ESCUELA TÉCNICA SUPERIOR

DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería del Software

Curso Académico 2022/2023

Trabajo Fin de Grado

ESTUDIO DE LA CALIDAD DEL SOFTWARE EN UNA EMPRESA: AUTOMATIZACIÓN DE PRUEBAS

Autor: Miguel Ruiz Morán

Directores: Gema Gutiérrez Peña

Agradecimientos

A mis padres, hermanos y toda mi familia la cual me ha enseñado a solucionar las peores dificultades de la mejor manera posible.

A mis amigos del colegio, de la universidad y en definitiva a toda la gente que me ha acompañado y apoyado durante los mejores y peores momentos de este camino.

Y a mi tutora por ayudarme a realizar este trabajo de la manera más cómoda y llevadera posible.

Resumen

La calidad del software se ha convertido en un factor crítico en el entorno empresarial actual, donde la tecnología juega un papel fundamental en el desarrollo y éxito de las organizaciones. En un mundo cada vez más digitalizado, los clientes demandan soluciones confiables y eficientes que cumplen con sus expectativas y necesidades. Por lo tanto, asegurar la calidad del software se ha convertido en un imperativo para las empresas que desean mantener una ventaja competitiva en el mercado.

Este Trabajo de Fin de Grado surge con la motivación por parte del alumno y la empresa de realizar un estudio sobre los avances realizados tras la implantación de un Plan de Calidad hace aproximadamente tres años. Este estudio servirá para evaluar de una mejor manera el progreso obtenido y los beneficios en metodologías de trabajo y gestión usadas durante este tiempo y ya asentadas en la compañía.

A esto se le suma la posibilidad de poder comenzar con la automatización de pruebas como parte del Plan de Calidad y que supone un nuevo avance en pro de la calidad de los productos ofrecidos. La DEMO que se presenta en este trabajo tiene como objetivo ofrecer la futura estructura que tendrá el futuro proyecto de pruebas en todos los equipos de la empresa, así como ofrecer una visión de lo que se es capaz de lograr con las herramientas con las que contamos en la actualidad.

En conclusión, este TFG proyectará una visión actual del campo de la calidad del software a través del estudio comentado sobre la empresa, reconociendo su importancia estratégica en el entorno empresarial actual. Y a través también del paso a la acción de uno de los objetivos del Plan de Calidad implantado en dicha compañía, y que supondrá un aumento en la eficiencia, confiabilidad y eficacia, pilares que se buscan al aceptar la calidad en el software.

Contenido

Índice de figuras

1 Introducción

1.1 Motivación del proyecto

La motivación de este proyecto viene infundida tras el comienzo de mis prácticas externas en la empresa sobre la cual se realizará el estudio en este TFG.

Me incorporé a esta empresa con el objetivo de aprender todos los aspectos relacionados con la calidad del software haciendo énfasis en el proceso de evolución que estaba realizando, y en el que sigue trabajando, la empresa con la implantación de calidad. Este es un tema que desde hace ya más de un año viene siendo de mi agrado tras descubrir el mundo de la calidad en alguna de las asignaturas cursadas durante la carrera. Me motiva el hecho de pensar la magnitud de mejora que puede sufrir una empresa tras la implantación de prácticas, herramientas y otros cambios dirigidos a la calidad.

Adicionalmente, tras un mes en la empresa surge la posibilidad de empezar un proyecto de automatización de pruebas como parte de todas estas medidas dirigidas a mejorar el control de la calidad en la empresa. Es ahí cuando decido que algo que va a ser fruto de mi dedicación como es sentar las bases de dicho proyecto puede ser el comienzo de mi carrera dentro de la calidad y surge la idea de incluir dicha experiencia en mi trabajo de fin de grado.

También es interesante comentar la motivación por parte de la empresa de promover el proyecto tratado, ya que, para esta, surge una oportunidad de realizar un estudio que analice a grandes rasgos la evolución sufrida en la empresa de manera no tan detallada y con una visión más externa. De igual manera, la propuesta de realizar este proyecto lanza una nueva oportunidad para la compañía de iniciar un proyecto que hasta ese momento no tenía una fecha de comienzo establecida y se encontraba en estado de espera. Por todo esto, este proyecto significa un gran beneficio para calidad que engloba a la empresa en todo su conjunto.

1.2 Objetivos del proyecto

Los objetivos del proyecto presentado en este documento son los siguientes:

- Estudiar la decisión y estrategia de implantación de un Plan de Calidad en una empresa.
- Realizar un seguimiento de la evolución que ha tenido un Plan de Calidad en una empresa.

- Presentación la primera DEMO de un proyecto de automatización de pruebas para una empresa.

Se pretende presentar un proyecto que trate por completo la implantación de un Plan de Calidad en una empresa, desde el estudio previo de este para conocer la situación inicial en cuanto a la calidad, hasta la evolución y paso por cada fase de la implantación, así como los pasos futuros a realizar, dado que en el ejemplo tratado en este proyecto no se ha completado dicha implantación.

En cuanto al proyecto de automatización de pruebas, se trata de un proyecto comenzado en enero de 2023 y el cual pertenece a una de las propuestas encontradas en la evaluación del Plan de Calidad. Se trata de un proyecto que, debido a su corta edad, aún está en fases tempranas de desarrollo, pero será presentada su implantación, las bases de dicho proyecto, una DEMO y el alcance, así como futuras evoluciones que sufrirá.

1.3 Estructura del documento

En este apartado se explicará la estrategia principal que se ha seguido durante la elaboración de este proyecto para estructurar los contenidos del mismo.

Tras el capítulo 1, la Introducción, en el cual encontraremos tanto apartados dedicados al comienzo de este proyecto como son la motivación y los objetivos de este, como el apartado en el que nos encontramos de estructura del documento y un apartado dedicado a explicar la metodología y herramientas usadas para el desarrollo del trabajo.

En el capítulo 2, estado del arte, encontramos toda la información necesaria para entender el contexto en el que se desarrolla todo el proyecto, explicando desde qué es la calidad del software hasta entender su utilidad en la actualidad haciendo uso del *testing* y metodologías ágiles.

En el capítulo 3, estudio de la calidad del software, podemos encontrar todo el estudio realizado sobre la empresa en la que se desarrolla todo el proyecto, su situación inicial, las decisiones tomadas en un inicio y su evolución con el paso del tiempo tras los sucesivos avances logrados.

En el capítulo 4, implantación de la automatización de pruebas, se nos presentará el proyecto mencionado durante todo el TFG, explicando su estructura y herramientas, así como todas las tecnologías usadas para desarrollar la DEMO de la cual se expone su ejecución.

Por último, en el capítulo 5, conclusiones y trabajos futuros, encontraremos un breve resumen de la transformación sufrida en la empresa tras la implantación del Plan de Calidad estudiando cada uno de sus beneficios y pasos a seguir a partir del estado actual. También se mencionarán todas las nuevas funcionalidades y herramientas que provocarán la evolución en el proyecto de automatización de pruebas.

1.4 Metodología del trabajo del TFG

Durante la realización de este proyecto se ha utilizado como herramienta principal de trabajo y organización la plataforma en línea llamada Trello. Esta es una herramienta para la gestión de proyectos y tareas que proporciona una interfaz basada en tableros, listas y tarjetas, lo que facilita el seguimiento del proyecto. Además, permite la colaboración en tiempo real, lo que significa que los miembros del equipo pueden trabajar juntos en un proyecto y mantenerse al tanto de los cambios realizados.

Al comienzo de este trabajo, se creó un tablero en Trello compartido junto con la tutora para la organización de este. Aunque al principio se trataba de una plantilla básica y generalizada para un trabajo fin de grado, se diseñó para el presente proyecto con la estructura que se puede observar en la siguiente imagen, basándose en un híbrido entre los marcos de gestión de proyectos más usados que son Scrum y Kanban. El primero ayuda a estructurar y gestionar el trabajo mediante un conjunto de valores, principios y prácticas, y el segundo, se basa en tareas visuales para gestionar los flujos de trabajo.

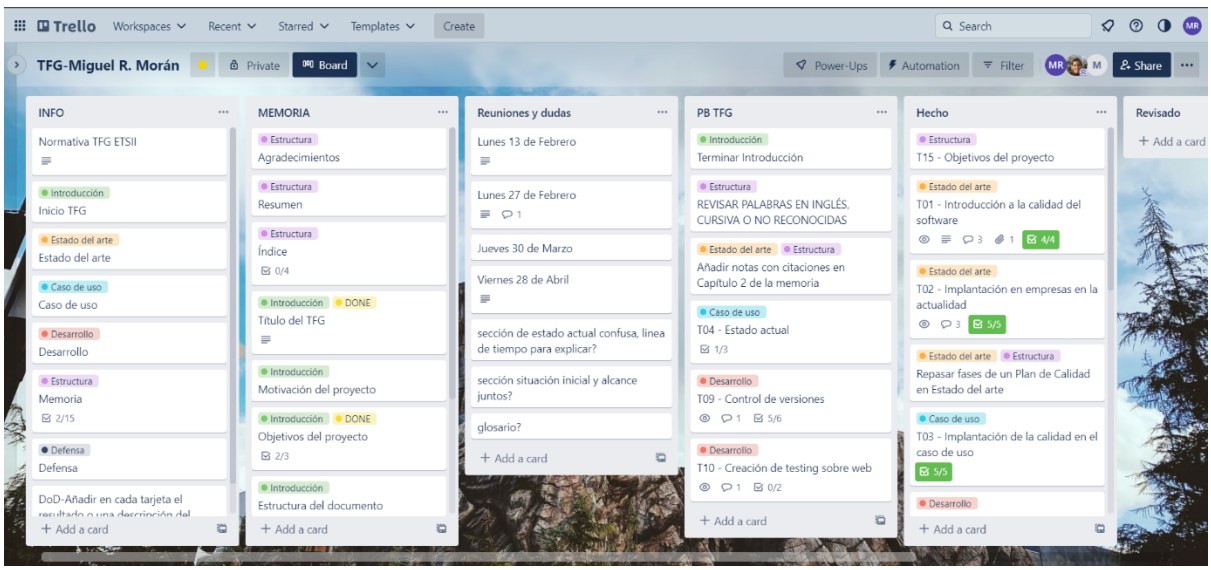

Ilustración 1. Captura del tablero del proyecto en Trello del actual TFG

Como se puede observar hay un total de 5 columnas en el tablero que tienen los siguientes propósitos:

- INFO: en esta columna encontramos información de ayuda al proyecto respecto a los principales campos a tratar como puede ser el estado del arte o la memoria, así como información relevante a tener en cuenta para la memoria.
- MEMORIA: en esta otra podemos encontrar todos y cada uno de los apartados que contendrá la memoria del proyecto con etiquetas que permiten hacer un rápido seguimiento del capítulo al que pertenecen y si están implementados ya o no.
- REUNIONES Y DUDAS: aquí encontraremos las reuniones principales realizadas hasta la fecha de la captura, así como ciertas dudas o anotaciones para futuras tutorías.
- PB, HECHO y REVISIÓN: en estas columnas tendremos algo parecido a un tablero Scrum con las tareas a realizar, realizadas y revisadas. Es una adaptación al típico tablero con las columnas de *ToDo*, *Doing* y *Done* que tantas veces se ha visto pero dirigido a el desarrollo de un proyecto por parte de una sola persona. Las tareas están etiquetadas con el capítulo al que pertenecen y almacenan información de utilidad para el desarrollo de la memoria.

La utilización de esta herramienta ha facilitado en gran escala el trabajo a realizar y ha permitido un orden y control de las tareas que no hubiera sido posible sin ella.

2 Estado del arte

En este capítulo se hablará de los aspectos más importantes para entender la Calidad del Software, su historia, el *testing* y el contexto en el que nos encontramos para realizar un posterior estudio sobre una determinada empresa.

Se explicarán los términos más importantes, así como etapas o motivaciones para las acciones más comunes en este ámbito.

2.1 Introducción a la calidad

2.1.1 ¿Qué es la calidad del Software?

La calidad del software es importante ya que este se utiliza en muchos aspectos de nuestra vida diaria, desde las aplicaciones móviles hasta los sistemas de control de tráfico aéreo. Cuando el software no cumple con los estándares de calidad, puede haber errores y fallas que pueden causar problemas graves, como la pérdida de datos o la exposición a vulnerabilidades de seguridad [1].

La calidad se trata de un término utilizado para referirse a la capacidad de un software para cumplir con ciertos requisitos funcionales y no funcionales especificados en alguna de sus diferentes etapas:

- Análisis de requisitos
- Diseño
- Implementación
- Verificación
- Mantenimiento

Además, es un elemento esencial a la hora de elaborar software ya que garantiza su utilidad, seguridad, fiabilidad y usabilidad.

Existen diferentes métricas que ayudan a medir la calidad de un software como pueden ser la eficiencia o la capacidad de mantenimiento, indicando la capacidad del software para realizar sus funciones de manera rápida y sin errores, o la facilidad con la que este se puede actualizar o modificar, respectivamente [2].

2.1.2 Historia de la calidad del software

La calidad del software ha evolucionado significativamente a lo largo de la historia de la informática. En sus primeros días, el software se desarrollaba de manera informal, sin estándares formales de calidad y sin procesos definidos[3]. A medida que la demanda de software aumentó y las aplicaciones se volvieron más complejas, se hizo evidente la necesidad de una forma más estructurada y rigurosa de desarrollo de software.

En la década de 1970, surgieron los primeros modelos de ciclo de vida del software, como el modelo en cascada, que establecieron un proceso estructurado y secuencial para el desarrollo de software. Estos modelos incluyeron un enfoque en la documentación y en la verificación y validación del software, lo que ayudó a mejorar la calidad del software.

