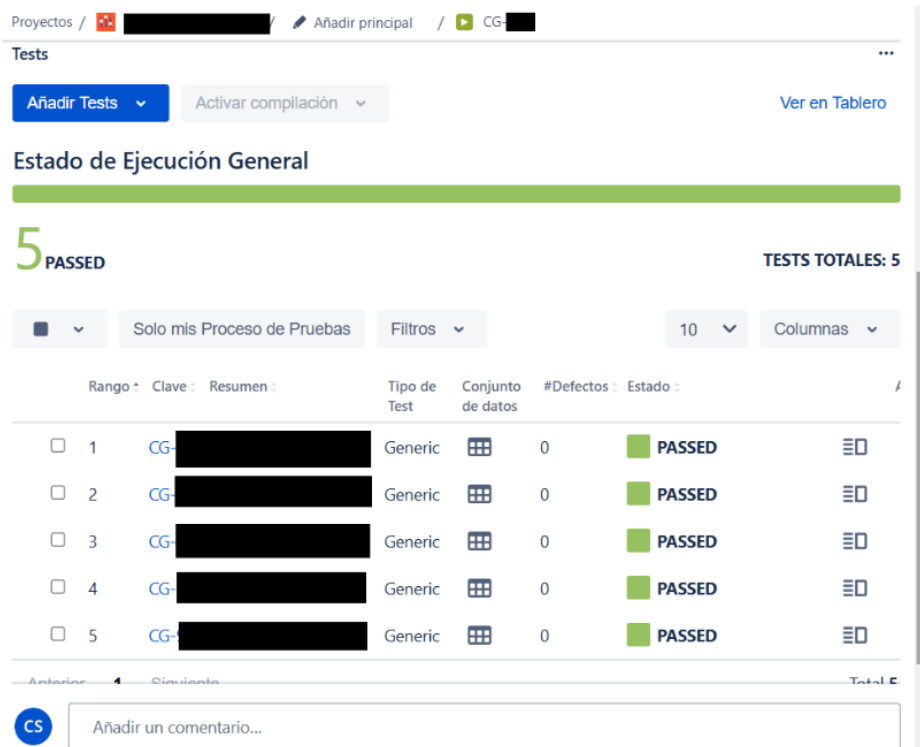
}
```

Nota En la figura se muestra el método beforeAll donde se fija los valores de las credenciales antes de la ejecución de la automatización y el método afterAll que importará el resultado del reporte hacia xray al finalizar la ejecución de la automatización. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

El reporte será enviado a la incidencia de pruebas dentro de la incidencia de ejecución de pruebas correspondiente como se ve en la figura 29. Dependiendo del resultado tendrá el estado PASSED si fue exitoso y FAILED si hubo errores, además de mostrar el reporte paso a paso con las capturas de pantallas dentro de los detalles de la prueba.

Figura 29

Incidencia de ejecución de pruebas en Jira



Nota La figura muestra la incidencia de ejecución de las pruebas que contiene la prueba automatizada y pruebas manuales en estado PASSED. Tomada del proyecto en Jira de la entidad financiera, 2023.

Con esta etapa se consiguió el objetivo de integrar la automatización con Jira XRay para enviar los reportes de resultados de pruebas automatizadas.

Etapa 5

Esta etapa duró 2 sprints. Para la implementación de tuberías se brindó el acceso al servidor de pruebas donde se desplegó Jenkins usando el archivo `jenkins.war` y la línea de comando `java -jar jenkins.war`.

Se instaló y configuró los plugins mostrados en la tabla 11 los cuales eran útiles para la automatización, como el envío de correos y la conexión al repositorio con git.

Posteriormente se limitó la cantidad de construcciones guardadas en el log por tubería a 20 construcciones por temas de espacio de servidor.

