Agile Automation Framework

Victor Parera Pastor

Resumen— Realizar pruebas sobre un nuevo desarrollo de software se ha convertido en algo esencial para medir la calidad del código y asegurar que los cambios introducidos se comportan exactamente como se espera que lo hagan. El problema que se presenta al aplicar este control de calidad es que puede actuar como un bloqueante para el trabajo de los desarrolladores, los cuales tienen que esperar a que se realicen estas pruebas para saber si la nueva implementación es viable o necesita ser modificada para ajustarse al comportamiento esperado. Este tipo de controles, generalmente, se hacen de forma manual, interactuando con la interfaz tal y como un usuario lo haría para detectar los posibles fallos que se producen fruto de los nuevos desarrollos introducidos, lo cual implica una cantidad de tiempo de espera muy elevada. Esta problemática es la que da origen a este proyecto, donde se desarrollará un framework de pruebas automáticas que nos permita reproducir las acciones de un usuario de forma automática para ahorrar los costes en tiempo tanto para el desarrollador como para aquél que se encarga de verificar la calidad del desarrollo.

Palabras clave—Aplicación Web, Java, Eclipse, Selenium, Pruebas automáticas, BrowserStack, Gherkin, Jenkins, Git, Control de Calidad, Desarrollo

Abstract—Testing a new software development has become essential to measure the quality of the code and ensure that the changes made behave exactly as they are expected to. The problem with applying this quality control is that it can act as a blocker to the work of developers, who have to wait for these tests to be carried out to know if the new implementation is viable or needs to be modified to adjust to expected behavior. This type of controls, generally, are done manually, interacting with the interface just as a user would to detect possible failures that occur as a result of the new developments introduced, which implies a very high amount of waiting time. This problem is what gives rise to this project, where an automatic testing framework will be developed that allows us to reproduce the actions of a user automatically to save time costs for both the developer and the one who is responsible for verifying the development quality.

Index Terms— Web Application, Java, Eclipse, Selenium, Automated Testing, BrowserStack, Gherkin, Jenkins, Git, Quality Assurance, Development

1 INTRODUCCIÓN

ACTUALMENTE en la empresa de moda *Mango*, se produce un despliegue por semana de una nueva versión de la página web que se encuentra en producción. Esta nueva versión incluye cambios y mejoras sobre el código de la anterior, con lo cual es necesario realizar pruebas funcionales para verificar que todo aquello que ya funcionaba, sigue funcionando correctamente antes de que el nuevo código esté en un entorno productivo. Para ello, existe un proyecto escrito en el lenguaje *Java*[1] que se apoya en las librerías de *Selenium*[2] y *TestNG*[3], a través del cual se ejecutan una serie de tests automáticos sobre la interfaz de usuario de la página web de *Mango* después de cada nuevo despliegue en el entorno de producción. Una vez todo ha sido validado, se incluyen nuevos tests referentes a este nuevo código, de manera que quede cubierto para verificar que sigue funcionando tras el siguiente despliegue. Sin embargo, hay otras herramientas que no están integradas y que ayudarían a tener un desarrollo más robusto del proyecto y a lograr una buena integración entre negocio y el programa.

Para este último punto, se incluirá un lenguaje comprensible tanto para ordenadores como para humanos llamado *Gherkin*[4], que será de ayuda para establecer una forma común de comunicación entre el departamento de negocio y el de desarrollo. Por otro lado, se usará la librería *Cucumber*[5] para poder escribir las pruebas relacionándolas con el archivo del tipo *".feature"* donde estará escrito el escenario a testear por parte de negocio. Por último, se ejecutarán estas pruebas desde un servidor de integración continua llamado *Jenkins*[6], que será de ayuda para programar en el tiempo la ejecución del *framework*[7], además de permitir la parametrización de las pruebas por parte de los técnicos de calidad.

- *E-mail de contacto: victor.parera@e-campus.uab.cat*
- *Mención realizada: Tecnologías de la Información.*
- *Trabajo tutorizado por: Ana Oropesa Física*
- *Curso 2019/20*

2 ESTADO DEL ARTE

Los *frameworks* de pruebas automáticas son una herramienta diseñada para facilitar y agilizar la tarea de los desarrolladores y técnicos de calidad en cuanto a probar que todos los cambios y nuevos desarrollos de código no afectan a la versión de la aplicación o página web que ya se encuentra en el entorno de producción.

Estos *frameworks* pueden ser desarrollados en una gran variedad de lenguajes, siendo los más principales *Java*, *Python*[8] y *C++*[9]. En este proyecto y basándonos en las directrices de la empresa para la que va a ser desarrollado, se ha elegido el lenguaje *Java*.