En la década de 1980, surgieron nuevos modelos de ciclo de vida del software, como el modelo de desarrollo en espiral, que enfatizó la importancia de la retroalimentación y la mejora continua en el proceso de desarrollo de software. También se desarrollaron nuevos estándares de calidad, como el estándar ISO 9000, que se aplicó originalmente a la industria manufacturera, pero se adaptó para el desarrollo de software.

En la década de 1990, la calidad del software se convirtió en un tema central en la industria del software. Se desarrollaron nuevos estándares de calidad específicos para el software, como el modelo de madurez de capacidad (CMM) y su sucesor, el CMMI. Estos modelos establecieron un enfoque sistemático y basado en procesos para el desarrollo de software de alta calidad.

En la actualidad, la calidad del software sigue siendo un tema importante, de hecho, cada vez más, especialmente en el contexto de la creciente complejidad de las aplicaciones de software y la necesidad de desarrollar software de manera más rápida y eficiente. Se han desarrollado nuevas metodologías de desarrollo de software, como el desarrollo ágil, que enfatiza la colaboración y la flexibilidad para lograr una mejor calidad de software.

En resumen, la calidad del software ha evolucionado significativamente a lo largo de la historia de la informática, desde los primeros días de desarrollo informal hasta los enfoques estructurados y basados en procesos de hoy en día [4]. Los modelos de ciclo de vida del software, los estándares de calidad y las metodologías de desarrollo han evolucionado para mejorar la calidad del software y hacer frente a los desafíos cada vez mayores en el desarrollo de software.

2.2 Testing

La importancia del *testing* en la gestión de la calidad del software es indiscutible. El proceso de *testing* desempeña un papel fundamental en la detección y corrección de errores, garantizando que el software funcione según lo previsto y cumpla con los requisitos establecidos. Además, contribuye a mejorar la confiabilidad, la seguridad y el rendimiento del software, proporcionando una experiencia satisfactoria para los usuarios finales del mismo.

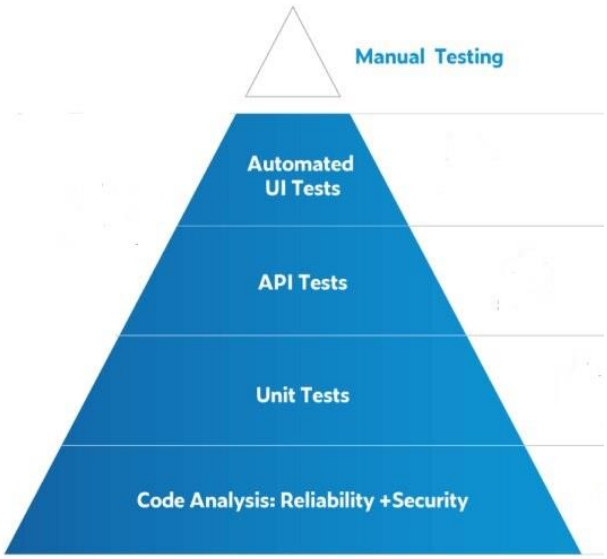

Ilustración 2. Pirámide de Cohn que estructura los principales tipos de pruebas

Para abordar el *testing* de manera efectiva, es común utilizar la pirámide de Cohn como una guía general. Esta pirámide propone una estructura jerárquica para los diferentes tipos de pruebas, clasificándolos según su nivel de abstracción y su velocidad de ejecución [5].

En la base de la pirámide se encuentran las pruebas de unidad, también conocidas como pruebas unitarias. Estas pruebas se enfocan en verificar el correcto funcionamiento de las partes más pequeñas y aisladas del software, como funciones o métodos individuales. Las pruebas unitarias se ejecutan de manera rápida y automatizada, y son esenciales para garantizar la integridad del código base de dicho software.

En el siguiente nivel de la pirámide se sitúan las pruebas de integración. Estas pruebas tienen como objetivo verificar la correcta interacción y comunicación entre diferentes componentes

del software. Aquí se comprueba que los módulos o subsistemas se integren de manera adecuada y que los datos fluyan correctamente entre ellos.

En el nivel medio de la pirámide se encuentran las pruebas de servicio, también conocidas como pruebas de API o pruebas de contrato. Estas pruebas se centran en verificar la interoperabilidad y el correcto funcionamiento de las interfaces de programación de aplicaciones (API). Se aseguran de que las diferentes partes del software se comuniquen y se integren adecuadamente a través de estas interfaces.

En la cúspide de la pirámide se encuentran las pruebas de interfaz de usuario (UI). Estas pruebas se enfocan en validar la experiencia del usuario final. Se verifican aspectos como la apariencia, la usabilidad y la interacción del software con el usuario. Aunque son importantes, este tipo de pruebas suelen ser más lentas y costosas de ejecutar en comparación con las pruebas de niveles inferiores.

Aparte de la pirámide de Cohn, existen otras estrategias y filosofías en el mundo del *testing* software. Algunas de ellas son:

- Pruebas de regresión: Se realizan para garantizar que los cambios o actualizaciones realizados en el software no hayan introducido nuevos errores o afectado negativamente a las funcionalidades existentes.
- Pruebas de carga y rendimiento: Se llevan a cabo para evaluar el comportamiento del software bajo condiciones de carga y estrés, asegurándose de que el rendimiento se mantenga dentro de los límites aceptables.
- Pruebas de seguridad: Se realizan para identificar y mitigar posibles vulnerabilidades y riesgos de seguridad en el software, protegiendo los datos y la integridad del sistema.
- Pruebas de usabilidad: Se centran en evaluar la facilidad de uso, la accesibilidad y la satisfacción general del usuario al interactuar con el software.
- Pruebas de compatibilidad: Se llevan a cabo para asegurar que el software funcione correctamente en diferentes plataformas.

2.3 Implantación de la calidad en empresas en la actualidad

En los siguientes apartados se hablará de la importancia de la calidad del software en el ámbito empresarial, estudiando la evolución que lo ha llevado a ser uno de los más importantes sectores del momento.

2.3.1 Transformación de la calidad del software en empresas

Como ya se ha mencionado en la anterior sección, la evolución del control de la calidad del software ha sido notoria y su relevancia en el mundo de la tecnología ha crecido acorde a ella. La evolución sufrida en todas las industrias se debe a como se ha incrementado la demanda de software y la competencia en el mercado en estos últimos años, así como la necesidad de satisfacer nuevas necesidades.

Con este rápido avance de la tecnología y creciente complejidad de los sistemas informáticos, la calidad del software, como se ha dicho, se ha convertido en un aspecto esencial para garantizar el éxito de un proyecto y la satisfacción del cliente. Además, ha llevado a un aumento en la demanda del software de alta calidad, y ha obligado a las empresas a tomar medidas para mejorar sus procesos de desarrollo y garantizar que sus productos cumplan con los estándares de calidad requeridos.

La calidad del software se ha introducido en las empresas actuales hasta convertirse en algo indispensable dentro de ellas. Se han dado cuenta de que la calidad del software no solo mejora la satisfacción del cliente, sino que también reduce los costos de mantenimiento y mejora la eficiencia del desarrollo. Algunas de las medidas tomadas por las empresas para garantizar la calidad del software, entre otras, son:

- Herramientas de automatización de pruebas
- Pruebas de seguridad
- Análisis estático, y dinámico del código
- Procesos de desarrollo software bien definidos y estructurados
- Métricas y objetivos de calidad
- Metodologías ágiles para permitir una mayor flexibilidad y adaptabilidad

Respecto a esta última, en la actualidad, se han implementado metodologías y técnicas de desarrollo de software que aseguran la calidad del producto final o al menos la calidad de los procesos en los que se desarrolla dicho producto. Entre estas metodologías se encuentran Agile, DevOps, Scrum, Kanban, entre otras, que se enfocan en la entrega continua de software de alta calidad y en la colaboración entre los miembros del equipo de desarrollo y los interesados en el proyecto [6].

En resumen, la calidad del software es un aspecto crítico que deben tener en cuenta hoy en día las empresas para asegurar el éxito en sus proyectos y trabajar de la manera más efectiva y correcta que se pueda.

2.3.2 Departamento de calidad

La introducción de los departamentos de calidad en las empresas se remonta a la década de 1960, cuando las empresas empezaron a darse cuenta de la necesidad de asegurar que sus productos cumplieran con los estándares de calidad. En este sentido, se comenzaron a crear departamentos de control de calidad encargados de asegurar la calidad en la producción y en los procesos.

Con el advenimiento de la tecnología de la información, se ha vuelto cada vez más importante para las empresas desarrollar software de calidad [7]. La implantación de un departamento de calidad específico para el software ha evolucionado a lo largo del tiempo, y actualmente se ha convertido en una práctica común en muchas empresas.

El departamento de calidad de software se encarga de garantizar que el software se desarrolle de manera adecuada y que cumpla con los estándares de calidad requeridos. Además, este departamento trabaja para asegurar que el software se mantenga y se mejore continuamente.

La implantación de un departamento de calidad de software en una empresa puede tener muchos beneficios. En primer lugar, permite una mayor eficacia y eficiencia en el desarrollo de software, ya que el departamento de calidad puede identificar y corregir problemas en las etapas tempranas del desarrollo. En segundo lugar, mejora la satisfacción del cliente, ya que el software se desarrolla teniendo en cuenta las necesidades del usuario final. En tercer lugar, el departamento de calidad de software puede ayudar a reducir los costos de mantenimiento del software, ya que el software se desarrolla teniendo en cuenta la facilidad de mantenimiento.

2.3.3 Plan de calidad, motivos para su implantación

Si se habla de departamento de calidad, hay que mencionar también de plan de calidad como recurso utilizado hoy en día para mejorar la calidad del software en los proyectos tecnológicos de una empresa.

La implantación de un Plan de Calidad es fundamental para cualquier empresa que quiera asegurar la calidad de sus productos y servicios relacionados con el mundo de la tecnología

[8]. Un Plan de Calidad es un conjunto de actividades planificadas y sistemáticas que se llevan a cabo para garantizar que los productos y servicios cumplan con los requisitos especificados y las expectativas del cliente.

En general, un Plan de Calidad se estructura en tres partes principales: planificación, implementación y control.

- En la primera fase, se define el alcance del plan, se establecen los objetivos y se planifican las actividades necesarias para alcanzarlos.
- En la segunda fase, se implementan las actividades planificadas.
- En la tercera fase, se lleva a cabo la evaluación y el control del proceso para asegurar que los objetivos se están cumpliendo.

Las funciones de un Plan de Calidad son numerosas, entre las que destacan la mejora de la calidad de los productos y servicios, la reducción de costes, el aumento de la eficiencia y la eficacia de los procesos, la mejora de la satisfacción del cliente y el cumplimiento de los requisitos y regulaciones legales. Además, un Plan de Calidad permite a las empresas medir y evaluar su desempeño, y establecer objetivos a largo plazo.

Sin embargo, la implantación de un Plan de Calidad también puede presentar algunas desventajas, como el costo y tiempo requerido para su desarrollo y mantenimiento, y la posible resistencia al cambio por parte de algunos empleados. No obstante, los beneficios de un Plan de Calidad superan ampliamente las desventajas, lo que lo convierte en una herramienta esencial para las empresas que quieran competir en un mercado cada vez más exigente.

3 Estudio de la calidad en la empresa

En este capítulo se tratará el estudio realizado sobre la empresa seleccionada y la cual se presentará en el primer apartado. Se trata de un estudio acerca de la implantación de la calidad con su respectivo Plan de Calidad, así como la evolución sufrida con el paso del tiempo y de diversas acciones realizadas sobre equipos que forman la compañía.

3.1 Implantación de la calidad en el caso de uso

Se comentará en esta sección como llega la calidad a los productos y servicios ofrecidos, así como en toda la organización de la empresa seleccionada para hacer el estudio.

3.1.1 Presentación del caso de uso

La empresa sobre la que se realizará el estudio de la calidad del software en este proyecto es una empresa que desarrolla aplicaciones móviles de ciclismo *indoor*. La empresa ofrece planes de entrenamiento personalizados y seguimiento de progreso para ayudar a los usuarios.

Nos encontramos ante una plataforma compuesta por un servicio de suscripción asociado a dos productos principales que son una aplicación de ciclismo y otra dedicada a clases de *fitness*, así como varias aplicaciones complementarias y adyacentes cuyo mantenimiento se encuentra en distintos estados, uno deprecado actualmente, aunque oficialmente no se ha comunicado su cese, y otras que han sido renovadas pero cuyo seguimiento es dispar. También hay otras aplicaciones o herramientas de uso para dar soporte a la empresa en distintos departamentos como es un chat que ayuda a derivar las dudas o problemas planteados por los usuarios y filtrar el volumen de trabajo del equipo de Soporte. Así mismo hay otros aplicativos y *BackOffice* que sirven para alimentar de una u otra forma a las otras dos aplicaciones principales o para las campañas de marketing, pero siempre dirigidas a las dos aplicaciones ya mencionadas.

Además, junto a los dos productos principales que forman un solo servicio están la web pública y la web privada. La web pública es el lugar donde los usuarios pueden conocer la marca y sus productos e iniciar el registro y suscripción. La web privada es el lugar donde los usuarios de ambos productos pueden gestionar sus cuentas, conocer sus avances, consultar sus registros, etc.

3.1.2 Motivos para la implantación

Como en toda empresa que se precie, en esta es necesario desarrollar un Plan de Calidad que asegure que los procesos de creación de sus productos, y por tanto los productos en sí mismos, cumplen con las expectativas de los clientes. La mejor forma de vender más es sin duda tener buena reputación y para ello, además de una buena estrategia de comunicación y marketing, se requiere de un producto con la calidad acorde a la demanda del mercado tanto en tiempo como en forma.