Tabla 11

Plugins de Jenkins

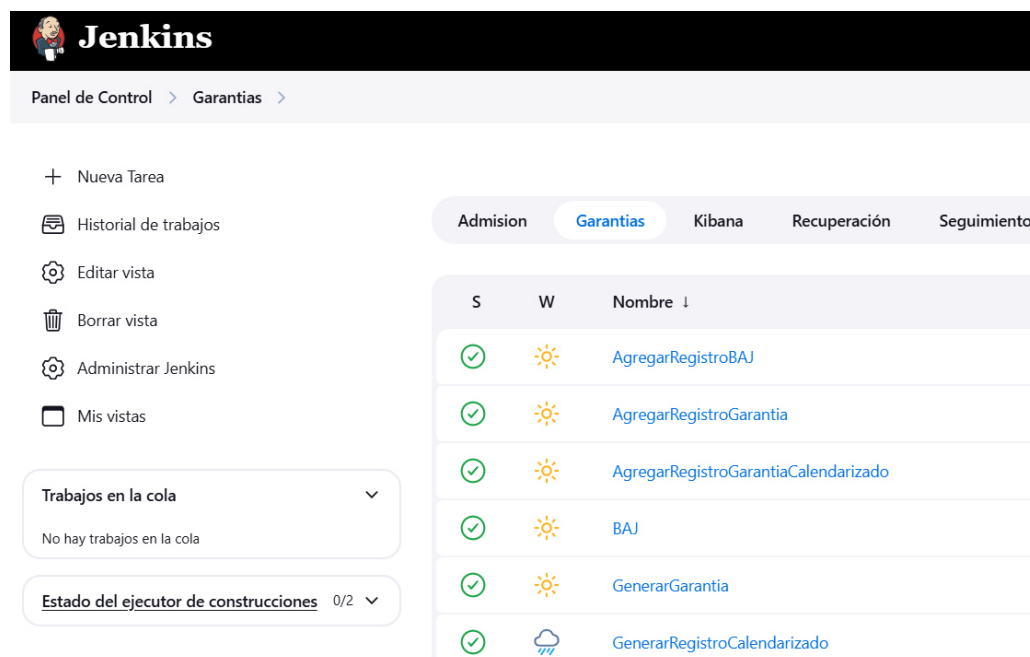
Plugin	Función
Git Plugin	Permite la clonación del proyecto desde el repositorio
Email Extension Plugin	Envío de correos desde el correo configurado y servidor smtp definido.
Parameterized Trigger Plugin	Permite configurar la llamada a otra construcción con parámetros especificados al finalizar una construcción
ThinBackup	Configura respaldos de Jenkins

Nota Obtenida de Plugins instalados en Jenkins del servidor de pruebas de la entidad financiera, 2023.

Después se estructuró las vistas de proyecto según el equipo como se ve en la figura 30, colocando las automatizaciones disponibles.

Figura 30

Dashboard de Jenkins

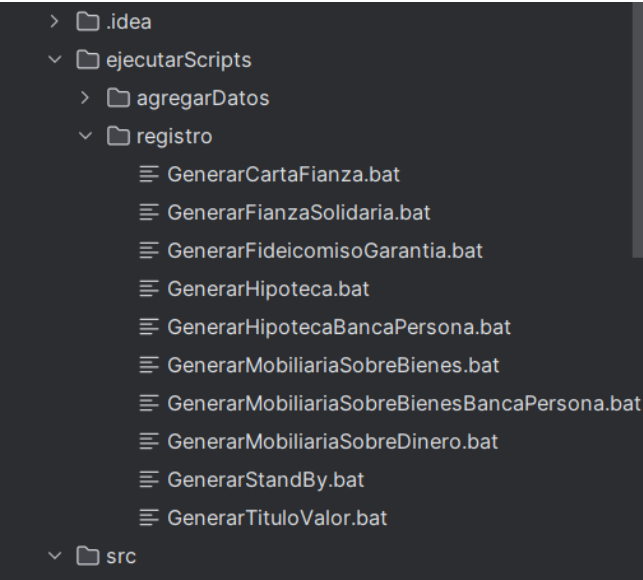


Nota En la figura se muestra la vista del equipo de garantías con las tuberías disponibles. Tomada de Jenkins del servidor de pruebas de la entidad financiera, 2023.

Junto a la construcción de tuberías se creo archivos batch situados en la carpeta ejecutarScripts dentro del proyecto como se muestra en la figura 31, para ser usados en las tuberías y en ejecuciones locales.

Figura 31

Carpeta ejecutarScripts



Nota En la figura se muestra los archivos batch los cuales contienen la cadena de ejecución de la automatización. Tomada del proyecto gac-automation-test de la entidad financiera, 2023.

