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

Un marco de trabajo para el diseño e implementación de pruebas de software aplicadas a una API desarrollada en Scrum

INFORME DE TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Silvana Yasmin MALPICA MARTINEZ

ASESOR

Luis ALARCÓN LOAYZA

Lima, Perú

2017

Referencia bibliográfica

Malpica, S. (2017). *Un marco de trabajo para el diseño e implementación de pruebas de software aplicadas a una API desarrollada en Scrum*. Informe de Trabajo de Suficiencia Profesional para optar el título profesional de Ingeniero de Software. Escuela Profesional de Ingeniería de Software, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima, Perú.

DEDICATORIA:

Dedico este trabajo a mi familia que siempre me ha apoyado y acompañado en el camino hacia mis sueños.

AGRADECIMIENTOS

Quiero agradecer a mi asesor por sus recomendaciones y consejos a lo largo del desarrollo de este informe. Así como también, a los docentes que compartieron su conocimiento conmigo durante mi carrera universitaria.

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

UN MARCO DE TRABAJO PARA EL DISEÑO E IMPLEMENTACIÓN DE PRUEBAS DE SOFTWARE APLICADAS A UNA API DESARROLLADA EN SCRUM

Autor: MALPICA MARTINEZ, Silvana Yasmin
Asesor: ALARCÓN LOAYZA, Luis
Título: Informe, para optar el Título Profesional de Ingeniero de Software
Fecha: Diciembre del 2017

RESUMEN

El presente informe de experiencia profesional describe el diseño e implementación de las pruebas que fueron necesarias para garantizar la calidad requerida de la API de un sistema de análisis de tráfico de clientes. El proyecto de desarrollo se realizó siguiendo las buenas prácticas de la metodología Scrum, en el área de Entrega de Servicios de la empresa Belatrix. Las pruebas para una API son muy diferentes de las que se realizan en las aplicaciones front-end y son de igual o mayor importancia dado que manejan el core del negocio. Para este proyecto se desarrolló tanto pruebas funcionales como no funcionales. Dentro las de las pruebas funcionales, se automatizaron los escenarios correspondientes con las pruebas de regresión. Se utilizó Java para escribir los scripts y TestNG como framework de las pruebas automatizadas. En el ámbito de la performance, se realizaron pruebas de rendimiento, de carga y de estrés con la herramienta JMeter. Adicionalmente, debido a que la fuente de los datos que recibe la API es inestable, se vio necesario construir una herramienta que compruebe la exactitud de la data antes de ser procesada por la API. Con miras a, en un futuro, tener un sistema independiente de pruebas que realice verificaciones no contempladas en las pruebas funcionales y no funcionales y alerte al cliente de posibles defectos antes de que estos sean visibles para el usuario final, esta herramienta se construyó como un web service. La suma de todas estas actividades alcanzo el objetivo de brindar la calidad requerida por el cliente para su API.

Palabra claves: pruebas de software, proceso de pruebas, pruebas funcionales, pruebas no funcionales, pruebas automatizadas

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

A FRAMEWORK FOR THE DESIGN AND IMPLEMENTATION OF SOFTWARE TESTING APPLIED TO AN API DEVELOPED IN SCRUM

Author: MALPICA MARTINEZ, Silvana Yasmin
Advisor: ALARCÓN LOAYZA, Luis
Title: Report, to achieve the Professional Title of Software Engineer
Date: December 2017

ABSTRACT

The present professional experience report describes the design and implementation of the tests that were necessary to guarantee the quality required in the API of a Clients Traffic Analytics System. The development project was carried out following the good practices of the Scrum methodology, in the Service Delivery area at Belatrix software factory. Testing in an API is very different from testing in front-end applications. This is equal or higher importance of the mayor who manage the core of the business. For this project both functional and non-functional tests were developed. Within the functional tests, the scenarios with the regression tests have been automated. Java was used to write scripts and TestNG as a framework for automated testing. In the field of execution, performance, load and stress tests were performed with the JMeter tool. Additionally, because the source of the information received by the API is unstable, it was necessary to construct a tool that checks the accuracy of the data before being processed by the API. In the future, with an integrated system with other verification functions (which are not included in the functional and non-functional tests) that alert the customer to possible defects before they are visible to the end user, this tool was built as a web service. The sum of all these measures reached the objective of providing the quality required by the client for the API.

Key words: software testing, testing process, functional testing, non-functional testing, automated tests

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El presente informe de experiencia profesional describe el proceso de control de la calidad de una API durante su mantenimiento y actualización que se llevó a cabo siguiendo las buenas prácticas de la metodología Scrum en la empresa Belatrix.

La empresa Belatrix es una fábrica de software que maneja Scrum a gran escala y lo utiliza en todos los proyectos de su área de Entrega de Servicios. Belatrix concentra sus clientes en el territorio americano, teniendo a los Estados Unidos de América como principal mercado. Actualmente cuenta con 6 sucursales distribuidas entre Perú, Argentina, Colombia y Estados Unidos.

Dentro del área de Entrega de Servicios de la empresa Belatrix, se desarrolló un proyecto para una empresa extranjera, el cual consistió en la actualización de la API principal de un sistema que transforma los datos del tráfico de clientes en tiendas retail, en indicadores y métricas útiles para la toma de decisiones empresariales por parte de los dueños y administradores de las tiendas retail y centros comerciales. Siendo este sistema de explotación de datos, uno de los servicios más lucrativos que ofrece la empresa cliente, el control de la calidad durante la implementación de los cambios es fundamental.

En este informe se detalla la experiencia profesional del autor y el proceso de planificación, diseño e implementación de las pruebas requeridas y necesarias para garantizar la calidad de la API a través del ciclo de desarrollo.

En el CAPÍTULO I se especifican cronológicamente roles y funciones, actividades y el aprendizaje formal del autor. Además, se precisa la experiencia laboral significativa.

En el CAPÍTULO II se describe parte de la historia de la empresa Belatrix Perú S.A.C., su estructura orgánica, la visión y la misión, sus valores y los productos y los servicios que brinda.

El CAPÍTULO III contiene el detalle del proyecto realizado para sustentar la experiencia profesional, el cual se refiere a la planificación, análisis, diseño y la implementación de las pruebas necesarias para garantizar la calidad de la API principal del sistema de explotación de datos y el diseño y la implementación de herramientas a medida para incrementar la cobertura las pruebas y atender necesidades específicas de la empresa cliente, dentro del área

de Entrega de Servicios de la empresa Belatrix. Básicamente, este capítulo incluye todo el proceso de pruebas realizado a lo largo del proyecto de actualización y mantenimiento de la API.

El CAPÍTULO IV se refiere al aporte del autor, el desarrollo profesional que le demandó su participación en el presente proyecto, las necesidades que se atendieron y la experiencia obtenida.

En el CAPÍTULO V se presentan las conclusiones y también se indican las recomendaciones finales.

CAPÍTULO I - TRAYECTORIA PROFESIONAL

Cuento con el grado de bachiller en Ingeniería de Software de la Universidad Nacional Mayor de San Marcos. Cuento con cerca de 3 años de experiencia profesional y un año de experiencia pre-profesional entre practicas pre-profesionales y profesionales. Inicié mi vida profesional como desarrolladora en Java y PHP, sin embargo, después de un corto tiempo decidí orientarme netamente al aseguramiento de la calidad del software y las pruebas de software. Mi gusto por la programación me llevo a convertirme en una Automation Tester, quien se encarga no solo del diseño y ejecución de pruebas manuales, sino también del diseño e implementación de pruebas automatizadas. Me he desempeñándome en los últimos años como Ingeniera de calidad para diferentes proyectos de clientes americanos utilizando la metodología Scrum. Tengo nivel de inglés avanzado y obtuve la categoría C1 en el Michigan English Test. Además, cuento con 3 certificaciones internacionales relacionadas a Scrum (fundamentos, desarrollador y Scrum Master) y una certificación internacional ISTQB en el nivel de fundamentos de pruebas.

La trayectoria profesional se detalla a continuación:

EXPERIENCIA PROFESIONAL	
Ingeniera de Calidad BELATRIX SOFTWARE FACTORY Migración de usuarios y actualización del sistema autenticación. - Diseño y ejecución de pruebas funcionales y automatizadas (unit test, feature test, regression test). - Implementación de pruebas automatizadas con Java y TestNg. - Diseño y ejecución de pruebas de performance para servicios con JMeter, SoapUI, Postman. - Especificación de criterios de aceptación. - Implementación de integración continua.	Noviembre 2016 – Actualidad
Ingeniera de calidad BELATRIX SOFTWARE FACTORY Sistema de explotación de datos para tiendas retail	Abril – Octubre 2016

- Diseño y ejecución de pruebas funcionales y automatizadas (unit test, feature test, regression test). - Implementación de pruebas automatizadas con Java y TestNg. - Diseño y ejecución de pruebas de performance para servicios con JMeter, SoapUI, Postman. - Especificación de criterios de aceptación. - Construcción de un web service para verificación de datos en Java con Spring boot.	
Ingeniera de calidad BELATRIX SOFTWARE FACTORY Sistema de administración de ejecuciones financieras - Diseño y ejecución de pruebas funcionales y automatizadas (feature test, regression test). - Implementación de pruebas de UI automatizadas con Silk test. - Diseño y ejecución de pruebas de performance con SoapUI.	Abril – Marzo 2016
Practicante de Ingeniería de calidad BELATRIX SOFTWARE FACTORY Sistema de incentivos para empleados - Diseño y ejecución de pruebas funcionales y automatizadas (feature test, regression test). - Implementación de pruebas de UI automatizadas con Selenium Web Driver.	Junio – Marzo 2015
Practicante de desarrollo Full stack GRUPO LIDERA Aula Virtual - Toma de requisitos. - Análisis y diseño del sistema.	Enero – Marzo 2013

- Diseño de base de datos. Desarrollo de las interfaces del sistema con html, javascript y CSS3. - Desarrollo del back-end en php.	
Analista programador Java QUIPUCAMAYOC UNMSM Sistema de rastreo de procesos documentario - Mantenimiento y desarrollo de nuevas características de un sistema que permite rastrear los trámites en proceso dentro las oficinas administrativas de la Universidad Nacional Mayor de San Marcos. - Corregir errores en el sistema elaborado con Spring MVC, Mybatis y Java Server Faces.	Enero – Marzo 2012
Analista programador Java QUIPUCAMAYOC UNMSM Sistema de Contabilidad - Mantenimiento del sistema de contabilidad de la Universidad Nacional Mayor de San Marcos en Visual Basic. - Corregir errores reportados por los usuarios finales.	Enero – Marzo 2012

FORMACIÓN	
Educación Superior Grado Académico de Bachiller en Ingeniería de Software – de la Escuela Académico Profesional de Ingeniería de Software – Facultad de Ingeniería de Sistemas e Informática – Universidad Nacional Mayor de San Marcos.	Marzo 2010 – Diciembre 2014
Educación Secundaria Institución Educativa Nacional Nuestra Señora de Lourdes	Abril 2004 – Diciembre 2008

IDIOMAS	
Inglés – Nivel Avanzado Instituto cultural peruano norteamericano	Junio 2013 – Enero 2016

CURSOS	
Curso para la certificación en Metodología Scrum Instituto GESAP	Marzo 2017
Curso para la certificación ISTQB Belatrix Software Factory	Enero 2017
Integración Continua Kleer	Febrero 2016
Unit Testing avanzado Kleer	Febrero 2016
Test automation survival camp Kleer	Febrero 2016

CERTIFICACIONES	
Scrum Master certificada	Mayo 2017
Scrum Developer certificada	Abril 2017
Certificación en fundamentos de Scrum	Marzo 2017
ISTQB Tester certificada	Marzo 2017
Examen MET (Michigan English Test) aprobado con categoría C1	Marzo 2016

OTROS CONOCIMIENTOS	
Lenguajes de programación	Java, C#, PHP
Bases de Datos	Oracle, Mysql, SQL Server, Mongo DB
Pruebas Unitarias	Nunit, Nsubstitute, Junit, Mockito, TestNG
Pruebas Automatizadas	Selenium IDE, Selenium Web driver, Silk test, Cucumber
Pruebas de Performance	SoapUI, JMeter

Controlador de versiones	Git, SVN
Repositorios en la nube	Bitbucket, Github
Integración continua	Jenkins, Teamcity
Issue trackers	Gemini, Trello, Jira
Metodologías	Scrum

CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLÓ LA EXPERIENCIA

2.1. Empresa - Actividad que realiza

Es una empresa transnacional Argentina que ofrece diversos servicios de desarrollo de software principalmente a compañías americanas desde sedes en Latinoamérica dónde se encuentran los profesionales mediante los cuales se proveen los servicios. Belatrix ha entregado soluciones tecnológicas a medida por más de 25 años desde su nacimiento, siempre utilizando las mejores metodologías de outsourcing.