La empresa tiene un producto con una instalación de usuarios constante que se mantiene en el tiempo en número, pero con cierta volatilidad y rotación. Esto hace intuir que, a pesar de tener un buen producto, pierde usuarios por distintos motivos. La usabilidad y el diseño no son todo lo intuitivos que cabría esperar, además, y esto no es ajeno a otros productos similares, la complejidad técnica se diversifica y juega en su contra al haber tantos tipos de dispositivos hardware y tal diversidad de sistemas operativos, *firmwares*, versiones y distinta capacidad de procesamiento. Esto hace que la predicción de los errores sea muy complicada. Si sumamos los factores tecnológicos de usabilidad, conectividad y compatibilidad, escalabilidad, y mantenibilidad vemos que la necesidad de sistematizar los procesos de control de la calidad ya en sí nos obligaría a tener que diseñar un plan de gestión de la calidad, pero si además lo ponemos en relación con las dificultades que actualmente existen para gestionar versiones estables en especial el simulador y las dependencias con los servicios *back* que éste consume, es necesario intervenir en los factores que obstaculizan la simplificación de los procesos de integración, verificación, testeo, validación, cierre de cada versión y entrega.

Los objetivos de este plan de implantación de calidad deben ser por lo tanto reducir los riesgos inherentes a la tecnología particular en los puntos que suman mayor riesgo por su propia naturaleza (conectividad, compatibilidad, usabilidad, diseño, seguridad, escalabilidad, etc.) o que son más críticos para la arquitectura, el negocio, o el propio producto en sí; y simplificar los procesos tanto de definición del producto como de los procesos de desarrollo, pruebas, integración y entrega del mismo, descomponiendo en unidades de trabajo más pequeñas que nos permita realizar entregas más frecuentes, más dinámicas, fiables y flexibles. Esto implica definir cambios en las dinámicas de trabajo, en las estructuras organizativas de los equipos, en la infraestructura y en la gestión de la cooperación, la comunicación y la transmisión de la información.

3.1.3 Situación inicial: Riesgos, fases y objetivos

En octubre de 2020 se realiza una presentación con la que da comienzo la implantación del Plan de Calidad y que expone la situación inicial de la que parte la empresa, así como un *roadmap* que ayude a visualizar todo el trabajo que se pretende hacer a lo largo del tiempo.

Lo primero que se expone en esta presentación son una serie de riesgos identificados tras un previo estudio realizado durante unos meses en la empresa en los que se entrevista y supervisa a todos los diferentes equipos que componen la empresa. Estos riesgos se dividen en tres categorías:

Riesgos de planificación y producto

Ciclos de producto *delivery* en producción de 4 meses lo cual genera:

- Desequilibrio entre las necesidades de negocio y la capacidad de entregar actualizaciones a clientes/usuarios.
- Lejanía del *time to market* deseable, propiciando la pérdida de oportunidades.

En cuanto a la planificación, se realizan peticiones no planificadas y sin respetar scrum, produciendo:

- Pérdida de foco en objetivos, afectando a la capacidad de priorizar, estresando los procesos, reduciendo la calidad y la productividad y posponiendo el *delivery*.
- Falta de una definición detallada de las historias de usuario o en reportes de incidencias (*bugs*).

Riesgos de la tecnología

En lo referido a la tecnología existen varios motivos causantes de consecuencias negativas:

- Los sistemas CI son útiles, pero no están optimizados para el marco agile.
- No hay estrategia de *testing* técnico, se realizan pocas pruebas y tarde.
- No hay una herramienta donde gestionar y registrar las pruebas y sus resultados. No hay *testing* unitarios ni de integración ni test de contrato.
- No hay un sistema de distribución de versiones sencillo.
- No hay forma de actualizar parcialmente.

- Los entornos no son los necesarios para evitar bloqueos en los flujos de trabajo y se utilizan recursos de producción y eta para los flujos de desarrollo interno y mixtos (bbdd, servicios).
- La deuda técnica acumulada es grande y se distribuye en servicios críticos.

Riesgos de la calidad

En esta sección encontramos riesgos relacionados a las dinámicas de trabajo como pueden ser:

- No se respetan las ceremonias scrum

- Se aceptan peticiones poco planificadas

- Estimación demasiado optimista generando falsas expectativas y desvío en la planificación.

Una vez presentados los riesgos que existían en la situación inicial comentada en el apartado anterior y de la cual parte la empresa en octubre de 2020, se establecen las fases que tendrá el plan de calidad propuesto. Son los siguientes:

Fase 1 – Implantación del sistema de gestión de la calidad: revisión del proceso de Control de Calidad (inspección, corrección y mejora de producto y sus procesos de desarrollo) e inicio de los procesos de Aseguramiento de la Calidad (prevención de riesgos).

Fase 2 – Afianzamiento: se caracteriza por ampliar el Aseguramiento de la Calidad y extender a más áreas los procesos, metodologías y procedimientos.

Fase 3 – Crecimiento: afectar a todos los procesos, productos, áreas y personas de la compañía adaptando el modelo de negocio y la tecnología.

Fase 4 – Mejora continua: esta fase es transversal. Cuando el aseguramiento y control han sido optimizados, los procesos de negocio y tecnología están preparados para la mejora constante.

Junto a las fases del Plan se establecen también los objetivos principales del Plan de Calidad que serán:

- De planificación y producto: reducir la incertidumbre, *product delivery* cada 6 semanas.
- De tecnología: mejorar la actividad de *testing* e integración y mitigar riesgos.
- De calidad: fomentar *agile* e implantar procesos de prevención y control.

3.2 Estado actual

Desde que se decidió implantar el Plan de Calidad en la compañía se han ideo realizando evaluaciones para poder realizar un estudio del trabajo y los avances realizados. Estas evaluaciones se presentan como documentos que reflejan y describen la evolución de la compañía en base a los objetivos presentados y definidos anteriormente, y sirve para redefinir estos objetivos, en caso de ser necesario, y revisar los próximos pasos a seguir.

A continuación, se presenta una línea temporal que resume los hitos acontecidos durante los últimos 3 años en la empresa respecto a la implantación del Plan de Calidad.

Ilustración 3. Línea temporal que resume los hitos acontecidos desde la implantación hasta la actualidad.

3.2.1 Fase 1: Implantación

Tras la presentación de la situación inicial en octubre de 2020, también se realiza una presentación de la fase 1, la Implantación, con la cual se da comienzo oficialmente al Plan de Calidad en la compañía.

Este documento es el primero dedicado a integrantes de la compañía fuera de las áreas encargadas de tomar las decisiones, ya que el documento anteriormente explicado (situación inicial) fue únicamente dirigido para dichos responsables. Significa una primera toma de contacto con la calidad para los trabajadores ya que en la empresa no existía ningún plan sistemático de calidad ni en los procesos de desarrollo ni en la definición de los productos ni en ningún área, desoyendo las buenas prácticas de la industria como ya se ha mencionado en otros apartados. En él se explica e introduce de manera estructurada los diferentes conceptos que conforman el mundo de la calidad del software (Aseguramiento de la calidad, control de la calidad, gestión de la calidad, …), así como el mundo de las metodologías ágiles y sus beneficios a la hora de estructurar y gestionar el trabajo. Se trata de una introducción al mundo de la calidad para entender su necesidad y el alcance que puede llegar a proporcionar en los marcos de trabajo establecidos.

Además, como ya se ha mencionado, también se definen los principales objetivos a seguir en la primera fase del Plan de Calidad (Fase de Implantación). Estos, están destinados a las áreas de producto y tecnología únicamente en un inicio y son los siguientes:

1. Reducir la incertidumbre: estableciendo dinámicas de trabajo ágiles acordes al marco Scrum estableciendo ceremonias de refinamiento y planificación.
2. Establecer una estrategia de pruebas.
3. Descomponer las unidades de trabajo para estimar y planificar mejor.
4. Entregas internas más pequeñas y manejables con mayor frecuencia: cada 2 y 3 semanas, mejorando la estrategia de integración.
5. Mitigar los riesgos de la tecnología: reduciendo las ratios de error.
6. Establecer una infraestructura de entornos versátiles.
7. Utilizar un sistema de integración (CI) acorde al marco de trabajo acordado.
8. Establecer una política de *product delivery* ágil: iteraciones incrementales de 4 y 6 semanas con ciclos de producto completo cada 1 y 2 meses.
9. Adaptar los procesos de desarrollo del producto a los tiempos del mercado.

En junio de 2021 se presenta la primera evaluación del Plan de Calidad, en el que se analiza el trabajo realizado desde octubre de 2020 (Inicio del Plan de Calidad) hasta abril de 2021, analizando así 6 meses naturales dividiendo el estudio en dos segmentos.

Primer segmento: implantación del sistema de gestión

El mes de octubre supuso la puesta en práctica del trabajo enfocado desde la metodología Scrum. Sobre todo, en el equipo de la aplicación de clases de *fitness* (a partir de ahora llamado *fitness*) ya que era el laboratorio perfecto tanto por su reducido número de integrantes (de 3 a 5 personas como involucradas máximo) como por las características tecnológicas (aplicaciones nativas con restricciones y una fuerte refactorización justo antes del inicio de los procesos de calidad).

El mes de noviembre se enfocó en la definición de las dinámicas de trabajo, el objetivo fue comenzar a definir las historias de usuario con un mínimo de contenido y una estructura escueta pero constante y paliar la falta casi total de refinamientos formales y de planificación en cuanto a la duración de los *sprints*.

Durante el mes de diciembre lo más significativo fue la implantación de la estrategia de ramas basada en *Git Flow* que quedó acordada y establecida y poco a poco comenzó a implementarse y establecerse como el método de trabajo en todos los equipos. También se acordó el inicio de trabajo de la creación de los nuevos entornos necesarios para la mejora de la estrategia de desarrollo, integración y pruebas, así como de la optimización de los procesos de despliegue.

Segundo segmento: implantación y desarrollo

El segundo segmento se caracteriza por cambios en las dinámicas de los equipos especialmente en el encargado de la app de ciclismo. Se continuó con el trabajo de implementación de test unitarios en *fitness* y web de forma más pronunciada, comenzando a determinar el objetivo de test unitarios en ciclismo.

También, en cuanto a la librería de comunicaciones (librería que permite la comunicación entre las aplicaciones de la compañía y los diferentes dispositivos que permiten el correcto uso de los servicios que las aplicaciones prestan, bicis, rodillos, pulsómetros, etc.) que tanto afecta a ciclismo y *fitness*, se definió, estableció y comenzó a utilizarse el proceso de certificación oficial interno.

Se siguió trabajando para poder mejorar y optimizar la infraestructura, y se terminó de montar los nuevos entornos que durante más adelante ya se comenzarían a utilizar durante el mes de mayo en *backend* y web y poco a poco extendiéndose al resto.

3.2.2 Fase 2: Afianzamiento

Tras la primera evaluación realizada en abril de 2021 y el estudio previo de los responsables del Plan de Calidad, se presenta en junio de ese mismo año la segunda fase del Plan de Calidad, la fase de Afianzamiento. Con esta presentación se da oficialmente comienzo a la nueva fase pese a tener deuda de la primera como se ha explicado al final del apartado anterior ya que se determina la importancia de comenzar con nuevos objetivos para no paralizar el proceso que supone el Plan de Calidad en la empresa.

Esta nueva fase se caracteriza por ampliar el Aseguramiento de la Calidad y extender a más áreas y niveles los procedimientos de Control de Calidad y Aseguramiento de los procesos. Con el objetivo de paliar los riesgos evaluados durante la 1º fase que afectó a las áreas de producto y tecnología, se amplía estas áreas incluyendo a negocio, *marketing* y dirección.

Los objetivos establecidos en junio de 2021 para esta nueva fase son los siguientes:

1. Completar la deuda del plan de calidad (Fase 1: Implantación).
2. Reducir el impacto de la incertidumbre del negocio.
3. Focalizar los esfuerzos en el *product delivery*.
4. Mejorar la integración continua (CI/CD).
5. Crear los nuevos entornos dinámicos.
6. Utilizar los nuevos entornos dinámicos.
7. Establecer métricas de calidad.
8. Automatización de los procesos, *pipelines* y tareas.
9. Gestionar y comunicar mejor las entregas.
10. Crear y mantener un cronograma de planificación.

Como podemos observar, la mayoría de estos objetivos pertenecen aún a la fase 1 del Plan de Calidad.

<u>Segunda evaluación</u>

La segunda evaluación fue presentada en enero del 2022, analizando el trabajo realizado desde mayo de 2021 hasta enero de 2022. Esta es realizada tras menos de un año debido a varias razones, entre ellas, el comienzo de implantación de nuevos proyectos dentro de la empresa que suponen un cambio a nivel de software significativo. También el hecho de que hayan

pasado 6 meses desde la presentación de los objetivos y la necesidad de un nuevo *roadmap* para 2022.

Se trató con prioridad la finalización de mejoras en entornos e integración continua y avances en los procesos de definición y entrega dentro del área de producto. Se destaca también el establecimiento de una planificación con un cronograma conocido y actualizado para optimizar los recursos, ahorrando así el esfuerzo y reinvirtiendo en la mejora de los productos, servicios y procesos.

En cuanto a las principales mejoras sufridas dentro de los diferentes equipos se encuentran:

- Mayor estrictez en la duración de *sprints*.
- Establecer mecanismos de análisis temprano.
- Mejorar y anticipar cambios en las comunicaciones a los usuarios.
- Revisión de política de ramas.