Posteriormente se creó clases para generar data de entrada en los archivos Excel de la garantía las cuales se ubicaron en src/main/java. La generación de data se basó en replicar la última o últimas filas y modificando los datos que requerían una variación. Para ello se modificó el color de fondo de las cabeceras dentro del archivo Excel, según la regla de colores indicada en la tabla 12

Tabla 12

Regla de colores de cabeceras en Excel de garantía

Color	Descripción
Rojo	Dato necesario y variable. El registro de garantía requiere de este dato y el valor tiene reglas que de no cumplirse fallaría la

	automatización. Ejemplos: fecha de vigencia mayor a fecha actual, código único de deudor de un solo uso.
Amarillo	Dato necesario. El registro de garantía requiere que este dato no este vacío. Ejemplos: tienda, monto
Blanco	Dato no relevante. El registro de garantía admite valores vacíos para dichos campos. Ejemplos: descripción, comentario

Nota Extraído de un archivo Excel del proyecto gac-automation-test de la entidad financiera, 2023.

Finalmente se configura las tuberías de AgregarRegistroGarantia y GenerarGarantia para ejecuciones a demanda y AgregarRegistroGarantiaCalendarizado y GenerarGarantiaCalendarizado para ejecuciones programadas las cuales llaman a las dos tuberías mencionadas antes respectivamente.

Figura 32

Script de tubería AgregarDatosGarantia

Pipeline

Definition

Pipeline script

```

1 pipeline {
2   agent any
3   stages {
4     stage('Extraer cambios') {
5       steps {
6         git credentialsId: [REDACTED]
7         url: [REDACTED]
8       }
9     }
10    stage('Generar garantia') {
11      steps {
12        echo 'Se genera la garantia ' + params.tipoGarantia
13        call "C:\users\[REDACTED]\workspace\gac-automation-test\ejecutarScripts\AgregarDatos"+params.tipoGarantia+".bat "
14      }
15    }
16    stage('Subir cambios') {
17      steps {
18        call "C:\users\[REDACTED]\workspace\gac-automation-test\ejecutarScripts\GitPush.bat"
19      }
20    }
21  }

```

Nota En la figura se muestra el script para la generación de data de entrada. Tomada de Jenkins del servidor de pruebas de la entidad financiera, 2023.

Figura 33

Script de tubería GenerarGarantia

```
Script ?
1 def emails = [REDACTED]
2 pipeline {
3   agent any
4   stages {
5     stage('Extraer cambios') {
6       steps {
7         git credentialsId: [REDACTED],
8           url: [REDACTED]
9       }
10    }
11    stage('Generar garantia') {
12      steps {
13        echo 'Se genera la garantia ' + params.tipoGarantia
14        call "C:\users\[REDACTED]\workspace\gac-automation-test\ejecutarScripts\Generar"+params.tipoGarantia+".bat "
15      }
16    }
17    stage('Subir cambios') {
18      steps {
19        call "C:\users\[REDACTED]\workspace\gac-automation-test\ejecutarScripts\GitPush.bat"
20      }
21    }
22  }
23  post {
24    success {
25      sendSuccessfulEmail()
26    }
27    failure {
28      sendFailureEmail()
29    }
30  }
31 }
32
33 def sendSuccessfulEmail() {
```

Nota En la figura se muestra el script para ejecutar la automatización además del envío de correos posterior a la ejecución. Tomada de Jenkins del servidor de pruebas de la entidad financiera, 2023.

Figura 34

Configuración de tubería AgregarRegistroGarantiaCalendarizado

AgregarRegistroGarantiaCalendarizado > Configuration

- ☐ build when a change is pushed to bitbucket
- ☐ Construir tras otros proyectos ?
- ☐ Consultar repositorio (SCM) ?
- ☒ Ejecutar periódicamente ?

Programador ?

0 16 * * 1-5

⚠ Spread load evenly by usingq 'H 16 * * 1-5' rather than '0 16 * * 1-5'

Nota La figura muestra la configuración del tiempo en que se ejecutará la generación de data para la automatización. Tomada de Jenkins del servidor de pruebas de la entidad financiera, 2023.