Hoy en día, la mayoría de técnicos de calidad y desarrolladores usan *frameworks* de pruebas automáticas para poder probar sus desarrollos antes de desplegar su nuevo código en un entorno productivo. Además, la facilidad de uso de éstos ayuda a que no sea una tarea tediosa en la que se deba dedicar un gran número de horas.

Para la ejecución de estos *frameworks* es recomendable hacer uso de herramientas de integración continua, las cuales nos permiten registrar información sobre las diferentes ejecuciones que se han ido generando a lo largo del tiempo. Para este fin, se ha acordado con la empresa *Mango* usar la herramienta llamada *Jenkins*.

3 OBJETIVOS

Tal y como se ha podido observar en los apartados anteriores, el objetivo final de este trabajo es poder validar que después de cada nuevo despliegue de código en la web de *Mango*, los principales casos de uso siguen teniendo el funcionamiento esperado. Para lograrlo, se ha definido una serie de objetivos que se deben cumplir, los cuales se exponen a continuación:

- Identificar el lenguaje de programación que se va a usar para escribir el *framework* de pruebas automáticas a través de los documentos de convenio de *Mango* donde se indica en qué lenguaje se debe desarrollar, para obtener información sobre cuál se usa para escribir el código de la página web de *Mango*, de manera que se pueda empezar a programar las pruebas automáticas durante los próximos tres meses en este mismo lenguaje.

- Analizar el comportamiento de los usuarios de la página web de *Mango* a través de los informes del equipo de analítica para obtener los diez casos de uso principales de manera que se puedan añadir al código del proyecto antes de los próximos tres meses.

- Desarrollar el código del *framework* de automatización de pruebas a través de una plataforma de desarrollo de código abierto antes de los próximos tres meses para comprobar que al ejecutar las pruebas automáticas se verifica que los diez casos de uso principales siguen funcionando correctamente, y que en caso de que no sea así, informe de ello para actuar con la mayor brevedad posible.

- Analizar el ahorro de tiempo que supone el hecho de tener este *framework* activo respecto a cuando no lo estaba, a través de los informes que especifican cuánto tiempo se empleaba en reproducir los diez casos de uso principales para verificar su correcto funcionamiento después de un nuevo despliegue de la página web de *Mango*. De esta manera poder demostrar la efectividad y ahorro que supone este proyecto antes del día 14 de junio de 2020.

La prioridad es que estos tests se puedan ejecutar en los navegadores con mayor cuota de uso en el mercado[10] , Google Chrome y Mozilla Firefox. Además, ambos deberán ejecutarse en el sistema operativo más popular[11]: Windows 10.

Queda fuera del alcance de este proyecto el hecho de automatizar pruebas para la totalidad de la página web de *Mango*.

4 DESARROLLO

Siguiendo la planificación inicial del proyecto, se empezó creando un repositorio en la plataforma de desarrollo colaborativo de software conocida como *Bitbucket*[12] con la finalidad de alojar todos los ficheros relativos al *framework* que se quiere crear usando el sistema de control de versiones *Git*[13].

Para ello lo primero que se ha hecho ha sido dar el nombre de *Agile Automation Framework* al repositorio creado. Después se ha creado una estructura general del proyecto, con los elementos necesarios para empezar a desarrollarlo. Esta estructura consta una carpeta principal y dos archivos.

La carpeta, se ha nombrado *src* y será donde almacenaremos todo el código del proyecto.

Por otro lado, se ha creado un archivo de configuración de la herramienta *Jenkins* que consiste en un fichero de texto que contiene la definición de una tarea del tipo *tubería*.

Por último, se ha creado el archivo *pom.xml*, que contendrá todas las dependencias del *framework* que crearemos.

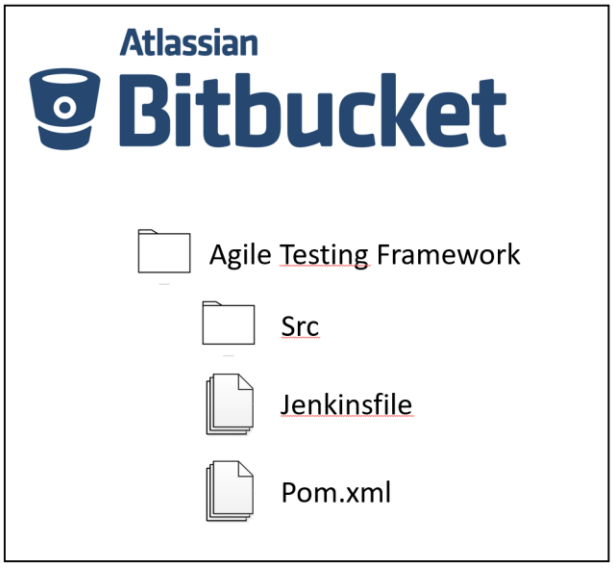

Figura 1. Estructura del proyecto.