Además, cabe mencionar que, en noviembre de 2021, en el equipo de *fitness*, se vieron totalmente interrumpidas las ceremonias scrum abandonando así dicha metodología de trabajo y pasando a un modelo Kanban artificial marcado por la refactorización técnica de Android y el aprovechamiento de la investigación e implementación de AppleWatch e investigación de AppleTV.

Tercera evaluación

Por último, tenemos la tercera evaluación presentada en noviembre de 2022 con un alcance desde enero de 2022 hasta noviembre de 2022. Es la última evaluación presentada hasta la fecha en la compañía y encargada de evaluar el estado de esta desde el comienzo del Plan de Calidad hasta fecha de hoy, ya que pese a ser realizada en noviembre del año pasado, nos sirve para conocer los resultados de todo el trabajo realizado.

Esta evaluación tiene un claro propósito y es dar por concluida la Fase 1 (Implantación) del Plan de Calidad por completada tras cumplir con todos los objetivos propuestos en octubre de 2020 cuando se dio comienzo dicha fase listados en el apartado anterior. Pese a estar completa la fase 1, siguen sin haberse resueltos todos los objetivos de la fase 2 propuestos en junio de 2021.

Se tratará con más detalle el trabajo realizado durante estos dos años en el capítulo 5.

4 Implementación de la automatización de pruebas

4.1 Presentación del proyecto

En junio de 2021 se presentaron varios objetivos con motivo del comienzo de la segunda fase del Plan de Calidad, entre los que se encontraba el objetivo de la automatización de los procesos, *pipelines* y tareas. Además, tras la tercera evaluación, se detalla aún más dicho objetivo estableciendo como nueva propuesta en diciembre de 2022 para el equipo de QA de la empresa, el comienzo de un proyecto de automatización de pruebas.

Como se ha mencionado, el responsable de dicho proyecto será el equipo de QA de la compañía el cual tras una nueva incorporación en noviembre de 2022 se ve capacitado para abordar la tarea de desarrollar un nuevo *framework* que suponga una mejora en la capacidad de *testing* mejorando así el control y gestión de la calidad de los productos ofrecidos al público.

Tras estudiarse junto a la directiva de la empresa se decide que el comienzo de la automatización de pruebas debe ser testear la página web pública (toda la web visible al público que no está aún registrado en el sistema) y el flujo de suscripción de un usuario a un plan lo cual implica un ingreso para la empresa.

En este proyecto se presentará la DEMO que se presentó en el mes de abril al departamento de producto y que supone una versión reducida en funcionalidades y prestaciones de lo que será en un futuro el *framework* pero que sirve para mostrar la estructura y demostrar lo que se es capaz de conseguir con el proyecto. Dentro de esta DEMO se han desarrollado dos conjuntos de pruebas que contienen a su vez 2 pruebas cada uno, dos pruebas de *login* y dos pruebas de registro. Cabe resaltar que la importancia de esta demostración reside en la estructura que se ha creado en el proyecto y algunas de las funcionalidades que no supondrán un reto en el futuro ya que se ha conseguido resolver tras el trabajo realizado en este tiempo. En el capítulo 5 de este proyecto (Conclusiones y trabajos futuros) se hablará de los próximos pasos a seguir en lo referido a este proyecto de automatización.

4.2 Herramientas usadas y metodología

A continuación, se repasarán las diferentes herramientas utilizadas en esta primera DEMO de pruebas web del *framework* que pertenece al proyecto de automatización de pruebas. Se hará

hincapié en las herramientas más importantes, así como las metodologías usadas en la implementación de estas pruebas.

La base del proyecto se basa en el lenguaje Java como lenguaje de programación principal (dejando abierta la posibilidad de la utilización de otros lenguajes que ayuden a la implementación de nuevas funcionalidades). Java es conocido por su portabilidad, lo que significa que el código Java se puede ejecutar en diferentes sistemas operativos sin cambios significativos en él.

En cuanto al JDK, paquete el cual nos permite desarrollar aplicaciones basadas en Java se ha decidido usar la versión 17 ya que pese a ser la 8 la más usada, este es un proyecto de futuro y se ha pensado que muchas funcionalidades que integra ya esta versión podrán beneficiarnos más adelante.

Por otro lado, tenemos la gestión de dependencias la cual se lleva a cabo mediante la herramienta Maven, la cual permite administrar y descargar dependencias de forma automática y de manera muy cómoda desde el archivo del proyecto llamado pom.xml . En el podemos encontrar entre las etiquetas <dependency> cada una de las librerías que se usan en el proyecto con diferentes fines como puede ser la utilización del *framework* TestNG o la librería poi de org.apache .

El IDE seleccionado para el desarrollo del proyecto ha sido IntelliJ debido a su previo uso por parte de los integrantes del equipo y del resto de la compañía y por las diferentes funcionalidades que permiten que su usabilidad sea excelente.

Selenium WebDriver

La herramienta principal del *framework* dirigido a la automatización de pruebas web, y en la que se basa, es WebDriver. Selenium WebDriver es una herramienta de automatización de pruebas que permite a los desarrolladores automatizar interacciones en un navegador web. Proporciona una amplia variedad de opciones para localizar elementos de la página web y realizar acciones sobre ellos, como pulsación sobre botones, escribir texto en campos de entrada y comprobar el contenido de la página.

Selenium WebDriver requiere del uso de controladores específicos de cada navegador para funcionar de manera correcta por lo que se debería descargar y añadir cada uno de estos para poder usarse en los test a implementar. A pesar de utilizar únicamente el navegador Chrome para esta DEMO, se ha buscado una solución al problema de versiones que ocurría al

actualizarse de manera automática el navegador de los equipos necesitando así descargar la versión correspondiente del controlador (en este caso ChromeDriver). WebDriverManager es una librería de código abierto que se encarga de gestionar los controladores de todos los navegadores compatibles con WebDriver de una manera automática, utilizando la versión que corresponda en cada momento.

POM (Page Object Model)

Los patrones de diseño son soluciones probadas y documentadas a problemas comunes en el desarrollo de software permitiendo a los desarrolladores diseñar soluciones más eficientes y mantenibles y, además, permite una mejor comunicación entre los miembros del equipo de desarrollo y ayudan a reducir la complejidad del código.

El patrón de diseño Page Object Model es una técnica para la automatización de pruebas que se utiliza para separar la lógica de prueba de la lógica de la interfaz de usuario. Con POM, se crean los objetos que representan los diferentes elementos de la interfaz de usuario, lo que permite a los desarrolladores escribir pruebas más legibles y mantenibles. Por todas estas razones, se ha decidido usar este patrón para estructurar el proyecto.

Consiste en estructurar los archivos del proyecto en tres tipos:

- Archivos Page: en ellos podemos encontrar la lógica de la interfaz de usuario que supone cada una de las páginas de la web a probar
- Archivos Test: contienen los conjuntos de pruebas a ejecutar y que hacen uso de los archivos page para interactuar con las páginas.
- Archivo Base: este contiene toda la interacción con el controlador del navegador y es de la clase de la que heredan todos los archivos Page.

TestNG

Se trata de un conjunto de pautas, reglas, estándares de codificación, gestión de datos de prueba que supone un estándar en los procesos de automatización de pruebas, es decir, un marco de automatización. Este marco de automatización de pruebas para Java se basa en NUnit y Junit, *frameworks* con los cuales los equipos de la compañía ya están familiarizados y que añade nuevas funcionalidades para hacerlo más usable y poderoso en cuanto a prestaciones.

Está pensado para abarcar todos los niveles: pruebas unitarias, de sistemas, de integración; y de todos los tipos: funcionales, de regresión, pruebas E2E (como es el caso en esta DEMO).

GitLab

Los sistemas de control de versiones son herramientas fundamentales para gestionar eficientemente el desarrollo de software y el trabajo colaborativo en proyectos. Permiten rastrear y controlar los cambios realizados en los archivos a lo largo del tiempo, facilitando la colaboración entre desarrolladores, manteniendo un historial completo de modificaciones y permitiendo la reversión a versiones anteriores cuando sea necesario. Para este proyecto se ha decidido usar GitLab como sistema de control de versiones debido su uso en el resto de los proyectos existentes dentro de la compañía.

Para el trabajo realizado en esta primera DEMO del proyecto se ha optado por una política similar a la seguida en los otros proyectos mencionados. Se trata de mantener dos ramas de vida infinita dentro del repositorio y varias (como mucho 3 en este caso ya que únicamente trabajan 3 integrantes en el equipo de calidad responsable del proyecto) ramas *feature*:

- *features*: ramas dedicadas a desarrollar una nueva funcionalidad o que tienen un único propósito. Son ramas que desaparecen una vez son *mergeadas* a *develop*.
- *develop*: rama a la que se deben ir *mergeando* todas las ramas *feature* e integrando sus funcionalidades.
- *master*: se trata de la rama que almacena una versión estable del proyecto.

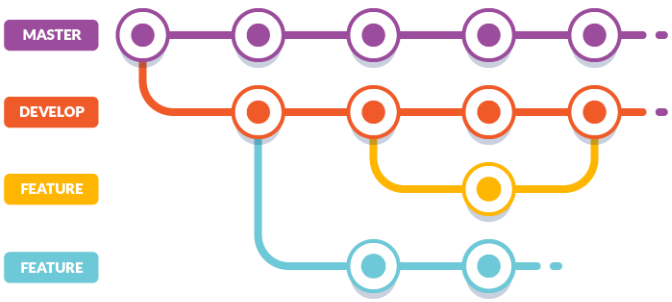

Ilustración 4. Ilustración de la política de ramas seguida en el proyecto con los tres tipos de ramas explicados anteriormente.

Cada cierto tiempo se revisa *develop* tras integrar en ella varias funcionalidades. Cuando *develop* es estable, se realiza un *mergeo* a master para constatar una nueva versión del proyecto.

También es importante la correcta utilización de los *commits*. Estos son confirmaciones de trabajo realizado sobre la versión actual de una rama. Tras realizar ciertos cambios en el código

debemos confirmar estos para poder enviarlos al repositorio y ser integrados en la rama correspondiente. Para este proyecto se ha cuidado la descripción de cada *commit* para poder hacer un mejor seguimiento de los cambios sufridos en la rama que se desea integrar en *develop*.

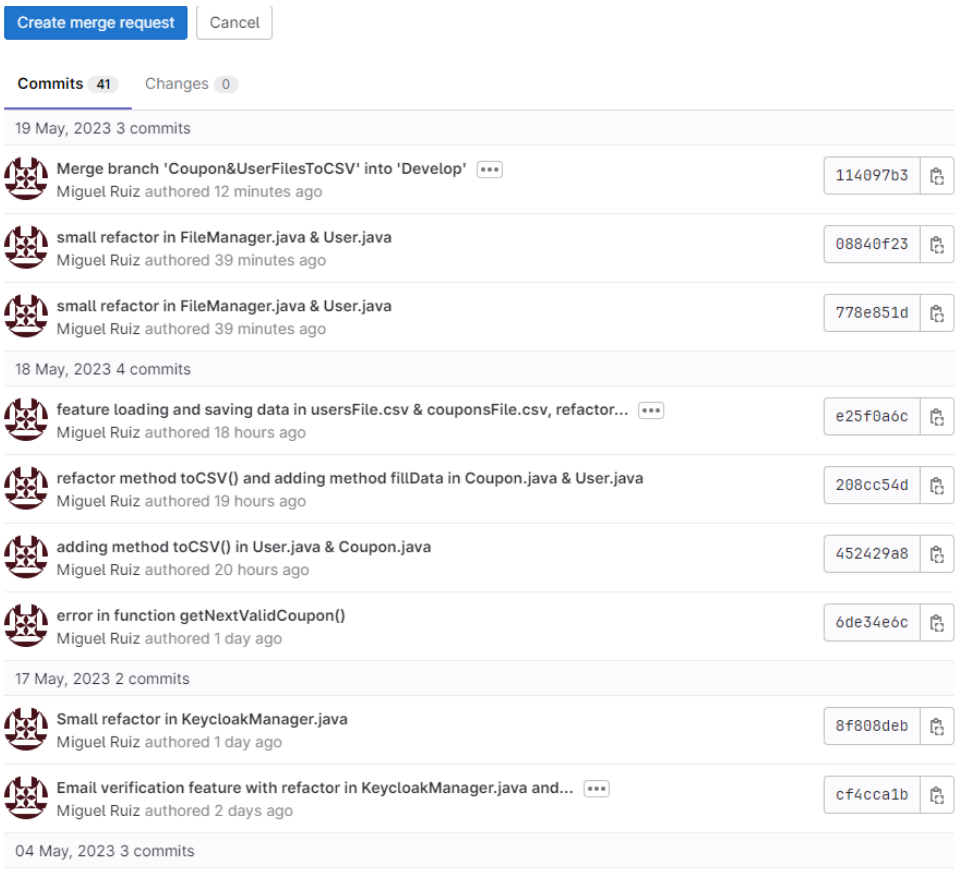

Ilustración 5. Ejemplo de traza de Gitlab de los commits registrados en la rama "Coupon&UserFilesToCSV"

Como se muestra en la imagen a través de la web de GitLab podemos observar los *commits* realizados en la rama “Coupon&UserFilesToCSV” así como la petición en sí de *mergeo* la cual necesita ser revisada y aprobada antes de ejecutarse.

4.3 Estructura

Este proyecto tiene una estructura común a cualquier proyecto Java creado con Maven y el resto de las herramientas mencionadas en el apartado anterior.