Figura 35

Configuración de tubería GenerarGarantiaCalendarizado

GenerarRegistroCalendarizado > Configuration

☐ Throttle builds ?

Build Triggers

☐ Build when a change is pushed to BitBucket

☒ Construir tras otros proyectos ?

Projects to watch

AgregarRegistroGarantiaCalendarizado,

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☐ Always trigger, even if the build is aborted

☐ Consultar repositorio (SCM) ?

☐ Ejecutar periódicamente ?

Nota La figura muestra la configuración que gatilla la ejecución de la automatización al terminar la ejecución programada de manera exitosa de la tubería AgregarRegistroGarantiaCalendarizado. Tomada de Jenkins del servidor de pruebas de la entidad financiera, 2023.

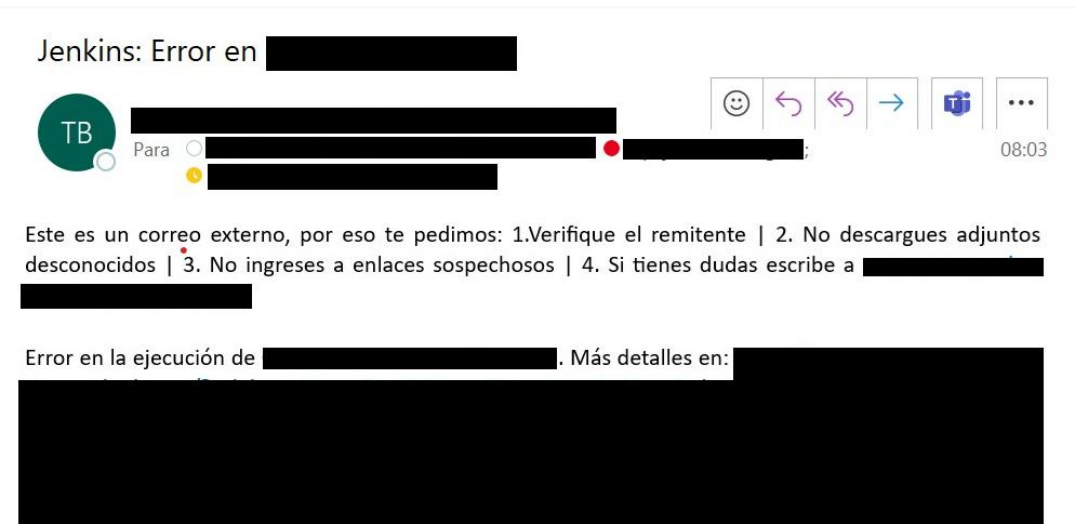
En esta etapa se ha podido cumplir con el objetivo de configurar Jenkins en un servidor de pruebas, creando tuberías para las ejecuciones de scripts automatizados

Etapa 6

La última etapa de la fase de implementación duró un sprint. En esta etapa se monitoreo el desempeño de las automatizaciones, revisando los correos recibidos de las ejecuciones como se ve en la figura 36 y corrigiendo errores que se presenten propios de la automatización.

Figura 36

Email con resultado fallido de la ejecución de la automatización



Nota En la figura se muestra el correo recibido de una ejecución fallida de la automatización, indicando la ejecución fallida y el enlace a los detalles de la falla. Tomado desde el Outlook del correo corporativo, 2023.

3.2.5.4 Lanzamiento

Finalmente, en esta fase se hace una comparativa con un caso real de un pedido externo de pruebas de la aplicación el cual requirió 90 registros. La comparación de tiempos se muestra en la figura 37.

Figura 37

Resultados de reducción de tiempos por automatización

Accion	Garantia	1 REGISTRO				90 REGISTROS			
		Tipo	Tiempo	Tipo	Tiempo	Tipo	Tiempo	Tipo	Tiempo
Registro	Carta Fianza	Automatizado	45s	Manual	3m	Automatizado	45m25s	Manual	5h
	Hipoteca	Automatizado	1m39s	Manual	4m	Automatizado	2h6m15s	Manual	7h

Nota En la figura se muestra la tabla comparativa de los tiempos en horas, minutos, segundos entre las pruebas automatizadas y manuales. Tomada del reporte de Q3 de la entidad financiera, 2023.