Belatrix provee los siguientes servicios: desarrollo de software, aseguramiento de la calidad y soporte de TI. Actualmente cuenta con 1 sede de marketing en Silicon Valley (California - Estados Unidos) y 5 sedes de Entrega de Servicios: 3 locales en Argentina (2 en Mendoza y 1 en Buenos Aires), un local en Lima - Perú y un local en Bogotá - Colombia.

Trabaja bajo estándares internacionales de calidad como ISO 9001 y CMMI. En el aspecto de seguridad, tiene implementada la ISO 27001. Actualmente la empresa está acreditada en el nivel 3 del CMMI.

2.1.1. Datos de la Empresa

- **Razón social:** BELATRIX PERÚ S.A.C.
- **Domicilio Legal:** Avenida República de Panamá 3591 - San Isidro
- **Teléfono:** 7173350
- **RUC:** 20392810316
- **Antigüedad en Perú:** 15 años
- **Mercado:** Compañías americanas que buscan servicios de outsourcing para sus áreas de tecnología de la información.

2.2. Visión

Nuestra visión es ser reconocidos como la mejor firma de innovación y desarrollo de productos software de américa latina.

2.3. Misión

Nuestra misión es ser un socio global apasionado que ofrece innovación de software.

2.4. Valores

Los valores de la organización son los siguientes:

- Excelencia
- Pasión
- Compromiso
- Trabajo en equipo
- Empoderamiento y confianza
- Innovación

2.5. Organización de la Empresa

A continuación, se muestra el organigrama de la empresa Belatrix:

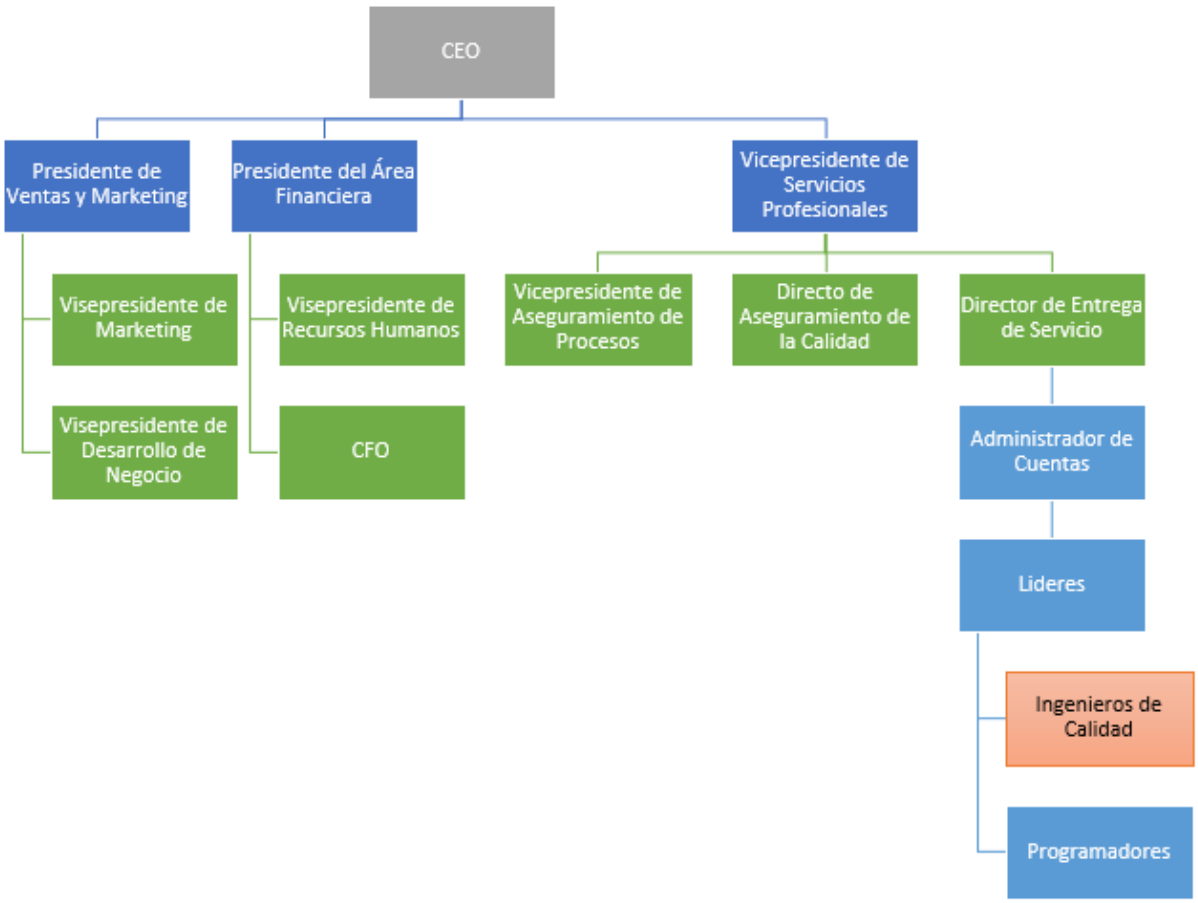

Figura 1. Organigrama de la empresa Belatrix

2.6. Área, Cargo y Funciones desempeñadas

La autora de este Informe de Experiencia Profesional se desempeñó como Ingeniera de Aseguramiento de la Calidad dentro del equipo Scrum donde fue asignada. A lo largo de su permanencia en la empresa ha participado en 4 diferentes proyectos, todos bajo la metodología Scrum. Los equipos Scrum pertenecen al área de Entrega de Servicios de la empresa Belatrix. Las funciones desempeñadas por la autora fueron: realizar el análisis, diseño e implementación de pruebas manuales y automatizadas en medida de las necesidades y exigencias de cada proyecto.

2.7. Experiencia profesional realizada en la organización

Las funciones realizadas fueron:

- Especificación de requisitos funcionales y no funcionales.
- Especificación de criterios de aceptación.
- Análisis y diseño de casos de prueba.
- Ejecución de casos de prueba manuales y automatizados (pruebas de humo, de característica y de regresión).
- Implementación de scripts automatizados para pruebas de regresión.
- Reportar y documentar incidencias encontradas durante las pruebas.
- Diseño e implementación de pruebas de performance (concurrencia, rendimiento, carga y estrés).
- Preparar informes legibles y didácticos con los resultados de las pruebas de performance.
- Implementación de integración continua.
- Construcción de herramientas de verificación para necesidades específicas.

CAPÍTULO III – ACTIVIDADES DESARROLLADAS

3.1. Situación problemática

3.1.1. Definición del problema

Una empresa extranjera dedicada a ofrecer herramientas para el análisis de tráfico de clientes en tiendas retail planeó la refactorización y actualización de uno de sus productos software más solicitados, con el objetivo de mejorar su mantenibilidad, escalabilidad y rendimiento. Este sistema fue desarrollado en tiempo record por un grupo de programadores que luego de terminar el proyecto fueron desvinculados de la organización. El tiempo otorgado al desarrollo de este proyecto fue muy corto, debido a que las empresas de la competencia liberaron o estaban por liberar un producto similar. En un intento de no perder competitividad, la empresa permitió un desarrollo desorganizado y con documentación casi inexistente.

Como resultado se obtuvo un sistema funcional pero sumamente frágil y susceptible a errores. Su mantenibilidad era complicada y su escalabilidad bastante baja. Además, se habían reportado quejas por parte de los usuarios finales acerca del tiempo de respuesta del sistema en horas específicas del día. A pesar de ello, el servicio que brinda este sistema incrementó su popularidad con el tiempo, convirtiéndose en el sistema más solicitado de la compañía y, por ende, en el más lucrativo. De esto se desprende que la mejora de este sistema tenía una alta prioridad para la organización y era bastante crítico para el negocio.

Dado que el proyecto estaba por alcanzar el presupuesto máximo asignado, la empresa cliente tomó la decisión de tercerizar el equipo de calidad para reducir costos. Belatrix fue la empresa seleccionada debido a la confianza que existía por el éxito de proyectos anteriores con esta misma compañía. Además, Belatrix tiene muchos años de experiencia en desarrollo ágil, el cual es el mismo enfoque de desarrollo de la empresa cliente. Belatrix proveyó este servicio mediante la participación de una ingeniera de calidad, quién es la autora del presente informe.

La empresa cliente solicitó explícitamente centrar los esfuerzos de las pruebas en la API que comunica los servicios con la interfaz gráfica y excluir las pruebas para otros componentes del sistema. Adicionalmente, el proceso de pruebas debía alinearse a la metodología de desarrollo de la empresa cliente, la cual fue Scrum.

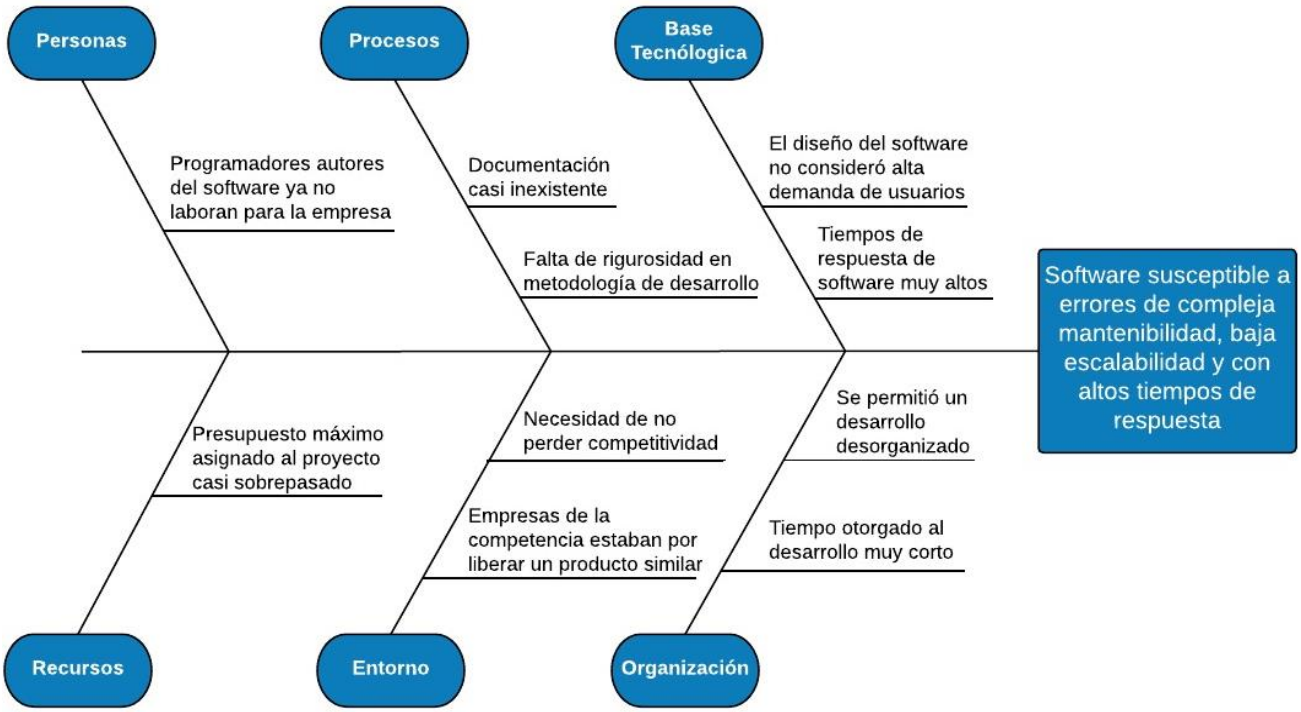

Figura 2. Diagrama de Ishikawa sobre las causas y efectos del problema

3.2. Solución

En esta sección se va a presentar la solución partiendo de la justificación para superar el problema. En este contexto se define el objetivo general y los objetivos específicos.