Recursos

En la carpeta de recursos del proyecto encontramos dos archivos CSV, documentos en formato abierto sencillo para representar datos en forma de tabla, en la que las columnas se separan por comas y las filas por saltos de línea. El archivo couponsFile.csv contiene los códigos de los cupones necesarios para realizar las pruebas de tipo registro de usuario que veremos más adelante, así como un campo que nos indica si está en uso o no, esto es para conseguir una mayor independencia por parte del proyecto y evitar la necesidad de añadir nuevos códigos en cada ejecución. El archivo usersFile.csv por otra parte, almacena la información de cada usuario registrado de manera que se consiguen dos beneficios:

1. Posibilidad de realizar un mejor seguimiento de la creación de usuarios y la información respectiva a cada uno de ellos, pudiendo ser consultada en cualquier momento de manera sencilla y ordenada.
2. Añadir independencia al proyecto de igual manera que se consigue con el archivo de los cupones ya que al crear un nuevo usuario, la dirección de correo necesaria para ello se genera automáticamente teniendo en cuenta los creados posteriormente.

Main/Java

En ella encontramos por ahora únicamente el paquete utils con cuatro archivos. El archivo Coupon y el archivo User son clases que definen la estructura de los objetos que se usan en el proyecto. Usar la programación orientada a objetos en este caso nos permite hacer un uso más eficiente de la información recurrente que se usa y se usará más adelante en las pruebas.

```
public class User implements Serializable {

    //propiedades
    4 usages
    private int numberID;
    4 usages
    private String name;
    4 usages
    private String surname;
    4 usages
    private String email;
    4 usages
    private String password;
    4 usages
    private String subscription;
    4 usages
    private String registerDate;
```

Ilustración 6. propiedades clase User

```
//Constructor

11 usages  Miguel Ruiz
public User() {}

//Otros métodos

1 usage  Miguel Ruiz
public String toCSV() {

    return this.numberID+","+this.name+","+this.surname+","+this.email+","+this.password+","+
            this.subscription+","+this.registerDate;
}

1 usage  Miguel Ruiz
public void fillData(String[] data) {

    this.numberID = Integer.parseInt(data[0]);
    this.name = data[1];
    this.surname = data[2];
    this.email = data[3];
    this.password = data[4];
    this.subscription = data[5];
    this.registerDate = data[6];
}
```

Ilustración 7. métodos clase User

Por otro lado, tenemos el archivo FileManager, clase la cual es usada para la gestión de los dos archivos .csv mencionados. Al comienzo de la ejecución de los conjuntos de pruebas carga la información de los archivos creando objetos y almacenándolos en una variable para poder acceder a ellos cuando la prueba lo requiera.

```
public void getInstance() {

    //Leemos del archivo .xlsx las tuplas correspondientes a los usuarios creando objetos de tipo
    //utils.User con la info y los añadimos a la variable userList

    File usersFile = new File( pathname: "C:
    String[] dataU;
    try (Scanner scFileU = new Scanner(usersFile)) {

        while(scFileU.hasNextLine()) {

            User user = new User();
            dataU = scFileU.nextLine().split( regex: ",");
            user.fillData(dataU);
            userList.add(user);
        }
        this.nextUserId = userList.get(userList.size() - 1).getNumberID() + 1;
    } catch (Exception e) { System.out.println("Error al cargar datos de usuarios"); }
```

Ilustración 8. método getInstance de la clase FileManager

De igual manera, al final de la ejecución de las pruebas se llama al método encargado de almacenar la información de las variables en los archivos pertinentes.

```
public void setInstance() {                                                    ⚠ 4  ✓ 57

    //Guardamos en el archivo usersFile.csv el contenido de la variable UserList

    File usersFile = new File( pathname: "C:
    try (FileWriter fwU = new FileWriter(usersFile)) {

        for (User user : userList) {

            fwU.write( str: user.toCSV() + "\n");
        }
    } catch (Exception e) { System.out.println("Error al guardar datos de usuarios"); }
```

Ilustración 9. método setInstance de la clase FileManager

En ambas imágenes mostradas se observa únicamente tanto la extracción de datos de usuarios y su guardado y no los de cupones ya que se realiza con una estructura similar.

Por último, tenemos el archivo KeycloakManager, el cual se encarga de hacer peticiones al servidor de Keycloak de una manera cómoda y sencilla. Keycloak es un producto de software de código abierto que pone a disposición de la empresa las siguientes funcionalidades: Inicio de sesión única, gestión de identidades y gestión de acceso. Por lo que para la implementación de algunas pruebas es necesario el uso de dichas peticiones.

Test/Java

En esta carpeta se encuentran los conjuntos de test que se ejecutarán para probar la web. Como ya se explica en el apartado anterior al hablar del patrón de diseño seguido en el proyecto existen tres tipos de archivos que se almacenan en una carpeta Pages, una carpeta Test y, por último, el archivo Base.

4.4 Conjuntos de pruebas

En los siguientes apartados se explicará el código correspondiente a cada prueba y se estudiarán los resultados de ellas tras su ejecución.

4.4.1 LoginTest

Los test que conforman el conjunto de pruebas de *login* se encuentran en el archivo llamado loginTest.java. Este conjunto consiste en dos sencillas pruebas independientes pero que gracias a las funcionalidades que nos ofrece TestNG podemos ejecutar en conjunto para que se realicen seguidamente.

Lo primero que podemos observar en la clase es la declaración de las propiedades a usar

```
public class LoginTest {

    // Encargado de ejecutar los métodos que controlan el navegador
    5 usages
    private WebDriver driver;

    // Variables tipo Page que permiten comunicar los tests con las diferentes páginas de la web
    13 usages
    HomePage homePage;
    5 usages
    LoginPage loginPage;
    7 usages
    LoggedHomePage loggedHomePage;
```

Ilustración 10. propiedades de la clase LoginTest

Encontramos aquí, la declaración de las variables para las que se creará un objeto de cada una de las clases Page que vamos a usar, en este caso son la página de Home, la de Login y la de Home estando logado (la cual es diferente a la Home del inicio). Estos objetos servirán para comunicarse con los elementos de la página y realizar las acciones que se deseen sobre ellos. También hay una variable que contendrá un objeto de la clase WebDriver el cual se pasará por parámetro a constructores (funciones de inicialización de una clase) de las clases Page. Esto es debido a que estas heredan todas como bien se ha mencionado de la clase Base la cual debe tener asociado el controlador del navegador a usar (variable driver).

Tras la declaración de las propiedades encontramos los métodos propios del *framework* explicado TestNG.

- SetUp(): el cual crea las clases de los objetos page declarados, así como la llamada al método chromeDriverConnection() desde la página con la que van a comenzar los test.

```
public WebDriver chromeDriverConnection() {

    WebDriverManager.chromedriver().setup();
    ChromeOptions options = new ChromeOptions();
    options.addArguments("user-data-dir=C:                                                                                    ");
    options.addArguments("profile-directory=          ");
    driver = new ChromeDriver(options);
    return driver;
}
```

Ilustración 11. método chromeDriverConection de la clase Base

Este método inicializa el controlador de Chrome con las características especificadas, en este caso, el perfil de Google Chrome a cagar ya que será necesario iniciar el navegador con la sesión iniciada con la cuenta de la empresa para poder acceder a la web a probar.

```
@BeforeClass
public void setUp() {

    //Creación de los objetos asociados a las propiedades

    homePage = new HomePage(driver);
    driver = homePage.chromeDriverConnection(); //Se crea el controlador que irá asociado a las Pages
    loginPage = new LoginPage(driver);
    loggedHomePage = new LoggedHomePage(driver);
    homePage.maximize(); // Se maximiza la ventana para una correcta ejecución
}
```

Ilustración 12. método SetUp de la clase LoginTest

Este método tiene la etiqueta @BeforeClass por lo que se ejecutará una única vez para todo el conjunto de pruebas.

- setUpTest(): el cual determina la URL a la que debe dirigirse el controlador, además de hacer un previo *logout* y borrar las cookies para evitar posibles errores.

```
@BeforeMethod
public void setUpTest() {

    /*Se hace logout gracias a keycloak y se visita la página de Home de la web para
    contemplar un flujo real, también se borran las cookies y se vuelven a aceptar
    por si llegase a influir en la ejecución en algún punto*/

    homePage.visit( url: "                                                    ");
    homePage.visit( url: "                                    ");
    homePage.deleteCookies();
    homePage.acceptCookies();
}
```

Ilustración 13. método setUpTest de la clase LoginTest

Este método tiene la etiqueta @Beforemethod ya que se ejecutará una vez por cada prueba del conjunto de pruebas a ejecutar. Esto es así ya que se pretende iniciar desde un mismo punto y definir pruebas *End to End.*

- tearDown(): el cual determina las acciones a realizar tras la ejecución del conjunto de pruebas ya que posee la etiqueta @AfterClass.

```
@AfterClass
public void tearDown() {

    //Se cierra el navegador

    driver.quit();
}
```

Ilustración 14. método tearDown de la clase LoginTest

LoginUser()

```
@Test
public void loginUser() throws InterruptedException{

    /* Existe un assert por cada página visitada durante el flujo de la prueba para comprobar
    que la ejecución es correcta. Adicionalmente se controla con mensajes en la consola
    los pasos que se van ejecutando*/

    Assert.assertTrue(homePage.isDisplayed(homePage.getHomePageLocator()), s: "No se encontró el localiz
    System.out.println("Redirección a página de Home");
    homePage.goToLogin();
    Assert.assertTrue(loginPage.isDisplayed(loginPage.getLoginPageLocator()), s: "No se encontró el loca
    System.out.println("Redirección a página de Login");
    loginPage.loginUser();
    System.out.println("Login realizado correctamente");
    Assert.assertTrue(loggedHomePage.isDisplayed(loggedHomePage.getLoggedHomePageLocator()), s: "No se e
    System.out.println("Redirección a página de Home logado");
    Thread.sleep( millis: 5000);
}
```

Ilustración 15. test loginUser de la clase LoginTest

Se trata de un método anotado con la etiqueta @Test por lo que pertenece al conjunto de pruebas que esta clase es capaz de ejecutar. Como observamos en su declaración tiene la línea "throws InterruptedException" lo cual permite usar métodos como el que vemos en la última linea que duerme la ejecución para ayudarnos a una mejor visualización.

En este método podemos encontrar cada llamada a los métodos que interesan de cada página creando así el flujo que se desea probar. Adicionalmente se han añadido *asserts* por cada paso a una nueva página para comprobar el correcto funcionamiento de los redireccionamientos que permiten poder tener un flujo correcto. Así como un mensaje de confirmación al ejecutar correctamente cada paso.

1. Tras situarse en la página HomePage se ejecutará el método goToLogin(), el cual ejecuta pulsación sobre el botón de Login situado en la esquina superior derecha de la página. Este botón está localizado, al igual que el resto de los elementos de las páginas mediante un *xpath* al que está ligado, este se guarda en una variable dentro de la clase HomePage desde donde se ejecuta el método mencionado.

```
public void goToLogin() { click(loginButtonLocator); }
```

Ilustración 16. método goToLogin de la clase HomePage

2. Una vez situados en la página LoginPage se ejecuta el método loginUser() de la misma, el cual rellena los campos del *login* localizados mediante sus respectivos *xpath* con los valores especificados en el método. Se usa de nuevo el método *sleep* para poder visualizar los datos introducidos y posteriormente se pulsa el botón de *submit*.

```
public void loginUser() throws InterruptedException {

    sendKeys( inputText: "                          ", emailLocator);
    sendKeys( inputText: "          ", passwordLocator);
    Thread.sleep( millis: 2000);
    click(submitLocator);
}
```

Ilustración 17. método loginUser de la clase LoginPage

3. Por último, se comprueba que se está ubicado en la página de loggedHomePage con su respectivo *assert* y se da por concluido el test.

LogoutUser()

```
@Test
public void logoutUser() throws InterruptedException{

    /* Existe un assert por cada página visitada durante el flujo de la prueba para comprobar
    que la ejecución es correcta. Adicionalmente se controla con mensajes en la consola
    los pasos que se van ejecutando*/

    homePage.goToLogin();
    loginPage.loginUser();
    System.out.println("Login realizado correctamente");
    loggedHomePage.acceptCookies();
    Assert.assertTrue(loggedHomePage.isDisplayed(loggedHomePage.getLoggedHomePageLocator()), s: "No se e
    System.out.println("Redirección a página de Home logado");
    loggedHomePage.logoutUser();
    System.out.println("Logout realizado correctamente");
    Assert.assertTrue(homePage.isDisplayed(homePage.getHomePageLocator()), s: "No se encontró el localiz
    System.out.println("Redirección a página de Home");
    Thread.sleep( millis: 5000);
}
```

Ilustración 18. test logoutUser de la clase LoginTest

Al igual que el método loginUser(), este método se trata de un test como indica su etiqueta de @Test perteneciendo al conjunto de pruebas de la clase que se está comentando, LoginTest.

También cuenta con su misma estructura, llamando a los métodos correspondientes de las clases y verificando por medio de los *asserts* el correcto flujo de ejecución imprimiendo mensajes por la consola.