También se hace la transferencia del proyecto al analista de calidad del equipo, apoyando con la instalación de las tecnologías necesarias. Se brindó una lista de comandos básicos de git mostrados en la tabla 13 y la guía desde la clonación hasta la

creación de un Pull Request. Se explicó los escenarios existentes, los pasos y la lógica detrás de ellos, así como las tareas y acciones utilizadas. Se enseñó la estrategia de búsqueda de localizadores de elementos web y otras dudas presentadas. La asistencia al analista de calidad se extendió hasta que ella misma implementara su propio flujo de automatización, el cual fue Salida de garantía.

Tabla 13

Comandos básicos de Git

Comando	Función
Git clone	Clonar el proyecto al ordenador
Git checkout feature/funcionalidad	Posicionarse en la rama correspondiente a la funcionalidad, la rama fue creada en Bitbucket
Git checkout -b feature/funcionalidad	Crear y posicionarse en la rama correspondiente a la funcionalidad de no haber creado la rama en Bitbucket
Git add .	Añade todos los cambios preparándolos para el commit
Git commit -m “mensaje”	Realiza un commit donde el mensaje describe a grandes rasgos el desarrollo realizado
Git push	Subir los cambios al repositorio si la rama ya existe en Bitbucket
Git push -u origin feature/funcionalidad	Subir la nueva rama con los cambios si esta no existía en Bitbucket

Nota En la tabla se muestra los comandos básicos que usarán los analistas de calidad para trabajar en las funcionalidades nuevas. Elaboración propia

3.3 Evaluación

3.3.1 Evaluación económica / Evaluación costo-beneficio

Las tecnologías usadas son de código abierto, y en caso de Bitbucket, la licencia ya se encontraba vigente antes del inicio del proyecto por lo que no se añadió ningún costo en la implementación de la automatización.

Se toma como referencia el caso de 90 registros presentado al final de la implementación en la tabla 14 para ver el ahorro de tiempo.

Tabla 14

Comparación de tiempo de prueba manual y automatizado para 90 registros

	Prueba manual	Prueba automatizada
Registro carta fianza	5 horas	45 minutos 25 segundos
Registro hipoteca	7 horas	2 horas 6 minutos 15 segundos
Registro carta fianza + 12 horas Registro hipoteca		2 horas 6 minutos 15 segundos

Nota Se muestra el tiempo tomado por tipo de garantía y ambas juntas, se considera que la prueba automatizada de diferentes garantías se puede ejecutar en paralelo. Elaboración propia.

El tiempo de pruebas fue reducido en 82.5% ahorrando más de un día de trabajo del desarrollo. El ahorro de costos se infiere al comparar el gasto estimado del desarrollo cuando se realizó pruebas manuales como se ve en la tabla 15, y el mismo caso con pruebas automatizadas como se ve en la tabla 16.

Tabla 15

Presupuesto estimado del desarrollo del caso con pruebas manuales

Personal	Días trabajados	Asignación	Sueldo mensual	Costo en el desarrollo
Líder técnico	15	25%	S/ 8,000.00	S/ 1,000.00
Desarrollador	15	100%	S/ 5,000.00	S/ 2,500.00

Analista	de 15	100%	S/ 3,500.00	S/ 1,750.00
calidad				
Total				S/ 5250.00

Nota Elaboración propia.

Tabla 16

Presupuesto estimado del desarrollo del caso con pruebas automatizadas

Personal	Días trabajados	Asignación	Sueldo mensual	Costo en el desarrollo
Líder técnico	13	25%	S/ 8,000.00	S/ 866.67
Desarrollador	13	100%	S/ 5,000.00	S/ 2166.67
Analista	de 13	100%	S/ 3,500.00	S/ 1516.67
calidad				
Total				S/ 4550.00

Nota Elaboración propia.

Por lo que finalmente se infiere que el desarrollo de software tuvo una reducción del costo de 13.33%, porcentaje que puede aumentar para desarrollos de mayor duración.