3.2.1. Objetivos

3.2.1.1. Objetivo General

Desarrollar un marco de trabajo para el diseño e implementación de pruebas de software aplicadas a la API del Sistema de análisis de trafico de clientes desarrollada en Scrum.

3.2.1.2. Objetivos Específicos

- Diseñar y ejecutar las pruebas funcionales.
- Diseñar, implementar y ejecutar las pruebas automatizadas.
- Diseñar y ejecutar las pruebas no funcionales.
- Reportar el resultado de las pruebas.
- Diseñar, implementar y ejecutar una herramienta de verificación a medida.

3.2.2. Alcance

El alcance de este informe cubre la planificación, diseño e implementación de las pruebas funcionales a nivel de integración y no funcionales a nivel de sistema, realizadas durante el proceso de desarrollo para asegurar la funcionalidad y el rendimiento del sistema de análisis de tráfico de clientes a nivel de software.

3.2.3.1. Fuera del alcance

- No incluye procesos de aseguramiento de la calidad.
- No se incluyen pruebas para los cambios en la infraestructura del sistema.
- Todo lo que no está especificado en el alcance.

3.2.3. Metodología

La metodología de desarrollo fue Scrum. El equipo Scrum estuvo conformado por dos desarrolladores, una ingeniera de calidad, una Scrum Master y un Product Owner. Se planifico un tiempo inicial de dos semanas para la capacitación del equipo y la planificación general del proyecto. Durante este primer periodo, que se denominó “Sprint 0”, se definieron aspectos importantes de la metodología como la longitud de los sprints, las reuniones, las columnas de la pizarra de scrum, las historias de usuario con los requisitos de los cambios a alto nivel con sus respectivos criterios de aceptación y el software de gestión para las tareas de Scrum.

El Sprint 0 constó de dos semanas de capacitaciones continuas a través de reuniones directas con el Product Owner y otros especialistas del negocio y técnicos. También el equipo Scrum se auto-capacitó leyendo documentación y revisando videos alojados en la wiki del cliente en Confluence (repositorio en la nube). Los desarrolladores tuvieron largas reuniones de análisis del código legacy y la ingeniera de calidad se encargó de diseñar el plan de pruebas. El Product Owner decidió utilizar la herramienta Jira para escribir las historias de usuario y los desarrolladores lo apoyaron en esta actividad, aportando su perspectiva técnica. Además, la ingeniera de calidad y el Product Owner trabajaron juntos en la definición de los criterios de aceptación. Por solicitud explicita de la empresa cliente, la documentación en este proyecto sería mínima y se almacenaría en Jira o Confluence.

Se definió que la longitud de los sprints sería de 2 semanas. Se tendrían reuniones diarias de 15 minutos para compartir el estado de las tareas e informar los impedimentos. También se realizarían reuniones de planificación al inicio de cada sprint en las cuales se definían las historias de usuario a realizar y quiénes las realizarían. Adicionalmente, al finalizar cada sprint, se tendría una reunión de demostración de los resultados del sprint en la cual se definiría si se realizaba o no el despliegue a producción de la versión demostrada y luego una sesión de retrospectiva. Todo esto siguiendo las buenas prácticas de la metodología.

Figura 3. Reuniones durante cada sprint

Durante el Sprint 0 también se definió el flujo de trabajo. Al inicio de cada sprint se seleccionaban las tareas a implementar durante esa iteración y eran desarrolladas por los programadores. En paralelo, la ingeniera de calidad se encarga de escribir los casos de prueba asociados a la funcionalidad en caso no se hayan diseñado antes. Cuando las tareas eran completadas debían ser revisadas por el desarrollador que no programo la tarea. Luego eran probadas por la ingeniera de calidad, quien se encargaba de dar el visto bueno a una tarea para que se pueda considerar como terminada. Si las tareas no cumplían con los criterios de aceptación, estas debían ser retornadas a los programadores para que corrijan los errores reportados por la ingeniera de calidad. Cuando las tareas de diseño de casos de prueba estén terminadas y la ingeniera a de calidad no tenga ninguna tarea lista para probar (este escenario sucedió a partir del tercer sprint en adelante, sobre todo en los primeros días del sprint) ella se encargaba de trabajar en las tareas de automatización y de pruebas de performance.

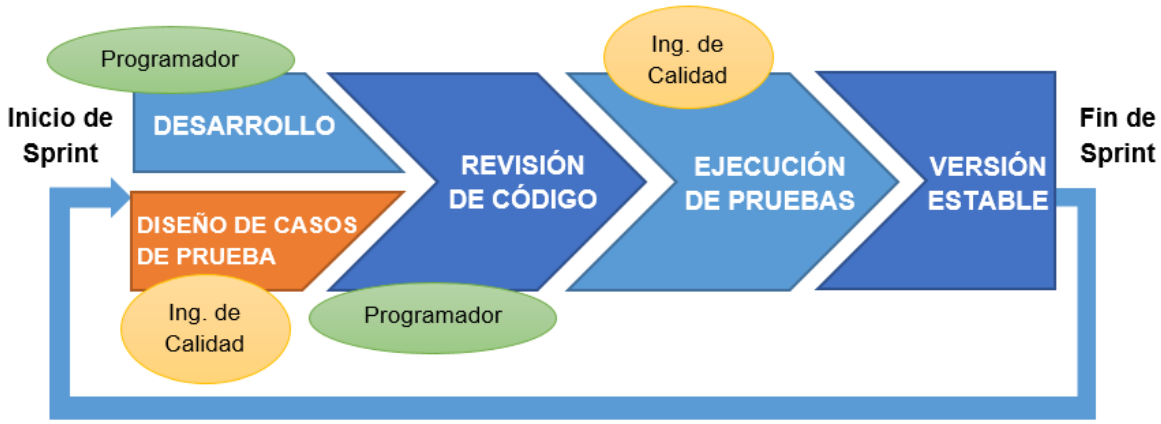

Figura 4. Las pruebas de cada tarea durante el sprint

Para plasmar este proceso en la pizarra de Scrum, se definieron 6 columnas: “Por hacer” (TO DO), “En desarrollo” (IN DEV), “En revisión” (CODE REVIEW), “Listo para probar” (READY FOR QA), “En prueba” (QA) y “Terminado” (DONE). La columna “Por hacer” se creó para contener las tareas asignadas al sprint activo y que no se encuentran en desarrollo. La columna “En desarrollo” era para las tareas que se encontraban en desarrollo y debían tener un programador asignado. La columna “En revisión” era para las tareas cuyo desarrollo había finalizado y estaban esperando la aprobación del desarrollador que no trabajo en ella. La columna “Listo para probar” contenía las tareas cuya revisión fue aprobada y por lo tanto estaban listas para ser probadas. La columna “En prueba” era para las tareas que estaban en etapa de pruebas y la columna “Terminado” contenía solo las tareas que pasaron exitosamente las pruebas.

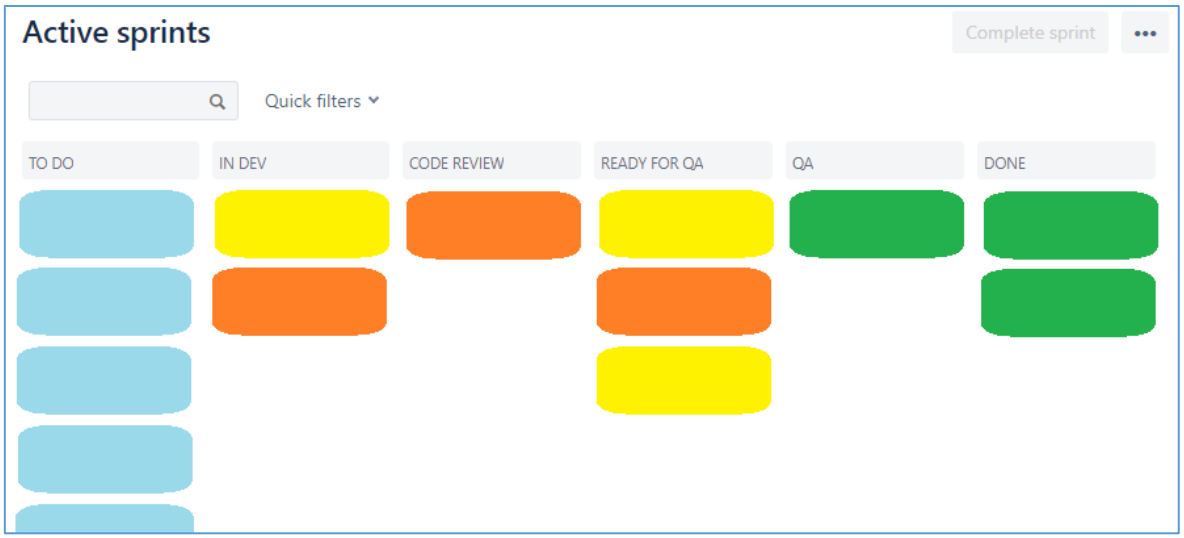

Figura 5. Pizarra de Scrum en Jira

Por lo tanto, el flujo de la pizarra de Scrum fue el siguiente: Los programadores se asignan las tareas y las mueven de “Por hacer” a “En desarrollo” una vez que comienzan a trabajar en ellas y cuando terminan las ubican en “En revisión”. El programador que no desarrollo la tarea se encarga de la revisión del código y si todo se encuentra en orden mueven esta tarea a “Listo para probar”, de lo contrario regresa a “En desarrollo”. La ingeniera de calidad se asigna las tareas en “Listo para probar” y las mueve a “En prueba” cuando comienzan a probarlas. Si las pruebas son exitosas, estas pasan a “Terminado”. De lo contrario, se reporta el error y se regresa la tarea a “En desarrollo”.