1. Como se ha mencionado al comienzo del apartado, se ha buscado realizar *pruebas End to End* por lo que para realizar un flujo completo de *logout* independiente de cualquier otra prueba, es necesario realizar un previo *login* en la web. Por esta razón el primer paso de este método consiste en realizar los pasos de *login* indicados en el anterior método.
2. Una vez situados en la página loggedHomePage se ejecuta el método logoutUser() el cual comprueba si es necesario realizar la pulsación a un anuncio que salta de manera aleatoria sobre la web. Tras ello hace uso de dos métodos implementados en la página Base (de la que deriva por lo cual puede hacer uso) los cuales mueven el cursor hacia los elementos de la página indicados, esto es así para proporcionar una ejecución lo más cercana a la realidad posible.

```
public void moveToElement(By locator) {

    Actions actions = new Actions(driver);
    actions.moveToElement(driver.findElement(locator)).perform();
}
```

Ilustración 19. método moveToElement de la clase Base

Por último, realiza pulsa el botón de *logout*.

```
public void logoutUser() throws InterruptedException{

    if (isDisplayed(psd2Locator)) { click(psd2Locator); }
    moveToElement(nameBoxLocator);
    moveToElement(logoutButtonLocator);

    clickHiddenButton(logoutButtonLocator);
}
```

Ilustración 20. método logoutUser de la clase loggedHomePage

3. Por último, se comprueba que se está ubicado en la página de HomePage.

Ejecución

Para la ejecución del conjunto de pruebas visto en este apartado podemos pulsar en el botón de ejecución situado a la izquierda de la declaración de la clase (LoginTest)

Ilustración 21. botón ejecución clase LoginTest

Al lanzar el conjunto de pruebas se observa cómo se abre el navegador con mensaje justo debajo de la barra de navegación (*url*) que nos indica que el navegador está siendo controlado mediante un software automático de pruebas.

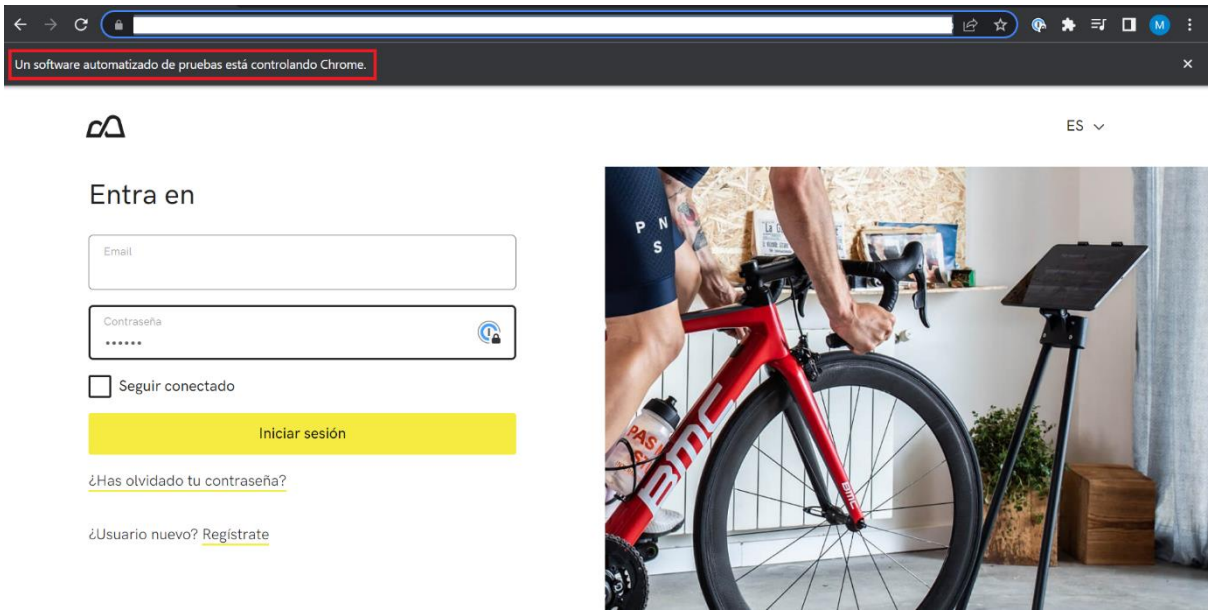

Ilustración 22. Formulario de login durante la ejecución del conjunto de pruebas. Muestra mensaje de software controlando el navegador

Por último, echando un vistazo a la consola del IDE, podemos observar el resumen de toda la ejecución, así como los diferentes mensajes que han impreso durante ella.

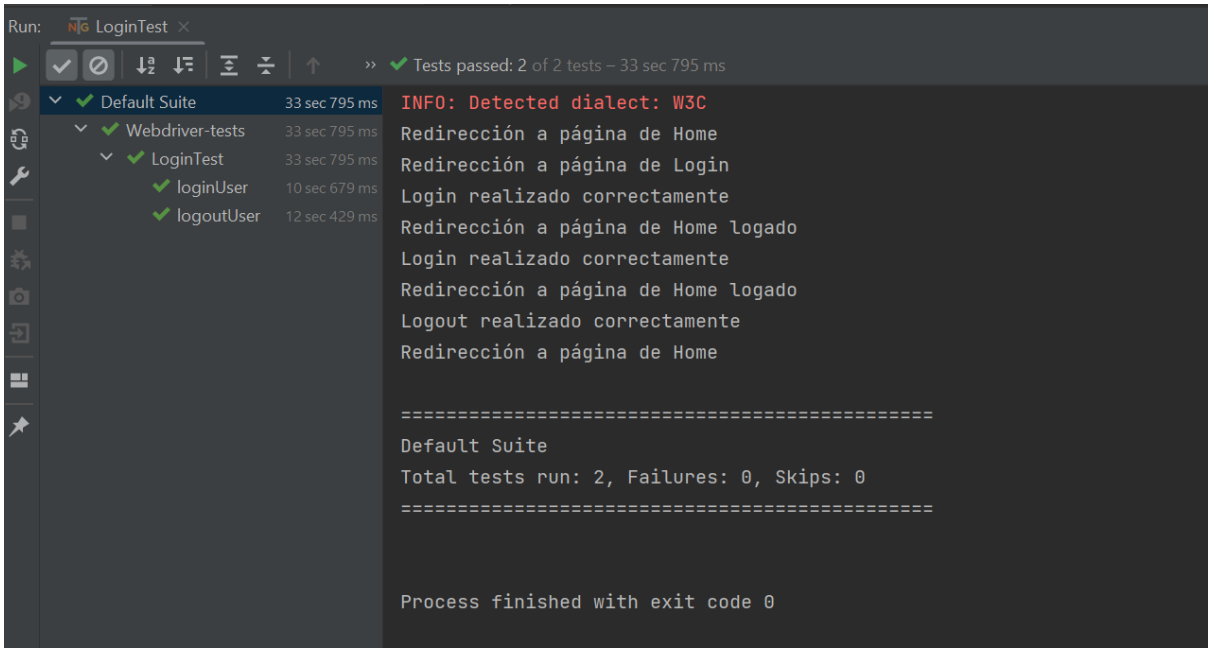

Ilustración 23. Resumen de la ejecución del conjunto de pruebas de LoginTest.java en consola

Como podemos observar en la parte izquierda de la consola, se ha ejecutado el conjunto de pruebas perteneciente a la clase LoginTest el cual consta de dos test. Se marcan con un *tick* verde en representación de una ejecución correcta y sin fallos.

Además de los mensajes imprimidos como control de ejecución vemos un resumen final que indicaría fallos o test que hubieran sido saltados por advertencias o errores en caso de haberlos y que en un futuro con casos de prueba más abundantes serán de gran ayuda para un mayor control.

4.4.2 RegisterTest

De la misma manera que el archivo LoginTest.java, el archivo RegisterTest.java contiene un conjunto de pruebas que consta de dos test a ejecutar de manera individual o de manera consecutiva.

Al igual que en LoginTest tenemos, al comienzo de la clase, la declaración de las propiedades a usar dentro de la misma. En este caso, a parte de las variables de cada tipo de Page y la variable que contiene el objeto WebDriver necesario para la utilización del navegador, encontramos dos nuevas variables de tipo FileManager y KeycloakManager.

```
/* driver Encargado de ejecutar los métodos que controlan el navegador
   fileManager Gestiona los archivos .csv de la carpeta recursos así como
   los usuarios y cupones a usar en los test
   keycloakManager Hace uso de la api de Keycloak para validaciones y otras acciones */
8 usages
private WebDriver driver; //
8 usages
private FileManager fileManager; //
3 usages
private KeycloakManager keycloakManager; //

//Variables tipo Page que permiten comunicar los tests con las diferentes páginas de la web
13 usages
HomePage homePage;
7 usages
RegisterPage registerPage;
3 usages
LoggedHomePage loggedHomePage;
3 usages
EmailConfirmationPage emailConfirmationPage;
7 usages
SubscriptionPage subscriptionPage;
4 usages
CouponPage couponPage;
```

Ilustración 24. propiedades de la clase RegisterTest

El objeto asociado a la variable fileManager será el encargado de comunicarse con los archivos .csv mencionados anteriormente y que contienen los datos sobre usuarios ya registrados en otras ejecuciones y los códigos de los cupones a usar en las pruebas así como la gestión de estos. El objeto keycloakManager por otro lado, será el encargado de comunicarse con la api de Keycloak y realizar las peticiones pertinentes para, por ahora, realizar la verificación del email del usuario a registrar.

Seguido de las propiedades de la clase, encontramos los mismos métodos setUp(), setUpTest() y tearDown() que para el conjunto de pruebas de *login* vistos en el apartado anterior por lo que solo comentaremos el hecho de añadir las páginas extra que se usarán en este conjunto de pruebas y deben ser inicializadas en este primer método, el setUp().

RegisterUser_Free()

```java
public void registerUser_Free() throws InterruptedException{
    /* Existe un assert por cada página visitada durante el flujo de la prueba para comprobar 142
    que la ejecución es correcta. Adicionalmente se controla con mensajes en la consola
    los pasos que se van ejecutando*/

    Assert.assertTrue(homePage.isDisplayed(homePage.getHomePageLocator()), s: "No se encontró el loca
    System.out.println("Redirección a página de Home");
    homePage.goToRegister();
    Assert.assertTrue(registerPage.isDisplayed(registerPage.getRegisterPageLocator()), s: "No se enco
    System.out.println("Redirección a página de Registro");
    registerPage.registerUser(fileManager.getNewUser( subscription: "free"));
    System.out.println("Formulario de registro completado");
    if(keycloakManager.verifymail(fileManager.getActualUser().getEmail())) {

        System.out.println("Email verificado con éxito");
    } else { System.out.println("Error al verificar el email"); }
    emailConfirmationPage.refresh();
    Assert.assertTrue(subscriptionPage.isDisplayed(subscriptionPage.getSubscriptionPageLocator()),
    System.out.println("Redirección a la página de Suscripciones");
    subscriptionPage.getFree();
    System.out.println("Formulario de facturación completado");
    Assert.assertTrue(loggedHomePage.isDisplayed(loggedHomePage.getLoggedHomePageLocator()), s: "No s
    System.out.println("Redirección a página de Home");
    Thread.sleep( millis: 5000);
```

Ilustración 25. test registerUser_Free de la clase RegisterTest

Al igual que sucedía en la clase LoginTest con sus métodos, en esta clase encontramos también la anotación @Test tanto en este método como en el siguiente que veremos ya que se tratan de cada uno de los test del conjunto de pruebas. La estructura del test es exactamente idéntica a las de los otros dos métodos ya explicados.

1. Tras situarse en la página HomePage se ejecutará el método goToRegister(), el cual ejecuta una pulsación sobre el botón de Registro situado en la esquina superior derecha de la página. Este paso será idéntico al ejecutado en los test de *login* pero con el botón de Registro en este caso.
2. Pasará a ejecutarse el método registerUser de la página RegisterPage la cual rellena los campos del formulario de registro con los datos del usuario que se le pase como parámetro. En este caso, se le pasará el resultado del método getNewUser de fileManager el cual crea un nuevo objeto de tipo User generando un nuevo email.

```java
public User getNewUser(String subscription) {

    User newUser = new User();
    newUser.setNumberID(this.nextUserId);
    newUser.setName("WebDriver");
    newUser.setSurname("User");
    newUser.setEmail("[redacted]"+this.nextUserId+"[redacted]"); // Se crea un nuevo email
    newUser.setPassword("[redacted]");
    newUser.setSubscription(subscription);
    newUser.setRegisterDate(getActualDate());
    this.userList.add(newUser); // Se añade el nuevo usuario a la lista de usuarios
    this.actualUser = newUser; // Se establece como el usuario actual
    this.nextUserId += 1; // El código numérico que permite la creación automática de emails
                          // se incrementa en 1
    return newUser;
}
```

Ilustración 26. método getNewUser de la clase fileManager

Dentro del método de registerUser que estábamos explicando encontramos un método scrollDown() el cual es necesario para que el controlador del navegador pueda localizar los siguientes botones a pulsar.

```java
public void registerUser(User newUser) throws InterruptedException {

    // Se rellena el formulario de registro con los datos del objeto
    // usuario pasado como parámetro

    sendKeys(newUser.getName(), nameLocator);
    sendKeys(newUser.getSurname(), surnameLocator);
    sendKeys(newUser.getEmail(), emailLocator);
    sendKeys(newUser.getPassword(), passwordLocator);
    sendKeys(newUser.getPassword(), passwordConfirmLocator);
    scrollDown( i: 200);
    click(termsLocator);
    Thread.sleep( millis: 5000);
    click(submitLocator);
}
```

Ilustración 27. registerUser de la clase RegisterPage

3. El siguiente paso es la verificación del email proporcionado en el registro. El servicio correspondiente manda un email al correo con un código el cual se pide en la siguiente pantalla a mostrar.

Para estas pruebas se han decidido usar peticiones a la api de keycloak (quien proporciona el servicio mencionado) mediante una librería que facilita su uso en proyectos de este tipo. Como ya se ha explicado, esto se realiza en la clase KeycloakManager a la cual, como vemos, se le llama al método verifymail() pasándole el email del usuario que se acababa de registrar y que había quedado almacenado en la variable getActualUser de la clase fileManager.