También se redujo el costo mensual de S/ 4500 que representa al automatizador de pruebas de manera permanente. Al final del presente trabajo, cada analista de calidad de la división adquirió la capacidad de automatizar pruebas por lo que un rol especializado únicamente en automatizar de pruebas ya no era necesario.

Otros beneficios para el equipo fueron:

- Tener una herramienta de monitoreo de la estabilidad del sistema ya que, con la ejecución diaria de la automatización, en caso se presente un error, una notificación sería enviada.
- Facilitar el apoyo externo. Al tener definida la data de entrada en el formato de Excel, los interesados solo requieren enviar la data como se solicita para ejecutar la automatización.
- Eliminación del error humano al generar data de entrada de manera automatizada, e instrucciones claras para el ingreso manual de data.

- Aumento en la calidad del producto. El analista de calidad pudo centrarse en crear más escenarios de pruebas mientras las pruebas automatizadas se ejecutaban, aumentando la cobertura de las pruebas.

3.3.2 Interpretación del VAN y del TIR

Para la inversión inicial se ha considerado el tiempo invertido por los roles dentro de la división, quienes asistieron a las reuniones semanales de 2 horas durante la etapa de la implementación la cual tomó 11 sprints donde cada sprint duraba 2 semanas.

Tabla 17

Inversión inicial estimada

Rol	Cantidad	Sueldo mensual estimado	Horas por día	Duración	Costo estimado
Analista de calidad	3	S/ 3,500.00	2 horas	22 días	S/ 1,925.00
Líder analista de calidad	1	S/ 8,000.00	2 horas	22 días	S/ 1,466.67
Dueño del producto	1	S/ 10,000.00	2 horas	22 días	S/ 1,833.33
Maestra Scrum	1	S/ 8,000.00	2 horas	22 días	S/ 1,466.67
Automatizador de pruebas	1	S/ 4,500.00	2 horas	22 días	S/ 825.00
Total					S/ 7,516.67

Nota Elaboración propia.

En la tabla 18 se muestra el flujo de caja del primer año donde ocurre la implementación y en la tabla 19 se muestra una proyección a 5 años del flujo efectivo neto para el cálculo del VAN y del TIR.

Tabla 18

Flujo de caja del primer año

Mes	Flujo de ingresos	Flujo de egresos	Flujo de efectivo neto
1	S/ 0.00	-S/ 4,000.00	-S/ 4,000.00
2	S/ 0.00	-S/ 4,000.00	-S/ 4,000.00
3	S/ 0.00	-S/ 4,000.00	-S/ 4,000.00
4	S/ 0.00	-S/ 4,000.00	-S/ 4,000.00
5	S/ 0.00	-S/ 4,000.00	-S/ 4,000.00
6	S/ 1,133.33	-S/ 4,000.00	-S/ 2,866.66
7	S/ 1,133.33	-S/ 4,000.00	-S/ 2,866.66
8	S/ 1,133.33	-S/ 4,000.00	-S/ 2,866.66
9	S/ 1,133.33	-S/ 4,000.00	-S/ 2,866.66
10	S/ 1,133.33	-S/ 4,000.00	-S/ 2,866.66
11	S/ 5,633.33	-S/ 4,000.00	S/ 1,633.33
12	S/ 5,633.33	-S/ 4,000.00	S/ 1,633.33

Nota Se considera como egreso el costo del servidor de pruebas y como ingreso el ahorro del desarrollo y del automatizador de pruebas. Elaboración propia.

Tabla 19

Flujo de efectivo neto proyectado a 5 años

Año	Flujo de efectivo neto
1	-S/ 31,066.69
2	S/ 19,600.00
3	S/ 19,600.00
4	S/ 19,600.00
5	S/ 19,600.00

Nota Elaboración propia.

Para el cálculo del VAN se asume una tasa de descuento del 5%, obteniendo un valor de S/ 29,807.00. Además, se calcula el TIR el cual da como resultado una tasa de retorno del 32%.

CAPITULO IV – CRÍTICA DE LA EXPERIENCIA

Como autor del presente trabajo, en el rol de automatizador de pruebas, resalto de mi aprendizaje y experiencia del proyecto lo siguiente:

- Ideé una estrategia de evolución de la automatización de pruebas, la que inicialmente tenía como alcance solamente la automatización local de flujos principales, llevándolo hasta la ejecución programada en tuberías.