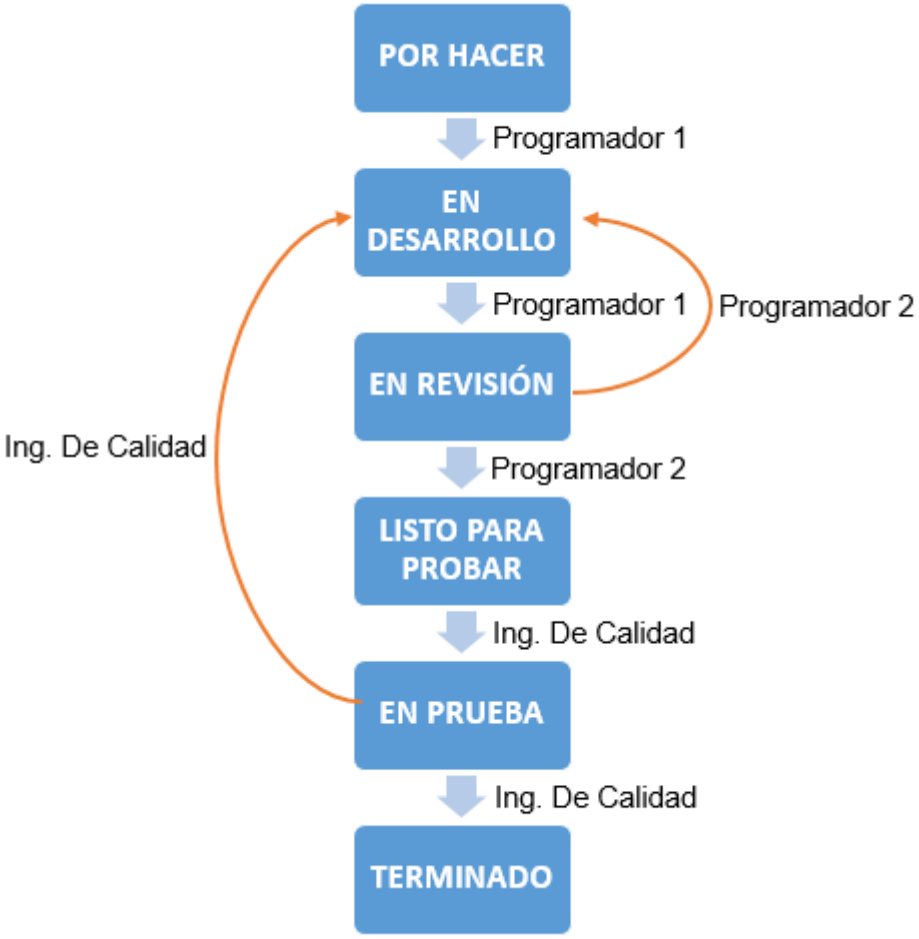

Figura 6. Flujo de los estados de las tareas en la pizarra de Scrum

3.2.4. Fundamentos Utilizados

3.2.4.1. Metodología Scrum

Scrum es uno de los métodos ágiles más populares. Es un framework adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer un valor considerable en forma rápida a lo largo del proyecto. Scrum garantiza transparencia en la comunicación y crea un ambiente de

responsabilidad colectiva y de progreso continuo. El framework de Scrum, tal como se define en la Guía SBOK™, está estructurado de tal manera que es compatible con el desarrollo de productos y servicios en todo tipo de industrias y en cualquier tipo de proyecto, independientemente de su complejidad.

Una fortaleza clave de Scrum radica en el uso de equipos interfuncionales (cross-functional), autoorganizados y empoderados que dividen su trabajo en ciclos de trabajo cortos y concentrados llamados Sprints.

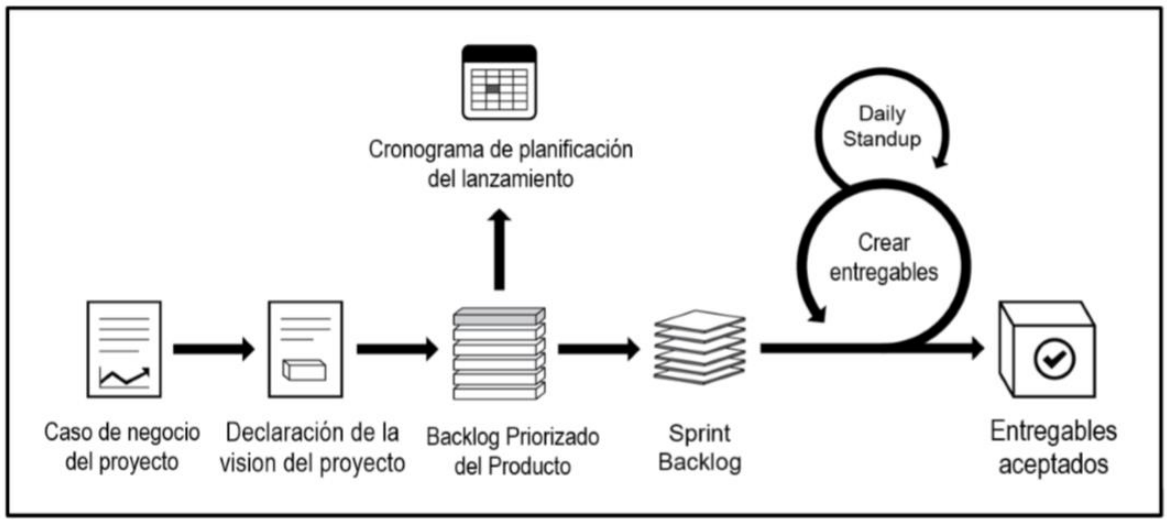

Figura 7. Visión general de flujo de un proyecto Scrum. (Guía SBOK™, 2016)

El ciclo de Scrum empieza con una reunión de stakeholders, durante la cual se crea la visión del proyecto. Después, el Product Owner desarrolla una Backlog Priorizado del Producto (Prioritized Product Backlog) que contiene una lista requerimientos del negocio y del proyecto por orden de importancia en forma de una historia de usuario. Cada sprint empieza con una reunión de planificación del sprint (Sprint Planning Meeting) durante la cual se consideran las historias de usuario de alta prioridad para su inclusión en el sprint. Un sprint generalmente tiene una duración de una a seis semanas durante las cuales el Equipo Scrum trabaja en la creación de entregables (del inglés deliverables) en incrementos del producto. Durante el sprint, se llevan cabo Daily Standups muy breves y concretos, donde los miembros del equipo discuten el progreso diario. Haca el final del sprint, se lleva a cabo una Reunión de Revisión del Sprint (Sprint Review Meeting) en la cual se proporciona una demostración de los entregables al Product Owner y a los stakeholders relevantes. El Product Owner acepta los entregables sólo si cumplen con los criterios de aceptación predefinidos.

El ciclo del sprint termina con una Reunión de Retrospectiva del Sprint (Retrospect Sprint Meeting), donde el equipo analiza las formas de mejorar los procesos y el rendimiento a medida que avanzan al siguiente sprint. (Guía SBOK™, 2016)

Principios de Scrum

Los principios de Scrum son las pautas básicas para aplicar el framework de Scrum y deben implementarse en forma obligatoria en todos los proyectos Scrum. Los seis principios de Scrum son los siguientes:

1. **Control del proceso empírico:** Este principio enfatiza la filosofía central de Scrum con base a las tres ideas principales de transparencia, inspección y adaptación.

2. **Auto-organización:** Este principio se enfoca en los trabajadores de hoy en día, que entregan un valor considerablemente mayor cuando se auto-organizan, lo cual resulta en equipos que poseen un gran sentido de compromiso y responsabilidad; a su vez, esto produce un ambiente innovador y creativo que es más propicio para el crecimiento.

3. **Colaboración:** Este principio se centra en las tres dimensiones básicas relacionadas con el trabajo colaborativo: conocimiento, articulación y apropiación. También fomenta la gestión de proyectos como un proceso de creación de valor compartido con equipos que trabajan e interactúan conjuntamente para ofrecer el mayor valor.

4. **Priorización basada en valor:** Este principio pone de relieve el enfoque de Scrum para ofrecer el máximo valor de negocio, desde el principio del proyecto hasta su conclusión.

5. **Time-boxing:** Este principio describe cómo el tiempo se considera una restricción limitante en Scrum, y cómo este se utiliza para ayudar a manejar eficazmente la planificación y ejecución del proyecto. Los elementos del time boxing en Scrum incluyen sprints, Daily Standups, reuniones de planificación del sprint y reuniones de revisión del sprint.

6. **Desarrollo iterativo:** Este principio define el desarrollo iterativo y hace énfasis en cómo gestionar mejor los cambios y crear productos que satisfagan las necesidades del cliente. También delinea las responsabilidades del Product Owner y las de la organización relacionadas con el desarrollo iterativo.

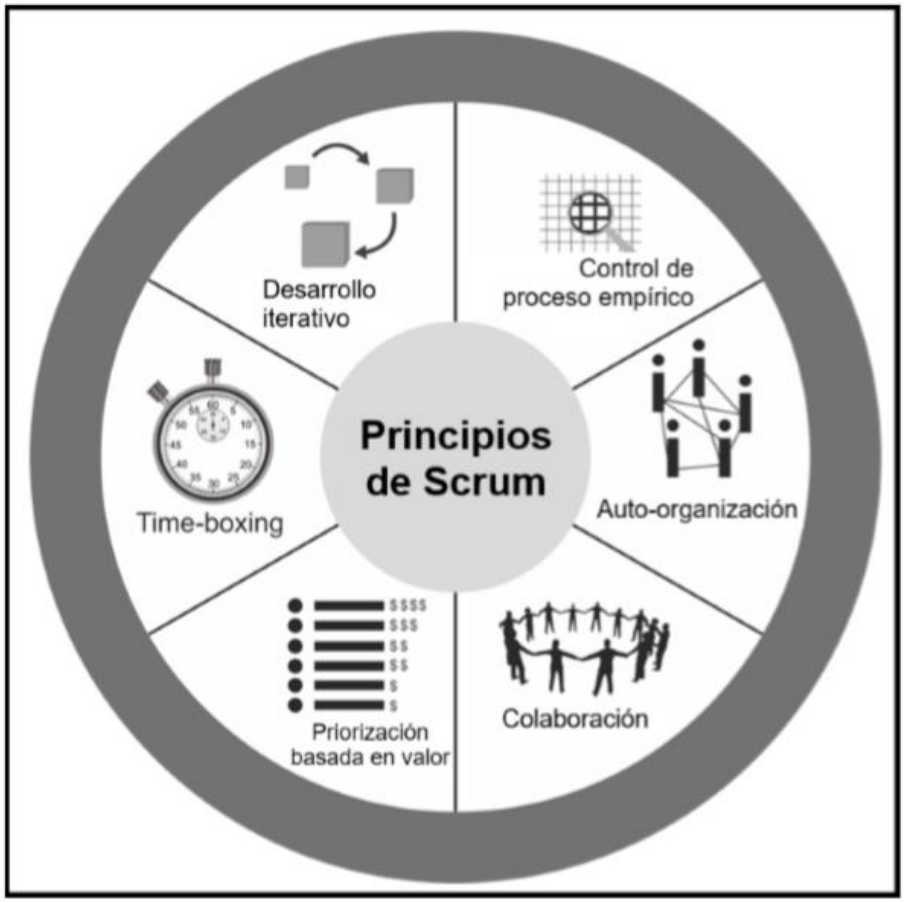

Figura 8. Los principios de Scrum (Guía SBOK™, 2016)

Aspectos de Scrum

Los aspectos de Scrum deben abordarse y gestionarse durante todo un proyecto Scrum. Los cinco aspectos de Scrum son los siguientes:

1. **Organización:** Entender los roles y responsabilidades definidos en un proyecto Scrum es muy importante a fin de asegurar la implementación exitosa de Scrum. Los roles de Scrum se dividen en roles centrales y no centrales:

   Los roles centrales son los cuales se requieren obligadamente para crear el producto o servicio del proyecto. Las personas a quienes se les asignan los roles centrales están

plenamente comprometidas con el proyecto y son las responsables del éxito de cada iteración del mismo, así como del proyecto en su totalidad. Estos roles incluyen:

- El Product Owner es la persona responsable de lograr el máximo valor empresarial para el proyecto. Este rol también es responsable de la articulación de requisitos del cliente y de mantener la justificación del negocio para el proyecto. El Product Owner representa la voz del cliente.

- El Scrum Master es un facilitador que asegura que el Equipo Scrum cuente con un ambiente propicio para completar el proyecto con éxito. El Scrum Master guía, facilita y enseña las prácticas de Scrum a todos los involucrados en el proyecto; elimina los impedimentos que pueda tener el equipo y se asegura de que se estén siguiendo los procesos de Scrum.

- El Equipo Scrum es el grupo o equipo de personas responsables de entender los requisitos especificados por el Product Owner y de crear los entregables del proyecto.

Los roles no centrales son los que no son necesariamente obligatorios para el proyecto Scrum, y estos pueden incluir a miembros de los equipos que estén interesados en el proyecto. No tienen ningún rol formal en el equipo del proyecto, y pueden interactuar con el equipo, pero pueden no ser responsables del éxito del proyecto. Los roles no centrales deben tenerse en cuenta en cualquier proyecto de Scrum y son los siguientes:

- Stakeholder(s) es un término colectivo que incluye a clientes, usuarios y patrocinadores, que con frecuencia interactúan con el equipo principal de Scrum, e influyen en el proyecto a lo largo de su desarrollo. Lo más importante es que el proyecto produzca beneficios colaborativos para los stakeholders.