A continuación, se ha creado un proyecto del tipo *Maven*[14] en el entorno de desarrollo integrado *IntelliJ*[15], y se han añadido las librerías de *Selenium*, *TestNG* y *Cucumber* al archivo *pom.xml*[16] del proyecto para que actúen como una dependencia de éste.

Seguidamente se ha hecho el primer *commit*[17] al repositorio creado en *Bitbucket*.

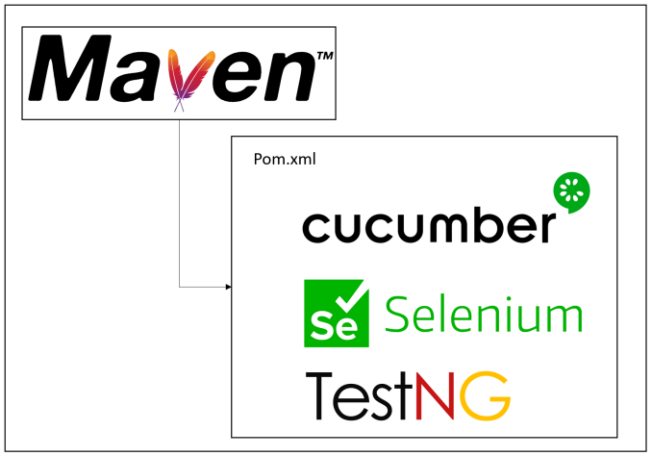

Figura 2. Maven y dependencias del proyecto.

Después, se ha escrito la estructura del proyecto en el lenguaje de programación *Java* ya que así está definido en los documentos de convenio de desarrollo de la compañía *Mango*.

Ésta, consta seis paquetes. El primero, llamado *Base*, contiene una clase *Java* con un objeto del tipo *WebDriver*[18], que usaremos para extender las clases que definiremos más adelante sin tener que definir un objeto de este tipo para cada una de ellas.

El segundo paquete que se ha creado ha sido el que recibe el nombre de *Driver y* contiene una clase escrita en lenguaje *Java* llamada *DriverFactory* que actúa como una factoría y devuelve un objeto del tipo *WebDriver* con la información que definirá el navegador web donde se ejecutarán las pruebas automáticas.

El que ocupa la tercera posición, ha sido llamado *Features*, y contiene las definiciones en lenguaje *Gherkin* de cada una de las pruebas automáticas que serán implementadas en el siguiente paquete en archivos con la extensión "*.feature*". Además, en estos archivos se han usado las anotaciones que nos proporciona *TestNG* para poder organizar la ejecución de las pruebas.

En cuarto lugar, se ha creado el paquete llamado *Steps*, en el cual se han implementado las pruebas descritas en el paso anterior con lenguaje *Java*.

Para llevar a cabo un desarrollo que use patrones se ha escrito el código siguiendo el patrón *PageObject*[19] y *PageFactory*[20].

Para contener las clases *Java* que resultan del uso de estos patrones, se ha creado el quinto paquete con el nombre *Pages.*

Por último, se ha creado el paquete llamado *Runner*, que nos facilitará la ejecución de las pruebas, centralizando la configuración necesaria para el lanzamiento de las pruebas automáticas.

Cabe destacar que se cuenta con dos archivos de tipo *xml*[21]. El primero es el archivo *pom.xml* del que se ha hablado anteriormente y que sirve para definir las dependencias del proyecto. El segundo es *testng.xml*[22] dónde se debe definir qué clase se encargará de la ejecución de las pruebas automáticas y qué otra hará la función de escuchar constantemente los posibles eventos que se generen durante la ejecución de las pruebas.

Esta estructura es lo que se conoce como *framework*, es decir, es el esquema que se usará para el desarrollo de las pruebas automáticas. Se puede observar en la figura 3.

Figura 3. Estructura interna del proyecto.

Desde este momento para añadir nuevas pruebas que verifiquen nuevos flujos, se deberá incluir el archivo *".feature"* correspondiente en la carpeta *Features* y automáticamente se deberá generar un archivo asociado a el anterior donde se definan todos los pasos escritos en el escenario del primer archivo.

Se han escrito, después de la implementación de esta estructura, dos pruebas automáticas a modo de ejemplo.

Se han probado también el resto de funcionalidades a desarrollar antes de la entrega del primer informe de progreso, como son la paralelización de los tests y el lanzamiento de estos desde el archivo *pom.xml* con un resultado favorable.