- Tuve la oportunidad de aprender mejores prácticas en programación de automatizaciones, mejorando la calidad de código en cada nuevo proyecto.
- Brindé el conocimiento a los analistas de calidad para que puedan iniciar en el desarrollo de automatización de pruebas, logrando adquirir esta competencia.
- Se aprendió la lección de monitorear los recursos disponibles, ya que se presentó una saturación del servidor de pruebas por la alta demanda de iniciativas, limitando los recursos hasta la adquisición de un nuevo servidor, generando retrasos.
- Tuve un crecimiento en el ámbito profesional al asumir el presente reto después de un cambio de especialidad de desarrollador frontend a automatizador de pruebas.

CAPITULO V – CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- Se consiguió implementar la automatización de pruebas funcionales usando Serenity BDD y patrón de diseño Screenplay, optimizando los tiempos de pruebas en el desarrollo de software en la entidad bancaria.
- Se elaboró un lineamiento interno para la implementación de proyectos de automatización de la división.
- Se integró la automatización con la herramienta Jira XRay para la visualización de reportes en la plataforma Jira.
- Se levantó el servicio de Jenkins en un servidor de pruebas para la ejecución de pruebas automatizadas mediante tuberías.
- Se redujo el tiempo de pruebas en 82.5% y el costo de desarrollo de software en 13.33%.
- Se aumentó la calidad del producto al tener al analista de calidad enfocado en probar nuevos escenarios mientras las pruebas automatizadas se ejecutan, aumentando la cobertura de pruebas.
- Se redujo los posibles errores humanos en su totalidad al automatizar la generación de data de entrada usada en la automatización, e instrucciones claras en caso se agregue data manualmente.
- Se obtuvo una buena retroalimentación de la usabilidad del proyecto de automatización por parte de los analistas de calidad para continuar con nuevas implementaciones.
- Se agregó nuevas responsabilidades al analista de calidad relacionadas a la automatización de pruebas, siendo capaces de cumplir con el nuevo rol.

5.2 Recomendaciones

- Se recomienda implementar nuevos flujos que mejoren la calidad del desarrollo de automatizaciones como la revisión de pares.
- En proyectos que requieran mayor visibilidad del desarrollo de software, se puede trabajar junto al ingeniero de DevOps para realizar un flujo de estados

de tareas en Jira para ejecutar las automatizaciones desde la propia herramienta.

- Se recomienda un repositorio centralizado de dependencias utilizando herramientas como Jfrog Artifactory para que nuevos proyectos de automatización manejen las mismas dependencias y versiones.
- En un siguiente paso, la tubería puede ser integrada al flujo de desarrollo para tener una tubería de integración continua y entrega continua.

Fuentes de información

- Ahamed, I. (7 de febrero de 2023). What Is Use Case Testing: [With Examples]. <https://www.lambdatest.com/learning-hub/use-case-testing>
- Alamo, J. (2022). Automatización del procedimiento de despliegue de APIs en la plataforma Azure API Management para una entidad del sector financiero [Trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos]. Repositorio institucional de la Universidad Nacional Mayor de San Marcos. <https://hdl.handle.net/20.500.12672/18809>
- Ali, H. M., Hamza, M. Y., y Rashid, T. A. (2023). A Comprehensive Study on Automated Testing with The Software Lifecycle. The Journal Of The University Of Duhok, 26(2), 613-620. <https://doi.org/10.26682/csjuod.2023.26.2.55>
- Atlassian. (s.f.). What is Jira? <https://www.atlassian.com/software/jira/guides/getting-started/introduction>
- Ayasta J. (2022). Implementación del framework de automatización Serenity BDD para la ejecución de pruebas de software para la empresa [Trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos]. Repositorio institucional de la Universidad Nacional Mayor de San Marcos. <https://hdl.handle.net/20.500.12672/18754>