- El Scrum Guidance Body (SGB) es un rol opcional, que generalmente consiste en un conjunto de documentos y/o un grupo de expertos que

normalmente están involucrados en la definición de los objetivos relacionados con la calidad, las regulaciones gubernamentales, la seguridad y otros parámetros claves de la organización. El SGB guía el trabajo llevado a cabo por el Product Owner, el Scrum Master y el Equipo Scrum.

- Los vendedores, incluyendo a individuos u organizaciones externas, ofrecen productos y/o servicios que no están dentro de las competencias centrales de la organización del proyecto.

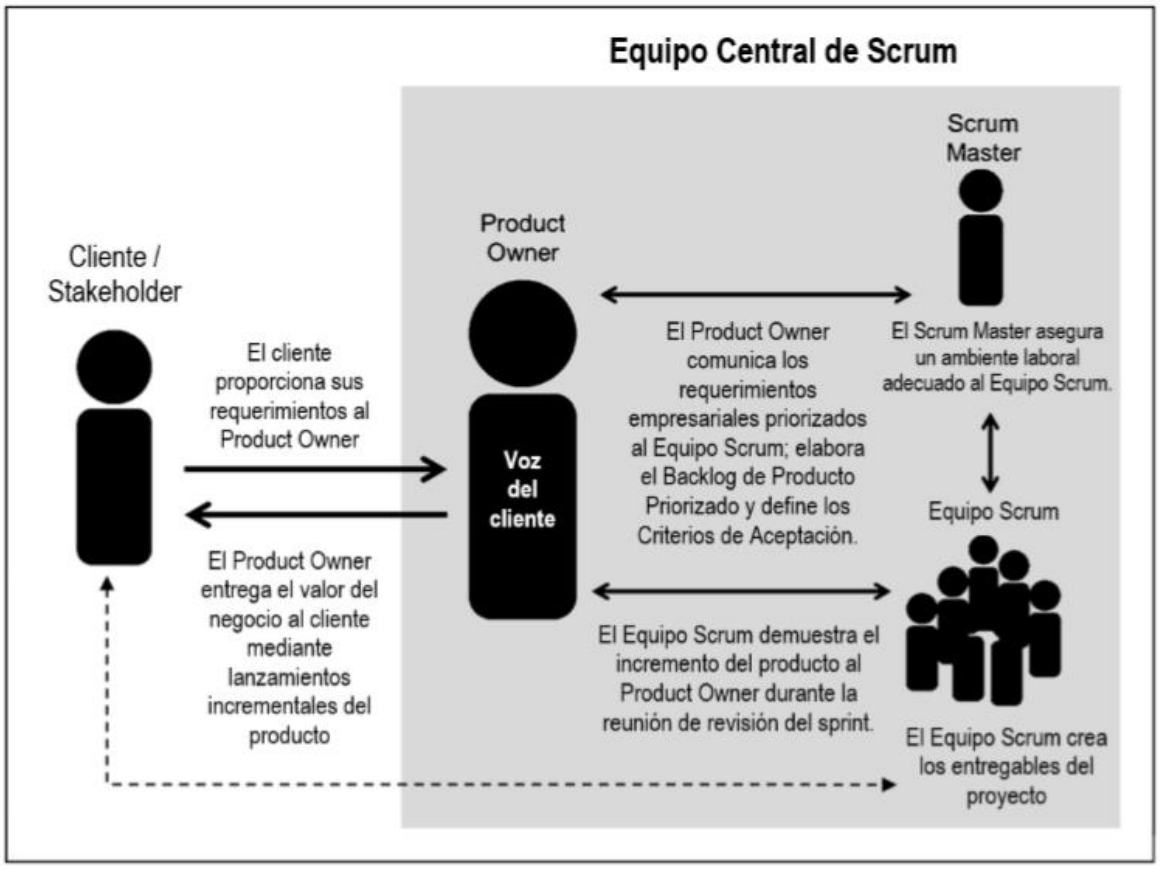

Figura 9. Organización en Scrum (Guía SBOK™, 2016)

El aspecto de organización de Scrum aborda también los requisitos de estructura del equipo para implementar Scrum en grandes proyectos, programas y portafolios. (Guía SBOK™, 2016)

2. **Justificación del negocio:** Es importante que una organización lleve a cabo una evaluación adecuada del negocio antes de iniciar cualquier proyecto. Esto ayuda a los tomadores de decisiones clave a entender la necesidad de cambio en la empresa

o de un nuevo producto o servicio, la justificación para seguir adelante con un proyecto y su viabilidad. La adaptabilidad de Scrum permite que los objetivos y procesos del proyecto cambien si cambia su justificación del negocio. Es importante señalar que, si bien el Product Owner es el responsable principal de la justificación del negocio, otros miembros del equipo también contribuyen considerablemente. (Guía SBOK™, 2016)

3. **Calidad:** En Scrum, la calidad se define como la capacidad con la que cuenta el producto o los entregables para cumplir con los criterios de aceptación y de alcanzar el valor de negocio que el cliente espera. Para garantizar que un proyecto cumpla con los requisitos de calidad, Scrum adopta un enfoque de mejora continua mediante el cual el equipo aprende de sus experiencias y de la participación de los stakeholders para mantener constantemente actualizado el Backlog Priorizado del Producto con cualquier cambio en los requisitos. El Backlog Priorizado del Producto nunca se completa sino hasta el cierre o conclusión del proyecto. Cualquier cambio en los requisitos debe reflejar los cambios en el entorno empresarial, ya sean internos o externos, y permitirle al equipo trabajar continuamente y adaptarse para lograr dichos requerimientos. (Guía SBOK™, 2016)

4. **Cambio:** Cada proyecto, independientemente del método o framework que se utilice, está expuesto a cambios. Es importante que los miembros del equipo del proyecto entiendan que los procesos de desarrollo de Scrum están diseñados para aceptar el cambio. Las organizaciones deben tratar de maximizar los beneficios que se deriven de los cambios y minimizar cualquier impacto negativo a través de procesos de gestión de cambio diligentes, según los principios de Scrum. (Guía SBOK™, 2016)

5. **Riesgo:** El riesgo se define como un evento incierto o serie de eventos que pueden afectar los objetivos de un proyecto y pueden contribuir a su éxito o fracaso. A los riegos que pueden tener un impacto positivo en el proyecto se les conoce como oportunidades, mientras que las amenazas son riesgos que pudieran afectar negativamente al proyecto. La gestión de riesgos debe hacerse de forma preventiva, y es un proceso iterativo que debe comenzar al inicio del proyecto y continuar a lo largo del ciclo de vida del mismo. (Guía SBOK™, 2016)

Procesos de Scrum

Los procesos de Scrum abordan las actividades específicas y el flujo de un proyecto de Scrum. En total hay diecinueve procesos fundamentales de Scrum que aplican a todos los proyectos.

Fase	Procesos fundamentales de Scrum
Inicio	1. Crear la visión del proyecto 2. Identificar al Scrum Master y Stakeholder(s) 3. Formar Equipos Scrum 4. Desarrollar épica(s) 5. Crear el Backlog Priorizado del Producto 6. Realizar la planificación de lanzamiento
Planificación y estimación	7. Crear historias de usuario 8. Estimar historias de usuario 9. Comprometer historias de usuario 10. Identificar tareas 11. Estimar tareas 12. Crear el Sprint Backlog
Implementación	13. Crear entregables 14. Realizar Daily Standup 15. Refinar el Backlog Priorizado del Producto
Revisión y retrospectiva	16. Demostrar y validar el sprint 17. Retrospectiva del sprint
Lanzamiento	18. Enviar entregables 19. Retrospectiva del proyecto

Tabla 1. Los procesos fundamentales de Scrum (Guía SBOK™, 2016)

3.2.4.2. Herramienta Apache JMeter

También se utilizó la herramienta Apache JMeter para diseñar y ejecutar las pruebas de performance. JMeter es una herramienta open source implementada en Java diseñada para probar el comportamiento funcional y medir el rendimiento de web services. Apache JMeter se puede usar para probar el rendimiento tanto de recursos dinámicos como estáticos de

aplicaciones web. Con esta herramienta se puede simular una gran carga en un servidor, grupo de servidores, red u objeto para probar su fortaleza o analizar el rendimiento general bajo diferentes tipos de carga.

3.2.4.3. Lenguaje Java

En la etapa de automatización de pruebas y para la construcción de la herramienta de verificación se utilizó el lenguaje Java. Adicionalmente se utilizaron los siguientes frameworks de este lenguaje:

TestNG

Es un framework basado en JUnit y NUnit que incluye funcionalidades que potencian lo que estas librerías ofrecen. Permite una configuración más flexible y adaptable. El uso de anotaciones y parámetros facilita su implementación. Provee soporte para fuentes externas de datos.

Spring Boot

Es una versión minimalista del conocido framework Spring. Permite tener una aplicación funcional y corriendo con el mínimo esfuerzo de configuración (que es un punto débil del tradicional Spring), pero con la misma versatilidad de utilizar los componentes deseados y las herramientas embebidas del framework.

3.2.4.4. Base de datos Mysql

Para la persistencia a base de datos de los resultados de las pruebas se utilizó MySQL, además este motor de base de datos también fue seleccionado para salvar una parte de la data del sistema. Mysql es la base de datos open source más popular del mundo debido a su alto rendimiento, confiabilidad y facilidad de uso. Se utiliza tanto en pequeños como en grandes proyectos como base de datos integrada y distribuida.

3.2.5. Implementación de pruebas para el Sistema de Análisis de Tráfico de Clientes

3.2.5.1. Planificación de las Pruebas

Durante el Sprint 0, la ingeniera de calidad llevo a cabo la planificación de las pruebas. La planificación recibió como entrada toda la documentación disponible sobre la funcionalidad y el diseño del sistema y las notas tomadas en las reuniones de capacitación. Las pruebas planificadas fueron aprobadas por el Product Owner. Las actividades desarrolladas durante esta etapa fueron las siguientes:

Definir el alcance de las pruebas

El alcance se estableció tomando en cuenta el tiempo estimado de duración del proyecto, la criticidad del mismo y las restricciones de la empresa cliente. Se acordó con el Product Owner que las pruebas de funcionalidad se realizarían contra la API que expone los servicios consumidos por la aplicación web, definiéndose así el objeto de prueba para las pruebas funcionales y los niveles de las pruebas, que fueron de integración y de sistema. Además, por políticas de seguridad, la ingeniera de calidad no tendría acceso al código fuente de ningún componente, lo cual descartaba la posibilidad de realizar pruebas de caja blanca y no se escribieron suficientes pruebas unitarias como para considerar pruebas funcionales a nivel de componentes. En resumen, el alcance las pruebas fue el siguiente:

- Diseño y ejecución de pruebas funcionales de integración y de sistema para todos los servicios de la API.
- Automatización de los casos de prueba funcionales más importantes.
- Diseño y ejecución de pruebas de performance (tiempo de respuesta, carga y estrés) para los servicios principales de la API.

Todo lo no especificado en los puntos anteriores, se considera fuera del alcance.

Para definir cuál o cuáles serían los servicios principales de la API y cuáles casos de prueba serian automatizados se concretaron un par de reuniones con el Product Owner. Esto se detalla un poco más en la sección de Diseño e implementación de casos de prueba no funcionales y Automatización de casos de prueba funcionales respectivamente.

Definir la ejecución de las pruebas

Luego de que cada tarea fuera terminada a nivel de desarrollo, se debían ejecutar todas las pruebas asociadas a la funcionalidad. Además, al finalizar cada sprint se debía ejecutar la prueba de regresión, la cual incluía todas las pruebas funcionales para asegurarse que los cambios realizados durante el sprint actual no hubieran introducido nuevos errores al sistema y/o hubieran afectado la performance del mismo.

Adicionalmente a las pruebas que se ejecutaban que al final de los sprints, se podían ejecutar estas mismas pruebas (pruebas de performance, pruebas de regresión o pruebas de humo) en cualquier momento de los sprints, a solicitud del Product Owner o por recomendación de la ingeniera de calidad. El tipo de ejecución de las pruebas funcionales dependía exclusivamente del estado de los scripts de automatización. Una vez que la automatización fue culminada, esta reemplazó la ejecución manual de los escenarios automatizados.