El método mencionado obtiene una instancia del cliente de administración de keycloak con las credenciales y datos proporcionados dentro del método. Tras ello, se busca el usuario por email, se obtiene el id de dicho usuario y por último, se modifica la propiedad de emailVerified del usuario a true.

```
public boolean verifymail(String userEmail) {

    try (
          // Obtener una instancia del cliente de administración de Keycloak
          Keycloak keycloak = KeycloakBuilder.builder()

                  .serverUrl(this.keycloakUrl)
                  .realm(this.realmName)
                  .clientId(this.clientId)
                  .username(this.username)
                  .password(this.password)
                  .build()) {

        // Obtener el usuario que deseas actualizar por email y posteriormente conseguir su id
        UsersResource usersResource = keycloak.realm(this.realmName).users();
        List<UserRepresentation> users = usersResource.search(userEmail);
        String userId = users.get(0).getId();
        UserRepresentation user = usersResource.get(userId).toRepresentation();

        // Actualizar la propiedad "emailVerified" del usuario
        user.setEmailVerified(true);
        usersResource.get(userId).update(user);
        return true;
    }
```

Ilustración 28. método verifymail de la clase KeycloakManager

Tras confirmar que se ha verificado con éxito el email, se hace un *refresh* de la página para proseguir con el flujo del test y saltar la página en la que, en condiciones normales, habría que proporcionar el código numérico mandado al email.

4. Por último, se lanza el método de getFree() dentro de la página SubscriptionPage. Primero rellena la información de facturación presentada tras verificar el email ejecutando el método fillBillingInformatios_noBill().

```
public void fillBillingInformation_NoBill() throws InterruptedException {

    //Rellena el formulario de información de facturación
    sendKeys( inputText: "Madrid", cityLocator);
    sendKeys( inputText: "Madrid", stateLocator);
    sendKeys( inputText: "[redacted]", postcodeLocator);
    click(countryLocator); //abre el desplegable de paises
    Thread.sleep( millis: 2000);
    click(germanyLocator); //selecciona el pais
    scrollDown( i: 150);
    click(submitLocator);
}
```

Ilustración 29. método fillBillingInformation_NoBill de la clase SubscriptionPage

A continuación, realiza una simple redirección a la página LoggedHomePage pulsando en el logo situado en la esquina superior izquierda.

```
public void getFree() throws InterruptedException {

    //Se llama a la función que rellena la información
    //de facturación
    fillBillingInformation_NoBill();
    Thread.sleep( millis: 2000);
    click(goToHomeLocator); // pulsa el botón que le lleva a Home
    Thread.sleep( millis: 2000);
}
```

Ilustración 30. método getFree de la clase LoggedHomePage

RegisterUser_Coupon()

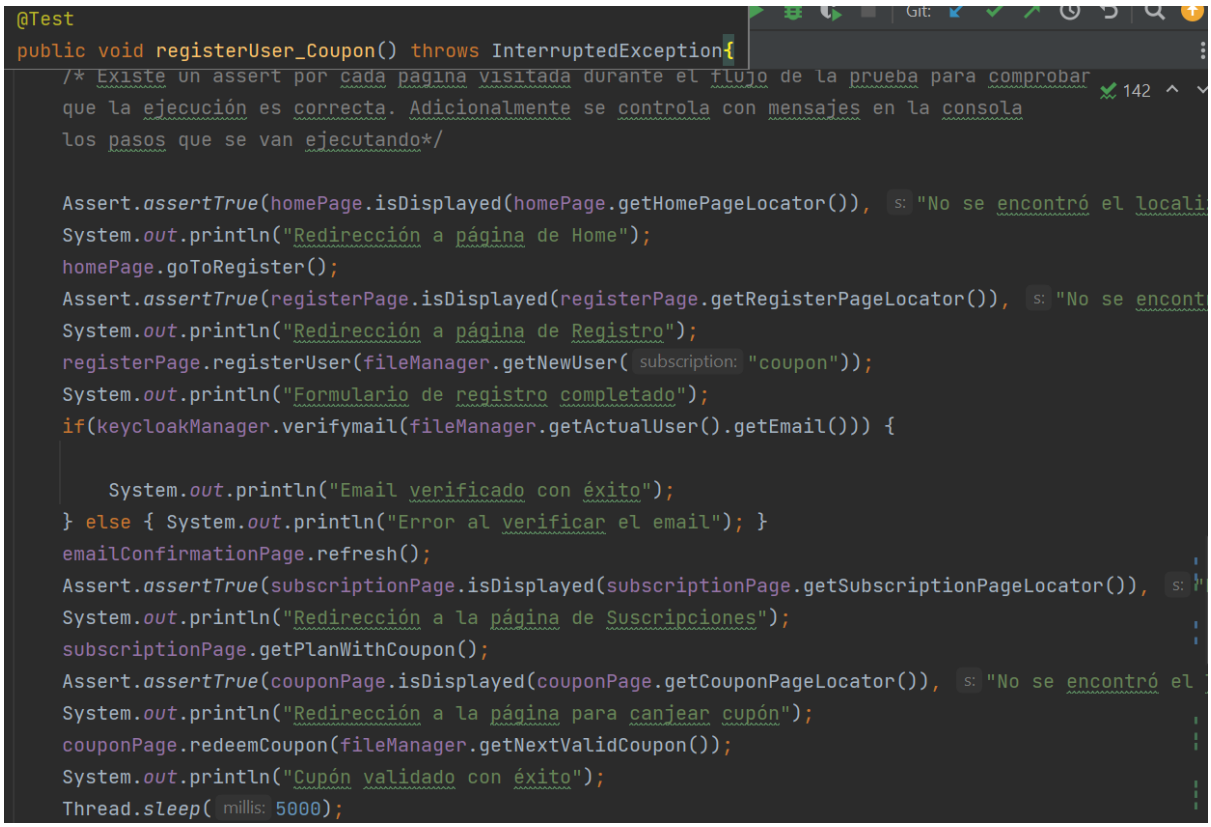

Ilustración 31. test registerUser_Coupon de la clase RegisterTest

Este será el último método anotado con la etiqueta @Test y por lo tanto la segunda prueba perteneciente a el conjunto de pruebas de RegisterTest. Ahora explicaremos los pasos del test con una estructura idéntica a los otros 3 del proyecto.

1. De la misma manera que en el test de registrar un usuario free que acabamos de ver, este test contendrá los primeros 3 pasos mencionados sin modificación a excepción del campo subscription que contiene el usuario registrado cuyo valor en este test será “coupon” en vez de “free”.
2. Lo siguiente que se ejecuta es el método getPlanWithCoupon() de la página SubscriptionPage el cual rellena la información de facturación solicitada como primer paso para contratar un plan al igual que en el test anterior.

   Y a continuación pulsa el botón que lleva a la página de contratación de un plan mediante un cupón.

```java
public void getPlanWithCoupon() throws InterruptedException {

    //Se llama a la función que rellena la información
    //de facturación
    fillBillingInformation_NoBill();
    Thread.sleep( millis: 2000);
    click(couponLocator);
    Thread.sleep( millis: 2000);
}
```

Ilustración 32. método getPlanWithCoupon de la clase SubscriptionPage

3. Por último, se rellena el campo del código del cupón a canjear con la información rescatada de uno de los objetos Coupon almacenados en una variable en el fileManager a quien se le pide un nuevo cupón valido para pasárselo al método encargado de ejecutar dicho paso, el redeemCoupon de la página CouponPage.

```java
public Coupon getNextValidCoupon() {

    //Se crea una variable tipo Coupon que servirá como variable aux
    //para recorrer la lista
    Coupon nextValidCoupon = new Coupon();
    for (Coupon coupon : this.couponList) { //por cada iteración se asigna
                                            // a la variable cada uno de los objetos

        if (!coupon.getIsUsed()) {  //si el Coupon no está usado se devuelve y termina la función

            coupon.setUsed(true);
            nextValidCoupon = coupon;
            return nextValidCoupon;
        } else if (coupon.equals(this.couponList.get(this.couponList.size() - 1))) {

            //Si el último objeto de la lista está usado, se devuelve un mensaje que
            //informa de que se han acabado los cupones promocionales a usar
            System.out.println("No hay códigos válidos para canjear");
        }
```

Ilustración 33. método getNextValidCoupon de la clase FileManager

Tras introducir el código del cupón, se aceptan los términos y se pulsa el botón de confirmación finalizando este último paso y la ejecución del test habiendo contratado una subscripción con un cupón.

```java
public void redeemCoupon(Coupon coupon) throws InterruptedException {

    //Se añade el código del cupón al campo y se pulsa el botón de submit
    sendKeys(coupon.getCode(), couponInputLocator);
    click(submitLocator);
    Thread.sleep( millis: 2000);
    //en la pantalla de confirmación se aceptan los términos y se pulsa aceptar
    click(termsConditionLocator);
    click(redeemCouponButtonLocator);
    Thread.sleep( millis: 2000);
}
```

Ilustración 34. método redeemCoupon de la clase couponPage

Ejecución

Al igual que para la ejecución del conjunto de pruebas de LoginTest, para el conjunto de pruebas de RegisterTest podemos pulsar en el botón de ejecución situado a la izquierda de la declaración de la clase.

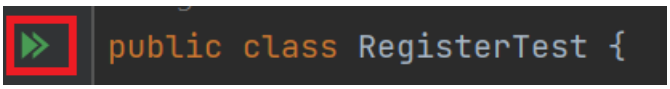

Ilustración 35. botón de ejecución de la clase RegisterTest

También seguimos observando durante toda la ejecución el mensaje de estar siendo controlado el navegador por un software de automatización de pruebas.

Por último, al igual que sucedía con la ejecución de LoginTest, echando un vistazo a la consola del IDE, podemos observar el resumen de toda la ejecución, así como los diferentes mensajes que han impreso durante ella.

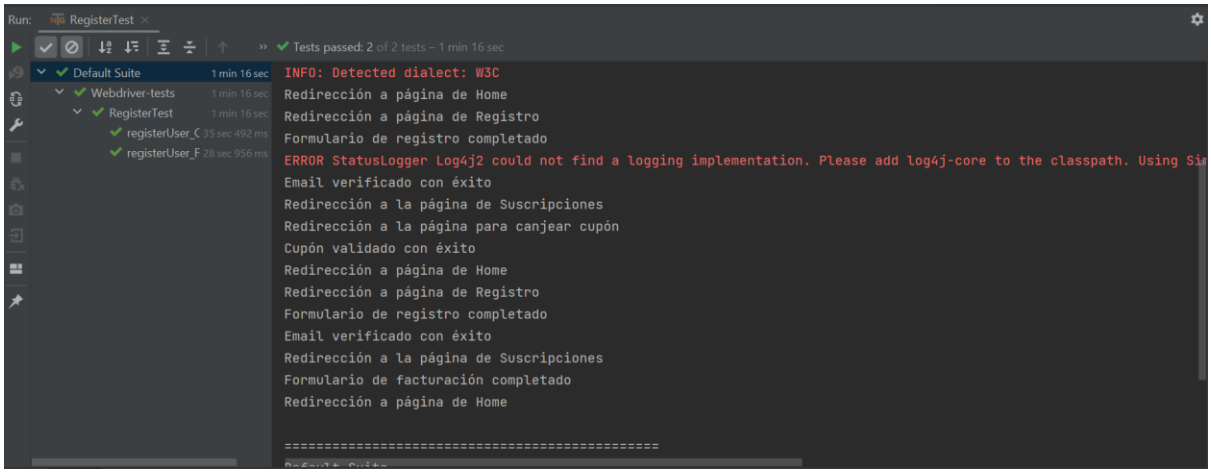

Ilustración 36.Resumen de la ejecución del conjunto de pruebas de LoginTest.java en consola

La estructura del resumen de ejecución es idéntica a la del conjunto de pruebas de LoginTest.

5 Conclusiones y trabajos futuros

En conclusión, el objetivo de este proyecto ha sido mostrar los beneficios de implementar un Plan de Calidad en una empresa dedicada al desarrollo de software tras mostrar la situación inicial de la misma, así como mostrar en detalle el comienzo de un proyecto de automatización de pruebas para la web de esta como parte de dicho Plan de Calidad.

En cuanto a la mejora sufrida gracias a la implantación del Plan de Calidad desde octubre de 2020 hasta la fecha podemos decir que se ha establecido ya un marco de trabajo compatible con la calidad que ve ya los frutos del esfuerzo dedicado a ello. Podemos medir el avance en la empresa desde la implantación gracias a los principales logros que se citan a continuación y que suponen un mayor cambio:

- Política de ramas estándar GitFlow para todos los equipos de desarrollo: Feature, Develop y Master.
- Entorno de trabajo versátiles y dinámicos: Lab, Staging, Beta y Prod.
- Unificación de los sistemas de integración: GitLab CI.
- Implantación de un repositorio oficial para las aplicaciones: AppRepo.

En cuanto al marco metodológico y la gestión de los *sprints* mediante dinámicas ágiles scrum y Kanban:

- Se han establecido un organigrama con roles y responsabilidades definidas cuyo trabajo individual y colectivo se apoya en:
    1. Utilización de historias de usuarios en un formato estandarizado.
    2. Las ceremonias scrum en todos los equipos de forma periódica.
    3. La gestión de los *sprints* y refinamientos.

En cuanto a la estrategia de pruebas, promoción y gestión de las *release*:

- Implementación en todos los equipos de Test Unitarios.
- Todos los productos y servicios tienen en las distintas capas o niveles de cobertura de test en mayor o menor medida, pero especialmente en las partes *core* de cada aplicación o servicio.