- Chavez, P. (2021). Automatización de casos de pruebas usando Selenium y TestNG Framework para mejorar el proceso de pruebas de un sistema que brinda servicios RPO, 2020 [Trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos]. Repositorio institucional de la Universidad Nacional Mayor de San Marcos. <https://hdl.handle.net/20.500.12672/17023>
- Cucumber. (s.f.). What is Cucumber? <https://cucumber.io/docs/guides/overview/>
- Delgado I. (2024). Implementación de pruebas automatizadas usando Selenium, Scrum y ISTQB para optimizar los tiempos de ejecución de pruebas en una entidad bancaria [Trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos]. Repositorio institucional de la Universidad Nacional Mayor de San Marcos. <https://hdl.handle.net/20.500.12672/22562>
- Horina, K., y Oleksandr, K. (2024). Advantages of Automated Testing of Medical Applications and Information Systems Using Gherkin and Behavior-Driven Development. En Lecture notes in networks and systems (pp. 379-391). https://doi.org/10.1007/978-3-031-60549-9_28

- Jenkins. (s.f.). Pipeline. <https://www.jenkins.io/doc/book/pipeline/>
- Kishan, S., y Pavithra, H. (2021). Test Case Recording using Java Script for Automation Testing. International Journal Of Recent Technology And Engineering (IJRTE), 10(1), 153-157. <https://doi.org/10.35940/ijrte.a5810.0510121>
- Lucero J. (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 [Trabajo de suficiencia profesional, Universidad Nacional Mayor de San Marcos]. Repositorio institucional de la Universidad Nacional Mayor de San Marcos. <https://hdl.handle.net/20.500.12672/18814>
- Pal, K., y Karakostas, B. (2021). Software Testing Under Agile, Scrum, and DevOps. En Research Anthology on Agile Software, Software Development, and Testing (pp.1059-1076). <https://doi.org/10.4018/978-1-6684-3702-5.ch053>
- Perez-Serrano, M. (2020). Gestión de Pruebas de Aceptación integrada en una metodología y herramienta para la gestión ágil de proyectos [Trabajo académico, Universidad Politécnica de Valencia]. Repositorio institucional de la Universidad Politécnica de Valencia. <http://hdl.handle.net/10251/152245>
- Pesantes, C. (2023). Automatización de pruebas de aceptación en el proceso de desarrollo de software [Tesis de maestría, Pontificia Universidad Católica del Perú]. Repositorio institucional de la Pontificia Universidad Católica del Perú. <http://hdl.handle.net/20.500.12404/26358>
- Schwaber K., y Sutherland J. (noviembre de 2020). The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>
- Serenity BDD. (s.f.). Introduction. https://serenity-bdd.github.io/docs/guide/user_guide_intro
- Toan D. (13 de junio de 2024). An Ultimate Guide to Different Test Case Design Techniques. <https://reliasoftware.com/blog/test-case-design-techniques>
- Yuniasri, D., Badriyah, T., y Sa'adah, U. (2020). A Comparative Analysis of Quality Page Object and Screenplay Design Pattern on Web-based Automation Testing. 2019 International Conference On Electrical, Communication, And Computer Engineering (ICECCE). <https://doi.org/10.1109/icecce49384.2020.9179470>

Glosario

- Archivo batch: Archivo de texto para ejecutar una secuencia de comandos el cual es ejecutado por el programa cmd.exe o Símbolo de sistema
- DevOps: Es la combinación del desarrollo de software y de las operaciones de tecnología de información.
- Ingeniero de DevOps: Especialista en unificar ambos enfoques de desarrollo y operaciones.
- Integración continua y entrega continua: Conocido como CI/CD, estos son un enfoque que buscan la entrega de valor de manera más frecuente mediante la automatización del desarrollo, de las pruebas y del despliegue del software.
- Jfrog Artifactory: Es una solución para alojar y gestionar artefactos, los cuales pueden ser instaladores, contenedores, librerías, archivos de configuración, etc.
- Log: Es un archivo que guarda eventos como errores o información.
- Plugin: Complementos que amplían o añaden funcionalidades a las aplicaciones.
- Revisión de pares: Evaluación donde un experto valida la calidad del trabajo de otro experto de una misma especialidad.
- Script: Es una secuencia de instrucciones que usan un lenguaje especializado para scripts y son interpretados por otros programas para ser ejecutados.
- Webdriver: Es una interfaz que permite la interacción con el navegador.