Definir los recursos

Se definieron tanto los recursos físicos como lógicos. Para la creación de los entornos de pruebas se solicitó que los servidores, bases de datos y todo el software instalado sea lo más similar posible al entorno de producción. También se solicitó accesos de solo-lectura para cada una de las plataformas que serían utilizadas.

Para el diseño y ejecución de las pruebas se utilizó solo software open source, por lo tanto, no hubo gestión de licencias:

- Elaboración de casos de prueba: Confluence
- Pruebas de performance: JMeter
- Elaboración de scripts de automatización: Eclipse, IntelliJ, Git, Java, TestNG y Mysql

Definir las dependencias

Se estableció de que servicios o funcionalidades de otros proyectos se iba a depender:

- Servicio de extracción de datos.
- Servicio de administración de dispositivos de captura de datos.
- Equipo encargado de la interfaz gráfica web.
- Equipos de gestión de base de datos (Mysql, Postgress, MongoDB).

- Equipo de infraestructura.

Se concertaron reuniones para comprometer la colaboración de los responsables de estos equipos.

3.2.5.2. Análisis y Diseño de casos de Prueba Funcionales

Se utilizó como entradas del diseño de los casos de prueba para las funciones principales del sistema la documentación relacionada al proyecto y las reuniones con el Product Owner. Los casos de prueba fueron organizados por funcionalidad (Test Suites) y priorizados por tipo de flujo en 2 niveles. El nivel 1 incluyó todos los flujos básicos definiendo así la prueba de humo y el nivel 2 incluyó ambos flujos dándonos 100% de cobertura para la prueba de regresión. La plantilla para definir los casos de prueba constó de las siguientes columnas:

Nombre de la columna	**Detalle**
Código del suite de prueba	Código único de identificación de un suite de prueba. El código está conformado de 2 partes: la primera es la abreviatura del proyecto y la segunda es una S seguida por un número correlativo de 3 dígitos.
Suite de prueba	Definición del suite de prueba.
Flujo	Contiene el texto “Happy path” or “Sad path” que especifica si el caso de prueba es parte del flujo básico o flujo alternativo. Esta información se utilizó para priorizar y definir las pruebas de humo y regresión.
Código del caso de prueba	Código único de identificación de un caso de prueba. El código tiene 3 partes: la primera es la abreviatura del proyecto, la segunda el número correlativo del suite de 3 dígitos y el tercero está conformado por la letra T seguida de un número correlativo de 3 dígitos.
Caso de prueba	Definición del caso de prueba.
Precondiciones	Precondiciones del caso de prueba. Las condiciones que se requieren para ejecutar el caso de prueba.
Pasos de la prueba	Definición de los pasos de cada caso de prueba.
Resultado esperado	Resultado esperado de cada caso de prueba.

Prioridad	Es un número que representa la prioridad, siendo 1 el más prioritario. Para este proyecto se utilizaron únicamente 2 niveles de prioridad.

Tabla 2. Explicación del contenido de las columnas de la tabla de especificaciones de casos de prueba

En total se escribieron 623 casos de prueba. La especificación de los casos de prueba fue enviada al Product Owner para su aprobación. Una vez obtenida la aprobación, estos se subieron a la nube (Confluence) del cliente.

3.2.5.3. Automatización de Casos de Prueba Funcionales

El primer paso para iniciar la automatización es definir qué casos de prueba se van a automatizar. Se utilizó la siguiente tabla para dicho análisis:

Caso de prueba	Se probará constantemente (2)	Existe riesgo de error humano (1)	No es complejo de automatizar (1)	Es difícil o toma mucho tiempo de realizar manualmente (2)	Es crítico para el negocio (2)	**Total puntos**	Se automatiza?
MA-001-T001	2	1	1	0	2	**6**	Sí
MA-001-T002	0	0	0	2	2	**4**	Sí
MA-001-T003	0	0	1	0	2	**3**	Sí
MA-001-T004	2	1	1	0	0	**4**	Sí
MA-001-T005	2	0	1	0	2	**5**	Sí
MA-001-T006	2	0	1	0	2	**5**	Sí
MA-001-T007	2	1	1	0	2	**6**	Sí

MA-001-T008	2	0	1	0	2	**5**	Sí

Tabla 3. Ejemplo de la tabla de análisis de automatización de casos de prueba

Los criterios de decisión fueron seleccionados por la ingeniera de calidad y los pesos se definieron en consenso con el Product Owner así como el indicador de selección que para este proyecto fue si el total de puntos era mayor o igual a 2, el caso de prueba se debía automatizar.

El análisis arrojó que todos los casos de prueba diseñados en la etapa anterior debían automatizarse. Aunque parece una labor pesada para una sola ingeniera de calidad, en la realidad no fue así debido a que si recordamos la pirámide del costo de la automatización de las pruebas:

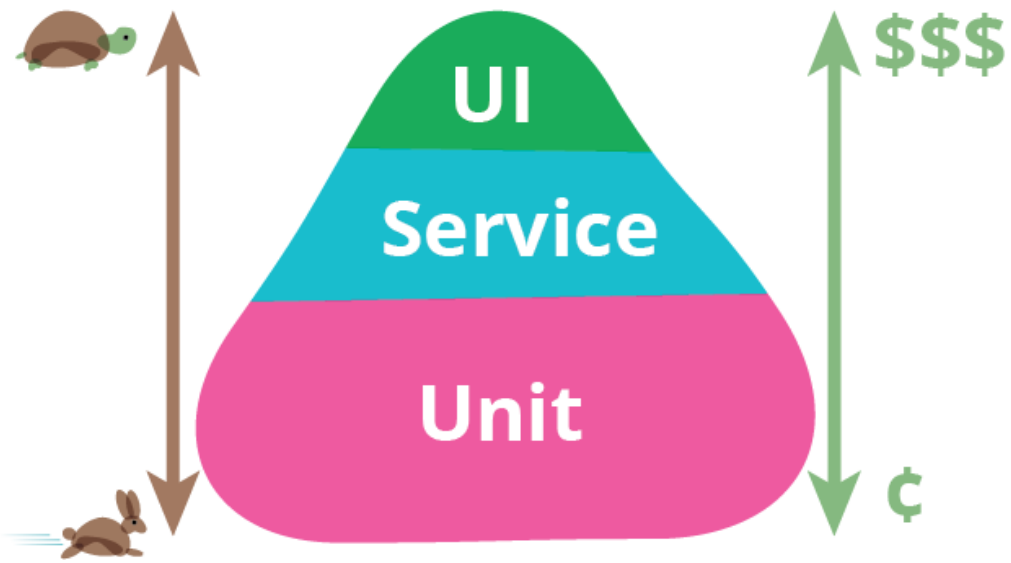

Figura 10. Pirámide del costo de la automatización de las pruebas

Los scripts de automatización de las pruebas de un servicio son más fáciles y rápidos de implementar que los casos de prueba para una IU. Además, son menos costosos y menos volátiles debido a que las capas internas como un servicio o una API tiene menos cambios en el tiempo en comparación con una aplicación IU. Por lo tanto, son pruebas más fáciles de mantener y aportan un mayor beneficio a menor costo.

Se creó las historias de usuario y las tareas para la creación del proyecto de automatización. La tecnología seleccionada para este proyecto fue TestNG (librería para pruebas de software), debido a que brinda una interfaz para la ejecución de pruebas y detección de las fuentes de error más amigable y más fácil de usar que otras librerías. Además, genera reportes por defecto bastante completos que requieren un mínimo esfuerzo de personalización. El proyecto de automatización tomo 2 sprints en finalizarse. Se realizó en paralelo a las tareas de prueba propias de cada sprint.

El framework de automatización de pruebas funcionales

Se construyó una aplicación en Java y TestNG para escribir y ejecutar los scripts automatizados de las pruebas funcionales. Se utilizó Maven para la administración de dependencias, Mysql para la persistencia de los resultados de las ejecuciones y Git para la administración de la configuración.

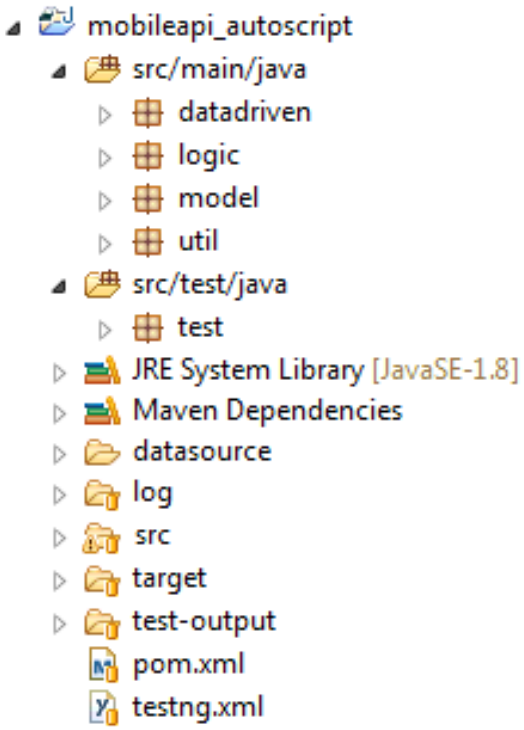

Figura 11. Estructura de paquetes del framework de automatización

El framework de automatización constaba de 5 paquetes:

- **Datadriven:** Este paquete contenía 2 clases, una para la administración de las hojas de Excel que contenían la data de prueba. Y la otra para la conexión y la persistencia a base de datos.

- **Logic:** Implementaba la conexión a la API, las peticiones y las respuestas y toda la lógica de los pasos de las pruebas.
- **Model:** Incluía todos los objetos que se utiliza el paquete "logic".
- **Util:** Contenía clases con métodos de apoyo para el desarrollo, como métodos de parseo, constantes globales, etc.
- **Test:** Incluía una clase por cada escenario de prueba y llamaba a los pasos de prueba que fueron implementados en el paquete "logic". Contenían las validaciones y luego de ejecutarse guardaban los resultados en una base de datos Mysql.

Este framework es reutilizable para otros proyectos de pruebas de backend. Para poder personalizarlo solo se debe cambiar los casos de prueba que están en la carpeta de "datasource" y modificar los pasos de las pruebas en el paquete "test" y en el paquete "logic".

3.2.5.4. Diseño e Implementación de Casos de Prueba No Funcionales

Un aspecto importante a considerar en este proyecto fue la performance del sistema. Se planificaron 3 tipos de pruebas no funcionales para medir el rendimiento del sistema en condiciones normales, con un bajo y un alto volumen de usuarios y también para identificar el límite máximo de usuarios concurrentes a los que puede responder de forma correcta. Por lo tanto, se creó un proyecto en JMeter para implementar estas 3 pruebas de performance. Las pruebas que se ejecutaron fueron:

- Pruebas de tiempo de respuesta
- Pruebas de carga
- Pruebas de stress

Estas pruebas se realizaron luego de que se implementara cada tarea que impactara en la performance del sistema y también cuando la ingeniera de calidad lo consideraba necesario, aunque no estuvieran ligadas a alguna tarea implementada. Además, se realizó una ejecución inicial de estas pruebas para utilizar sus resultados como índices de referencia para pruebas posteriores y así determinar el grado de mejora o perdida de la performance del sistema. Adicionalmente, se realizó una ejecución final para determinar la performance final del sistema luego de todos los cambios.

La ejecución inicial se realizó sobre una versión sin cambios del sistema, desplegada en el entorno de pruebas. Esta primera ejecución incluyo pruebas de rendimiento y estrés a los principales endpoints de la API. La identificación de los principales endpoints fue con el apoyo y en acuerdo del Product Owner.

Pruebas de tiempo de respuesta

Los escenarios para esta prueba se definieron recreando 3 momentos del día en los que los usuarios ingresan al sistema con una frecuencia baja, media y alta.