Como última prueba de la mejora en el rendimiento de la empresa, se muestran dos gráficas que ayudan a visualizar el resultado de los logros mencionados y dichas mejoras:

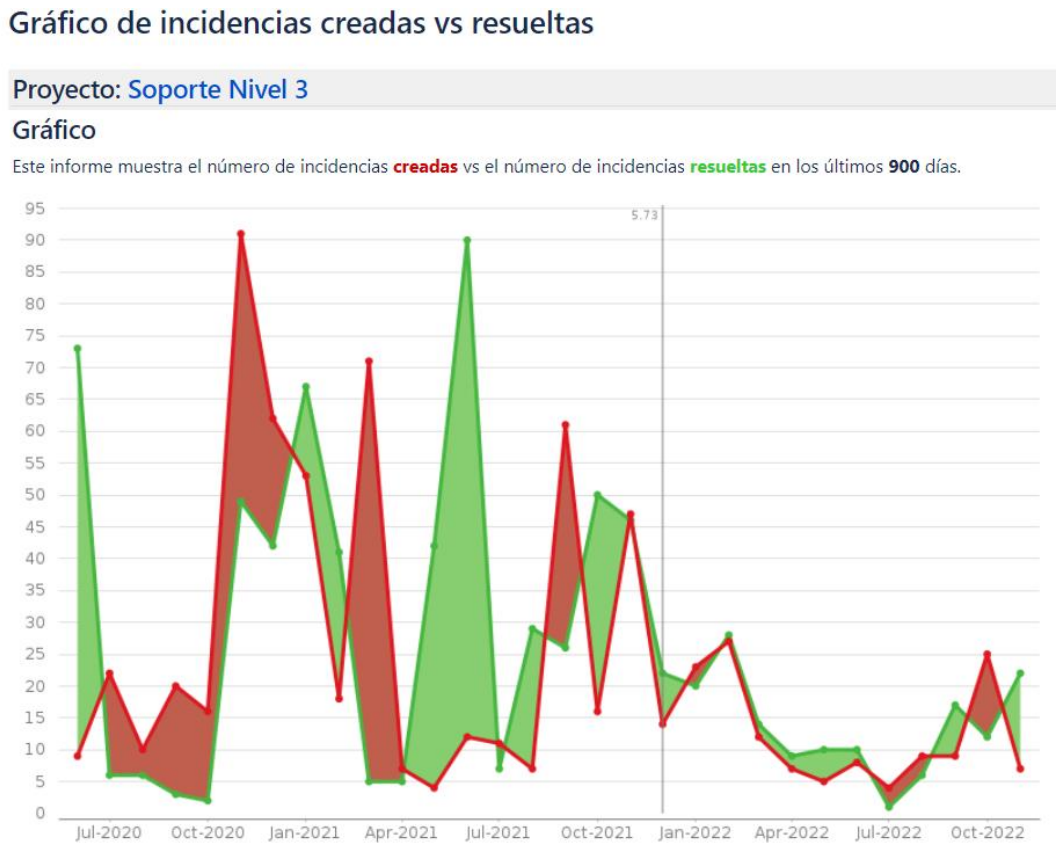

Ilustración 37. Gráfica que compara el número de incidencias creadas vs resueltas desde 2020 hasta 2022.

En la gráfica presentada podemos observar el número de incidencias acumuladas desde el 2020 hasta finales del 2022. En color rojo tenemos la representación de las incidencias creadas frente a las incidencias resueltas en color verde.

Tanto para esta gráfica como para la siguiente hay que tener en cuenta el condicionante que resulta la época del año para el uso y por tanto reporte de incidencias debido al tipo de deporte que es el ciclismo *indoor* ya que en primavera y verano su uso disminuye considerablemente.

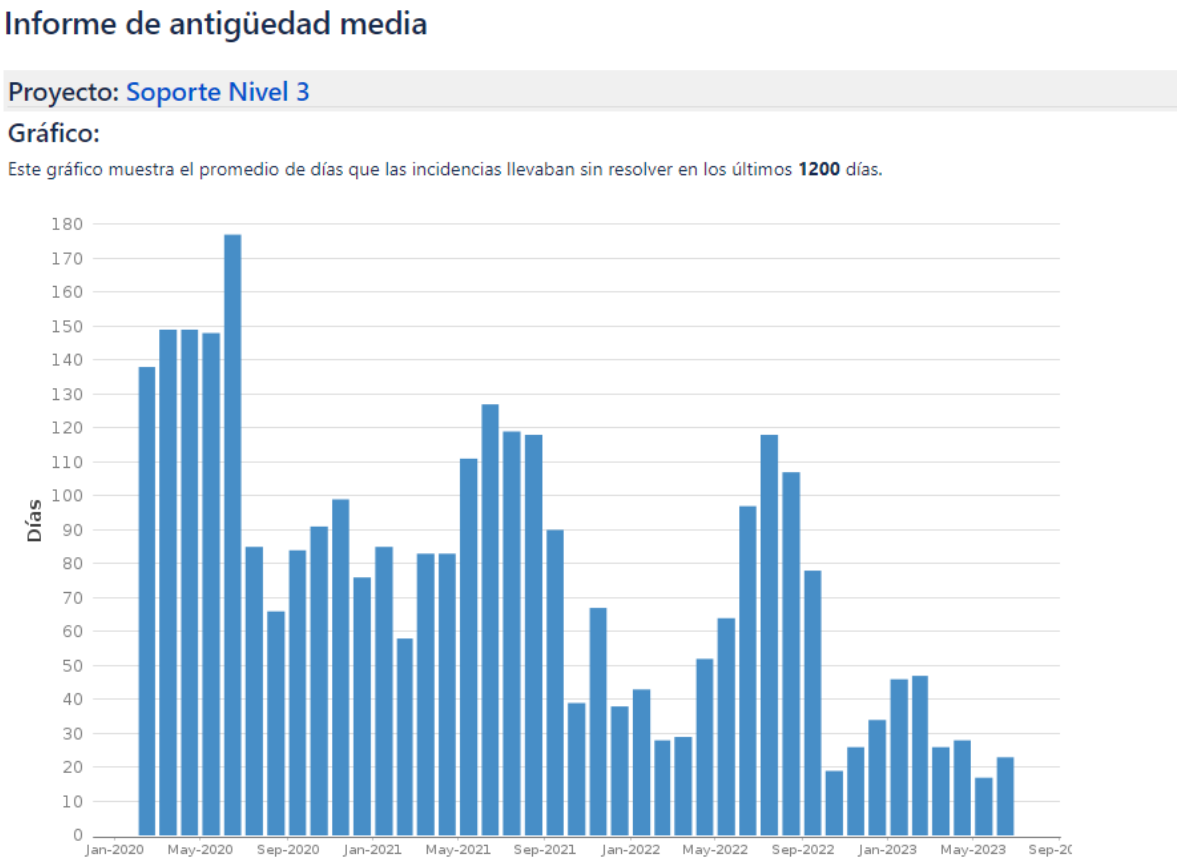

Ilustración 38. Gráfica que muestra el promedio de días que se tarda en resolver una incidencia desde su creación desde 2020 hasta 2023

Al igual que se observa en la anterior gráfica, el progreso durante los últimos años es evidente y permite ver los resultados de una aplicación correcta de las mejoras propuestas durante estos primeros pasos en el Plan de Calidad.

Por otro lado, tenemos la DEMO del proyecto de automatización de pruebas web presentada en este proyecto y que servirá de referencia para la evolución a una automatización completa de las pruebas dedicadas a todas las áreas de desarrollo de la empresa. Gracias al trabajo desarrollado se han establecido criterios importantes a seguir tanto ahora como en un futuro, se han resuelto problemas y dudas que facilitarán los siguientes pasos a seguir y se han sentado las bases en cuanto a la estructura del proyecto que conseguirá un trabajo más eficiente y cómodo por parte de los encargados del proyecto.

En cuanto a esos pasos futuros a seguir dentro del proyecto de automatización de pruebas tenemos:

- Implementación de Log4j como solución a la falta de información tras las ejecuciones permitiendo un mayor seguimiento de errores, advertencias y más información proporcionada gracias a la librería de Log4j.
- Uso de Selenium Grid como herramienta que permita la múltiple ejecución de los test en una red de máquinas conectadas las cuales ejecuten los conjuntos de pruebas en diferentes navegadores y condiciones.
- Extender las pruebas para cubrir todos los flujos de subscripción y contratación de planes dentro de la web.
- Comienzo de investigación de automatización de pruebas dentro del simulador para comenzar la expansión de la automatización de test a más áreas de la empresa una vez cubierta la web.

Como podemos observar, este proyecto tiene un futuro mayúsculo en términos de funcionalidad y aprovechamiento para los productos y servicios de la empresa. Pero lo importante es haber establecido unas bases que aseguren su mantenibilidad y productiva evolución.

6 Bibliografía

[1] Lovelle, J. M. C. (1999). Calidad del Software. de Grupo Gidis Universidad Nacional de la Pamba, Oviedo.

[2] Carrizo, D., & Alfaro, A. (2018). Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico. *Ingeniare. Revista chilena de ingeniería*.

[3] Sommerville, I. (2016). Software Engineering. Pearson.

[4] McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality. The Bell System Technical Journal.

[5] Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing. John Wiley & Sons.

[6] Pressman, R. S., & Maxim, B. R. (2014). Software engineering: A practitioner's approach. McGraw-Hill Education.

[7] Garvin, D. A. (1984). What does "product quality" really mean? Sloan Management Review.

[8] Juran, J. M., & De Feo, J. A. (2010). *Juran's quality handbook: the complete guide to performance excellence*. McGraw-Hill Education.

7. Glosario

- Demo: Abreviatura de "demostración". Se refiere a una presentación o muestra del funcionamiento de un producto o software con el fin de mostrar sus características y capacidades.
- Metodología: Conjunto de métodos, técnicas y procedimientos utilizados para abordar una tarea o alcanzar un objetivo específico. En el contexto del desarrollo de software, se refiere a enfoques sistemáticos y estructurados utilizados para gestionar y realizar proyectos de manera eficiente.
- Metodología ágil: En el desarrollo de software, es un enfoque iterativo e incremental que se centra en la colaboración, la flexibilidad y la respuesta rápida a los cambios. Las metodologías ágiles promueven la entrega continua de software funcional y fomentan la retroalimentación y la adaptabilidad en el proceso de desarrollo.
- *Testing*: También conocido como pruebas de software, es el proceso de evaluar un sistema o programa para verificar su funcionamiento, detectar errores o defectos y asegurar que cumple con los requisitos establecidos.
- Scrum: Es una metodología ágil para la gestión de proyectos de desarrollo de software. Se basa en la colaboración y la autoorganización del equipo de desarrollo en ciclos llamados "*sprints*". Scrum enfatiza la entrega iterativa de incrementos de software funcionales y la adaptabilidad a medida que se obtiene retroalimentación del cliente.
- Kanban: Es un sistema visual de gestión de tareas y trabajo que se utiliza para optimizar el flujo de trabajo en un proyecto. Kanban se basa en tarjetas o "post-its" que representan tareas y se mueven a través de etapas o columnas para indicar su estado y progreso.
- API (Interfaz de Programación de Aplicaciones): Conjunto de reglas y protocolos que permiten a diferentes sistemas y aplicaciones comunicarse y compartir información entre sí.
- *Backoffice*: Es la parte de un sistema o aplicación que se ocupa de las funciones y tareas administrativas y de gestión internas de una empresa. Por lo general, no es visible para los usuarios finales.
- *Firmware*: Es un software embebido en un dispositivo electrónico o hardware que controla su funcionamiento y proporciona las funcionalidades básicas del dispositivo.

- *Roadmap*: Es una representación visual o plan que muestra la dirección y las metas futuras de un proyecto, producto o empresa.
- Ciclos de producto *delivery*: Se refiere a las fases o etapas en el desarrollo y entrega de un producto, desde su concepción hasta su lanzamiento al mercado.
- *Time to market*: Es el tiempo que transcurre desde el inicio del desarrollo de un producto hasta que está disponible para su comercialización o lanzamiento al mercado.
- Sistemas de CI (Integración Continua): Son herramientas y prácticas que permiten a los desarrolladores integrar y probar su código de manera frecuente y automática para detectar errores y problemas de compatibilidad de forma temprana en el proceso de desarrollo.
- Ceremonias Scrum: Son reuniones y eventos regulares dentro del marco de trabajo Scrum, como la planificación del sprint, la revisión del sprint y la retrospectiva del sprint, que permiten al equipo de desarrollo colaborar, revisar el progreso y mejorar continuamente.
- *Sprints*: En Scrum, son ciclos de tiempo cortos y fijos en los que se desarrolla, prueba y entrega un incremento de software funcional. Los *sprints* suelen tener una duración de 1 a 4 semanas.
- *Gitflow*: Es un flujo de trabajo de control de versiones basado en el sistema de control de versiones Git. Define ramas y reglas claras para el desarrollo, la integración y el despliegue de código en proyectos colaborativos.
- *Backend*: Se refiere a la parte del software o sistema que se ocupa del procesamiento y almacenamiento de datos, así como de la lógica de negocio. El *backend* suele estar oculto para los usuarios finales y se comunica con el *frontend* o la interfaz de usuario.
- Entorno: Se refiere al conjunto de condiciones y configuraciones en las que se ejecuta un software o sistema. Puede incluir el hardware, el sistema operativo, las bibliotecas y otros componentes necesarios para su funcionamiento.
- *Pipelines*: En el desarrollo de software, se refiere a la automatización del flujo de trabajo, donde las tareas y procesos se organizan y ejecutan en secuencia, permitiendo una entrega continua y rápida de software.
- *Framework*: Es un conjunto de herramientas, bibliotecas y pautas que proporcionan una estructura y abstracciones para el desarrollo de software. Los *frameworks* facilitan la creación de aplicaciones al proporcionar funcionalidades predefinidas y reutilizables.