- Frecuencia baja de usuarios (de 7am hasta las 8am)
- Frecuencia media de usuarios (de 3pm hasta las 4pm)
- Frecuencia alta de usuarios (de 9am hasta las 10am)

Como resultado de la ejecución inicial se obtuvo la siguiente información:

Escenario de pruebas	Tiempo promedio de respuesta esperada
Baja concurrencia de usuarios (100 usuarios por minuto)	1,12 segundos
Media concurrencia de usuarios (200 usuarios por minuto)	1,94 segundos
Alta concurrencia de usuarios (400 usuarios por minuto)	2,73 segundos

Tabla 4. Resultado de la prueba de tiempo de respuesta inicial

Estos resultados se compararon con los resultados de las pruebas posteriores para determinar si el rendimiento de sistema se estaba viendo afectado por los cambios. Al finalizar el proyecto, los resultados fueron:

Escenario de pruebas	Tiempo promedio de respuesta esperada
Baja concurrencia de usuarios (100 usuarios por minuto)	1,03 segundos
Media concurrencia de usuarios (200 usuarios por minuto)	1,61 segundos
Alta concurrencia de usuarios (400 usuarios por minuto)	2,24 segundos

Tabla 5. Resultado de la prueba de tiempo de respuesta final

En conclusión, se logró mejorar el rendimiento del sistema con la implementación de los cambios.

Pruebas de carga

Estas pruebas se realizaron basadas en los mismos escenarios de las pruebas de rendimiento, como se puede sobreentender de los resultados de las pruebas de rendimiento, estas pruebas fueron siempre exitosas tanto al iniciar el proyecto como al terminarlo:

Escenario de pruebas	Porcentaje de usuarios que recibieron la respuesta esperada
Baja concurrencia de usuarios (100 usuarios por minuto)	100%
Media concurrencia de usuarios (200 usuarios por minuto)	100%
Alta concurrencia de usuarios (400 usuarios por minuto)	100%

Tabla 6. Resultado de la prueba de carga

Pruebas de estrés

La prueba de estrés considero como punto de partida el escenario exitoso de mayor volumen de usuarios en la prueba de carga. A partir de ese punto, se fue incrementando en 100 la carga de usuarios hasta encontrar errores o respuestas diferentes a las esperadas. Luego de que aparecieran los primeros errores, se redujo la carga a 50 para acotar el resultado de la prueba. Los resultados fueron:

Escenario de pruebas	Porcentaje de usuarios que recibieron la respuesta esperada
400 usuarios por minuto(Alta concurrencia de usuarios)	100%
500 usuarios por minuto	100%
550 usuarios por minuto	97,45%
600 usuarios por minuto	88,33%

Tabla 7. Resultado de la prueba de estrés inicial

Por lo tanto, inicialmente el sistema podía funcionar correctamente con 500 usuarios por minuto. Al finalizar el proyecto estos resultados mejoraron como se puede observar en el siguiente cuadro:

Escenario de pruebas	Porcentaje de usuarios que recibieron la respuesta esperada
400 usuarios por minuto(Alta concurrencia de usuarios)	100%
500 usuarios por minuto	100%
550 usuarios por minuto	100%
600 usuarios por minuto	96,83%

Tabla 8. Resultado de la prueba de estrés final

3.2.5.5. Implementación de Herramientas de Verificación

A mediados del proyecto, comenzaron a surgir necesidades de verificación muy específicas que no eran cubiertas por ninguna de las pruebas planificadas. Inicialmente se pensó en crear scripts independientes para cada necesidad, que se ejecutaran a medida que se requieran. Finalmente, se acordó en construir estos scripts de una forma más escalable, creándolos como servicios web que luego puedan ser consumidos por una aplicación web. De esta forma se facilita el adicionar scripts de verificación para necesidades que surjan en el futuro. Esta aplicación se convertiría en una herramienta de pruebas a medida para el sistema de la organización.

La necesidad de verificación fue causada porque el Product Owner quería conocer en cualquier momento del día si la extracción de la data se estaba realizando correctamente para asegurarse que la data que procesaba la API era adecuada. De esta forma, si se detectaba una anomalía durante el proceso de extracción de la data, el Product Owner o la persona a cargo del monitoreo podía ordenar un reprocesamiento antes de que la data se muestre en el sistema principal y sea visible para el usuario final. La planificación y el desarrollo de este web service para la verificación de integridad de datos se realizaron en el transcurso del sexto sprint.

3.3. Evaluación

3.3.1 Evaluación Técnica

Con la implementación y ejecución de estas pruebas periódicamente durante el proceso de desarrollo se pudo garantizar la calidad esperada de la API luego de la implementación de los cambios. Las funcionalidades del sistema no se vieron alteradas por los cambios y la performance del mismo mejoro en aspectos de tiempo de respuesta y en la tolerancia al número de usuarios conectados simultáneamente.

Además, se aportó herramientas de pruebas como los scripts de pruebas funcionales automatizadas y el web service de verificación de integridad de datos. Estas herramientas aportaron valor a la organización facilitando el proceso de control de calidad del sistema en el desarrollo de este proyecto, así como también lo harán en cualquier otra implementación de cambios futura. El web service de verificación de integridad de datos es el primero de los muchos web services que se implementarán para validar otros aspectos importantes del sistema creando así el backend de una aplicación orientada específicamente al control de calidad. Al terminar este proyecto, se inició la planificación para el desarrollo de esta aplicación que se llamará Herramienta de Administración Interna.

CAPÍTULO IV – REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

4.1. Contribución y Responsabilidades

El rol de la autora del presente informe fue el de ingeniera de calidad a cargo de todas las actividades necesarias para garantizar la calidad de la API del Sistema de Análisis de Tráfico de Clientes durante el proceso de implementación de cambios para mejorar su mantenibilidad y performance siguiendo las buenas prácticas de la metodología Scrum.

El equipo de trabajo estuvo conformado por un Scrum Master, dos desarrolladores y una ingeniera de calidad. La ingeniera de calidad se encargó de la planificación, diseño, implementación y ejecución de las pruebas de software a lo largo del proceso de desarrollo tanto manuales como automatizadas. Además, se encargó de diseñar e implementar un web service para la verificación de la integridad de datos, el cual fue un requisito específico del proyecto.

Este trabajo fue una experiencia enriquecedora para mi carrera profesional debido a que me dio la oportunidad de asumir mayores responsabilidades al estar enteramente a cargo de la calidad de un proyecto. Además, me permitió aprender y trabajar con nuevas tecnologías y fortalecer mis conocimientos en la metodología Scrum.

4.2. Lecciones Aprendidas

- La comunicación entre los miembros del equipo fue un aspecto muy importante para el éxito de este proyecto. Los malos entendidos retrasan el desarrollo. Es vital cuando se trabaja con un equipo multinacional asegurarse de que la información fue correctamente recibida y entendida. Muchas veces la comunicación oral no es suficiente y debe ser complementada con comunicación escrita.

- En ámbitos técnicos, fue importante considerar en las estimaciones de las tareas el tiempo de investigación que requerían algunas de ellas.

- Las pruebas no funcionales son tan importantes como las pruebas funcionales y deben evaluarse de distinta forma, pero con la misma rigurosidad.

- Cuando se estima que un proyecto tendrá una duración corta es mejor utilizar herramientas que el equipo conozca o que hayan utilizado antes, en lugar de probar herramientas nuevas con grandes curvas de aprendizaje.

- La automatización agiliza y optimiza las tareas calidad además de reducir el riesgo de error humano. Sin embargo, su implementación es sumamente costosa. Por lo tanto, siempre debe ser aprobada y apoyada por el Product Owner. El Product Owner debe ser consciente de las ventajas que provee, pero también el tiempo y los recursos que consume.

- Se debe buscar aprovechar las herramientas que se tienen disponibles en un proyecto para su beneficio. Por ejemplo, durante este proyecto se pudo haber utilizado el TestRail de Jira para la gestión de casos de prueba y para manejar la trazabilidad de los mismos con las historias de usuario y tareas en lugar de utilizar Confluence que es un repositorio genérico más no, una herramienta especializada en la gestión de casos de prueba.

- Para la efectiva medición de la performance se debe contar con indicadores que permitan analizarla por comparación o contraste. Si no se cuenta con estos indicadores inicialmente (como el tiempo promedio de respuesta) las pruebas de performance se deben ejecutar antes de implementar los cambios con el objetivo de usar los resultados de estas pruebas como indicadores iniciales.

CAPÍTULO V – CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

Se cumplió con el objetivo principal de este proyecto, el cuál fue garantizar la calidad de la API del Sistema de Análisis de Trafico de Clientes luego de la implementación de los cambios planificados. Las actividades realizadas como parte de las pruebas de software de este proyecto permitieron asegurar la funcionalidad y la performance del sistema.

- Se diseñaron y se ejecutaron las pruebas funcionales.
- Se diseñaron e implantaron y ejecutaron las pruebas automatizadas.
- Se diseñaron y ejecutaron las pruebas no funcionales.
- Se reportaron el resultado de las pruebas.
- Se diseñó, implemento y ejecuto una herramienta de verificación a medida.

Adicionalmente, en el ámbito de la performance, no sólo se garantizó que no se perdiera rendimiento con los cambios, sino que también se logró mejorar el tiempo de respuesta y la tolerancia a usuarios concurrentes como muestran las siguientes tablas de resultados:

Escenario de prueba	**Tiempo promedio de respuesta antes de lo cambios**	**Tiempo promedio de respuesta luego de los cambios**
Baja concurrencia de usuarios (100 usuarios por minuto)	1,12 segundos	1,03 segundos
Media concurrencia de usuarios (200 usuarios por minuto)	1,94 segundos	1,61 segundos
Alta concurrencia de usuarios (400 usuarios por minuto)	2,73 segundos	2,24 segundos

Tabla 9. Cuadro comparativo de los resultados de las pruebas de rendimiento antes y después de los cambios.

Escenario de prueba	**Porcentaje de usuarios que recibieron la respuesta antes de los cambios**	**Porcentaje de usuarios que recibieron la respuesta luego de los cambios**
400 usuarios por minuto (Alta concurrencia de usuarios)	100%	100%
500 usuarios por minuto	100%	100%
550 usuarios por minuto	97,45%	100%
600 usuarios por minuto	88,33%	96,83%

Tabla 10. Cuadro comparativo de los resultados de las pruebas de estrés antes y después de los cambios.

5.2. Recomendaciones

Mis recomendaciones para trabajos futuros son:

- Utilizar una herramienta de gestión de casos de prueba. Las herramientas de gestión de casos de prueba facilitan su escritura y su administración al ser específicos para esta labor. Favorecen a la organización en equipos dónde hay varios ingenieros de calidad trabajando en conjunto y/o una gran cantidad de casos de prueba.

- Implementar integración continua. Utilizando un servidor de integración continua se puede incluir la ejecución de las pruebas de regresión automatizadas en cada build. Con el objetivo de que se ejecuten luego de cada cambio desplegado en el servidor ya sea de pruebas, staging o producción. De esta manera se reducirá el tiempo que los ingenieros de calidad invierten en las pruebas de regresión.

REFERENCIAS BIBLIOGRÁFICAS

Estándares

1. IEEE 610.12:1990, Standard Glossary of Software Engineering Terminology.
2. UNE-EN ISO 8402:1995, Gestión de la calidad y aseguramiento de la calidad. Vocabulario.
3. ISO 9000:2015, Sistemas de Gestión de la Calidad. Fundamentos y vocabulario.
4. ISO 9001:2015, Sistemas de Gestión de la Calidad Requisitos.
5. ISO 9004:2009, Sistemas de Gestión de la Calidad. Directrices para la mejora del desempeño.
6. ISO 9126:2015, Standard Software Quality.

Libros

1. Una guía para el Cuerpo de Conocimiento de Scrum (Guía SBOK™) – 3ra Edición, 2016
2. International Software Testing Qualifications Board Glossary. Recuperado de: http://glossary.istqb.org/

GLOSARIO

- **Ambiente de pruebas:** Es el ambiente asignado para realizar las pruebas del sistema. Este entorno debe ser lo más parecido posible al ambiente de producción en software e infraestructura.

- **Análisis de datos:** El análisis de los datos puede ayudar a determinar la causa de los problemas existentes o potenciales y por lo tanto guiar las decisiones acerca de las acciones correctivas y preventivas necesarias para la mejora. (ISO 9004, 2009)

- **API:** Acrónimo de Application Programming Interface (Interface de programación de aplicaciones).

- **Aseguramiento de la calidad:** el conjunto de acciones planificadas y sistemáticas implantadas dentro del sistema de calidad, y demostrables si es necesario, para proporcionar la confianza adecuada de que una entidad cumplirá los requisitos para la calidad. (ISO 8402, 1995)

- **Base de datos:** Es un conjunto de datos relacionados que corresponden a un mismo contexto y se encuentran almacenados computacionalmente para su tratamiento y administración posterior.

- **Calidad:** Conjunto de propiedades y características de un producto o servicio, que le confieren aptitud para satisfacer unas necesidades explícitas o implícitas. (ISO 8402, 1995)

- **Calidad de software:** La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario. (IEEE, Std. 610-1990)

- **Caso de prueba:** es un conjunto de acciones con resultados y salidas previstas basadas en los requisitos de especificación del sistema. Se compone de entradas, precondiciones para la ejecución y salidas esperadas desarrolladas con el objetivo de testear un aspecto concreto del software (ejecutar un camino del programa en

particular, verificar la conformidad de un requisito concreto, detectar tipos de errores específicos).

- **Cliente:** persona u organización que podría recibir o que recibe un producto o un servicio destinado a esa persona u organización o requerido por ella. (ISO 9000, 2015)

- **CMMI:** Capability Maturity Model Integration. Modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Fue desarrollado por el Instituto de Ingeniería del Software de la Universidad Carnegie Mellon (SEI), y publicado en su primera versión en enero de 2002.

- **Criterio de aceptación:** Son los criterios de salida que un componente o sistema debe cumplir para ser aceptado por un usuario, cliente u otra entidad autorizada. (IEEE 610, 1990)

- **Code Review:** Su traducción directa es revisión de código. Se refiere a la actividad de revisión que realiza un desarrollador asignado sobre el código de otro desarrollador. El propósito de esta actividad es encontrar defectos en una etapa temprana, problemas con estándares del leguaje o del proyecto, duplicidad de código, sugerencias de optimización, etc.

- **Código legacy:** O también llamado "código heredado" es un término que se utiliza para llamar a aplicaciones antiguas o que se encuentran en un lenguaje obsoleto.

- **Control de calidad:** el conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requerimientos relativos a la calidad del producto o servicio. (ISO 8402, 1995)

- **Defecto:** Es el error ubicado en el código fuente del sistema.

- **Equipo Scrum:** El Equipo Scrum es un grupo o equipo de personas responsables de entender los requerimientos del negocio especificados por el Product Owner, de

estimar las historias de usuarios y de la creación final de los entregables del proyecto. (Guía SBOK™, 2016)

- **Error:** Una acción humana que puede producir resultados incorrectos.

- **Fallo:** Es la manifestación del defecto en el sistema.

- **Flujo básico de prueba:** O también llamado "Happy Path". Agrupa a todos los casos de prueba que incluyen la lógica principal de la funcionalidad en prueba. Son escenarios que se centran en la situación ideal dónde todos los pasos de la prueba son ejecutados exitosamente y se reciben los resultados esperados.

- **Flujo alternativo de prueba:** O también llamado "Sad Path". Agrupa todos los flujos alternativos y distintos del flujo básico. Todas las posibles combinaciones a partir de un paso de prueba que no retorno un resultado esperado.

- **Framework:** Es un conjunto de plantillas o librerías que tienen como propósito reducir el esfuerzo de implementación. Contiene elementos básicos y comunes de diseño e implementación para un determinado fin.

- **Funcionalidad:** Grado en que las necesidades asumidas o descritas se satisfacen. Se divide en las subcaracteríticas idoneidad, precisión, interoperabilidad, seguridad. (ISO 9126, 2015)

- **Historia de Usuario:** Generalmente las escribe el Product Owner y están diseñadas para asegurar que los requisitos del cliente estén claramente representados y puedan ser plenamente comprendidos por todos los stakeholders. (Guía SBOK™, 2016)

- **Integración continua:** es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua requiere un componente de automatización (como un software para integración continua o servicio de versiones) y un componente cultural de aprender a integrar los cambios con frecuencia para que su implementación sea exitosa. Los

objetivos clave de la integración continua consisten en encontrar y resolver errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas versiones de software.

- **Issue tracker:** Traducido al español como rastreador de problemas, es una herramienta que permite dar seguimiento a las tareas que se realizan durante los ciclos de desarrollo. Actualmente existen issue trackers específicos por metodología los cuales permiten una mejor visualización del avance realizado por el equipo y fluyen junto con las iteraciones de acuerdo a la metodología de desarrollo.

- **Lenguaje de programación:** es un lenguaje formal con una sintaxis especifica que permite escribir instrucciones determinadas para que sean ejecutadas por el computador. Los lenguajes de programación son distintos de acuerdo con la plataforma en la que se quiera ejecutar las instrucciones.

- **Manifiesto ágil:** Una declaración sobre los valores que sustentan el desarrollo ágil de software. Los valores son: individuos e interacciones sobre procesos y herramientas, software funcionando sobre documentación integral, colaboración del cliente sobre negociación de contrato, respuesta al cambio sobre seguir un plan.

- **Refactorización:** Alterar la estructura interna del código fuente de un proyecto sin cambiar su comportamiento externo con el objetivo de mejorar la facilidad de comprensión del código, eliminar código innecesario y facilitar el mantenimiento en el futuro. La refactorización no arregla errores ni incorpora funcionalidades.

- **Open source:** Código abierto traducido al español, se refiere a aquellos productos software que no requieren licencias y/o pagos para su utilización.

- **Pasos de prueba:** Son el conjunto de instrucciones que conforman un caso de prueba. Cada paso de prueba puede tener un resultado esperado.

- **Pizarra de Scrum:** Es un radiador de información que muestra el progreso del Equipo Scrum en completar las tareas del sprint actual. (Guía SBOK™, 2016)

- **Product Owner:** Es la persona responsable de maximizar el valor del negocio para el proyecto. Este rol es responsable de articular los requisitos del cliente y de mantener la justificación del negocio del proyecto. El Product Owner representa la voz del cliente. (Guía SBOK™, 2016)

- **Producto:** Un producto es el resultado de un proceso dado en la organización. Pueden ser productos físicos, servicios o programas informáticos. (ISO 9001, 2015)

- **Pruebas automatizadas:** Son los escenarios de prueba que han sido implementados en algún lenguaje de programación o a través de alguna herramienta de automatización y se pueden ejecutar automáticamente.

- **Pruebas de caja blanca:** Son las pruebas realizadas al código fuente de la aplicación sin ejecutar el sistema. Para realizar estas pruebas es necesario tener acceso al código del aplicativo.

- **Pruebas de caja negra:** Son las pruebas realizadas al sistema en ejecución. Para realizar estas pruebas no es necesario tener acceso o conocer el código fuente del aplicativo.

- **Pruebas de carga:** La carga de trabajo se refiere a la capacidad máxima que tiene un servidor web (hardware y software), para atender a un conjunto de usuarios de manera simultánea. Por ello, las actividades de esta etapa tienen relación con comprobar, de manera anticipada, el funcionamiento que tendrá el servidor del Sitio Web cuando esté en plena operación. Las pruebas en este caso consisten en simular una carga de trabajo similar y superior a la que tendrá cuando el sitio esté funcionando, con el fin de detectar si el software instalado (programas y aplicaciones) cumple con los requerimientos de muchos usuarios simultáneos y también si el hardware (servidor y el equipamiento computacional de redes y enlace que lo conecta a Internet) es capaz de soportar la cantidad de visitas esperadas.

- **Pruebas de estrés:** Estas pruebas consisten en estresar un servidor con cargas mayores a las esperadas hasta dejar de recibir respuestas positivas. El principal

objetivo de esta prueba es determinar el límite superior de tolerancia del servidor en el que puede funcionar con normalidad devolviendo las respuestas esperadas.

- **Pruebas de humo:** Estas pruebas están enfocadas en probar las funcionalidades más críticas e importantes de un sistema, modulo o componente. Normalmente incluyen los casos de prueba que contienen los happy paths o flujos básicos.

- **Pruebas de integración:** Son las pruebas que comprueban la interrelación entre los componentes. Es el segundo nivel de las pruebas verificando la funcionalidad conjunta de diversos componentes.

- **Pruebas de regresión:** El objetivo de esta prueba es determinar que no se hayan introducido nuevos errores alrededor de una funcionalidad que acaba de sufrir un cambio.

- **Pruebas de rendimiento:** Proceso de pruebas para determinar el rendimiento de un producto software (tiempo de respuesta, operaciones/hora, etc.).

- **Pruebas funcionales:** Son las pruebas relacionadas con validar y verificar solo la funcionalidad de un sistema.

- **Pruebas no funcionales:** Son todas las pruebas que no verifican la funcionalidad de sistema. Están relacionadas a los atributos de calidad como portabilidad, mantenibilidad, eficiencia, confiabilidad y escalabilidad.

- **Pruebas unitarias:** Testeo de unidades o componentes de software individualmente. Normalmente estas pruebas son codificadas y ejecutadas por los mismos desarrolladores. Es el nivel de pruebas más bajo y más atómico.

- **Resultado esperado:** Es el resultado que se espera recibir luego de la ejecución de una serie de pasos de prueba. Si los resultados esperados coinciden con los resultados obtenidos durante la ejecución de las pruebas, se considera que la prueba o el caso

de prueba fue exitoso. De lo contrario, la prueba falla y se reporta el defecto al equipo de desarrollo.

- **Repositorio:** Es un lugar lógico de almacenamiento de software. El repositorio puede ser creado localmente o se puede utilizar servicios en la nube.

- **Scrum:** Es uno de los métodos ágiles más populares. Es un framework adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer un valor considerable en forma rápida a lo largo del proyecto. Scrum garantiza transparencia en la comunicación y crea un ambiente de responsabilidad colectiva y de progreso continuo. El framework de Scrum, tal como se define en la Guía SBOK™, está estructurado de tal manera que es compatible con el desarrollo de productos y servicios en todo tipo de industrias y en cualquier tipo de proyecto, independientemente de su complejidad. (Guía SBOK™, 2016)

- **Scrum Master:** Es un facilitador que asegura que el Equipo Scrum esté dotado de un ambiente propicio para completar con éxito el desarrollo del producto. El Scrum Master guía, facilita e enseña las prácticas de Scrum a todos los participantes en el proyecto, elimina los impedimentos que enfrenta el equipo y se asegura de que se estén siguiendo los procesos de Scrum.

  Debe tenerse en cuenta que el rol de Scrum Master es muy diferente a la función que desempeña el Project Manager en un modelo tradicional de cascada en la gestión de proyectos, en el que el Project Manager trabaja como gerente o líder del mismo. El Scrum Master sólo trabaja como un facilitador y está en el mismo nivel jerárquico que cualquier otra persona en el Equipo Scrum— cualquier persona del Equipo Scrum que aprenda a facilitar proyectos Scrum puede convertirse en el Scrum Master de un proyecto o sprint. (Guía SBOK™, 2016)

- **Sprint:** Son ciclos de trabajo cortos y concentrados. (Guía SBOK™, 2016)

- **Stakeholder:** Cualquier persona interesada en, afectada por y/o implicada con el funcionamiento del sistema software. Por ejemplo, el usuario, el cliente, nuestra empresa, etc.

- **Outsourcing:** Subcontrata a terceros o empresas externas encargadas de loa procesos relacionados con el área de tecnologías de la información que realiza la organización.

- **Validación:** Comprobación de que se está construyendo el producto correcto.

- **Verificación:** Comprobación de que se está construyendo el producto correctamente.