A continuación, se ha usado un archivo de texto del tipo *Jenkinsfile*[23] para definir las diferentes etapas que debe tener el flujo de integración contínua en la herramienta *Jenkins*. Este fichero ha sido almacenado junto con el código del proyecto.

Para este proyecto se han definido dos etapas: la primera de ellas consiste en comprobar que podemos hacer uso de *Maven* para la construcción del proyecto, la segunda se basa en la ejecución de nuestro código, tal y como se puede observar en la figura 4.

Figura 4. Definición de etapas del archivo jenkinsfile.

Para poder hacer uso del archivo anteriormente mencionado, ha sido necesario crear una nueva tarea en *Jenkins*, asignándole el tipo *Multibranch pipeline* debido a que el repositorio *git* contiene dos ramas distintas: *master* y *develop*.

Una vez se ha creado la tarea, se ha indicado dentro de ésta, en que repositorio *git* se encuentra el código de nuestro proyecto y el archivo *Jenkinsfile*.

De esta manera, la tarea de *Jenkins* ha podido escanear el contenido de estas dos ramas y ejecutar las etapas descritas en el archivo de definición, tal y como se puede observar en la figura 5.

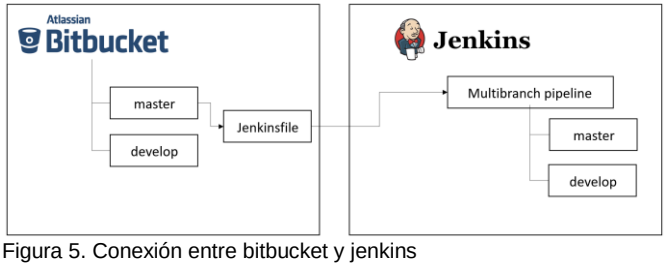

Figura 5. Conexión entre bitbucket y jenkins

Para completar esta parte del desarrollo, se ha ejecutado la tarea anteriormente mencionada y se ha observado cómo ha seguido una a una las etapas descritas en el fichero *Jenkinsfile*.

Paralelamente, se ha creado una nueva tarea en *Jenkins* para facilitar la parametrización de las pruebas por parte de los técnicos de calidad y desarrolladores.

Ésta les permite hacer una selección que se adapte de una forma más exacta a sus necesidades, en el sentido de que pueden ejecutar sólo aquellas pruebas que más les interesen, en lugar de tener que esperar a la ejecución de todo el código.

La parametrización ha sido posible gracias a las anotaciones que nos permite incorporar en los tests el *framework* conocido como *TestNG* y gracias a las variables de entorno que nos permite utilizar el lenguaje *Java*.

Los distintos parámetros que se han habilitado para hacer un filtrado más exacto de las pruebas automáticas han sido los que se pueden observar en la figura 6.

Figura 6. Parametrización de la ejecución del framework de pruebas automáticas

Dentro de cada uno de éstos, encontramos las distintas opciones que permiten llevar a cabo la ejecución de aquellas pruebas que se quieran realizar de una forma más precisa.

Más adelante, se han extraído los diez comportamientos que verifican que tanto los modales de captación de clientas como los flujos más comunes que siguen dentro de la página web de Mango, funcionan correctamente. Estos diez casos pueden encontrarse en el Anexo I.

Para decidir cuáles serían estos diez flujos, se ha trabajado con el departamento de analítica, identificando paso por paso cómo se comportan las clientas desde su entrada en la página web hasta que deciden abandonarla.

Posteriormente, se han escrito uno por uno en un archivo con la extensión "*.feature*" diferente. Este tipo de archivo permite un lenguaje comprensible tanto por máquinas como por personas, de manera que ha facilitado mucho la definición de las pruebas a automatizar y la implementación de éstas.

Para la implementación de los tests, ha sido necesario identificar cada uno de los elementos con los que necesitábamos interactuar y extraer sus identificadores a través de la estructura de objetos que genera el navegador, tal y como se muestra en la figura 7.

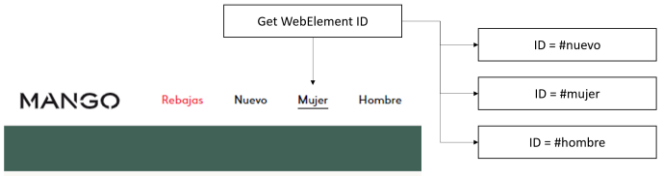

Figura 7. Identificación de elementos web.

Una vez se han implementado los diez casos de uso anteriormente descritos, se ha probado que desde la herramienta *Jenkins* se hubieran ejecutado correctamente.

Posteriormente, se ha habilitado la opción de la librería *Cucumber* que permite crear un archivo del tipo *JSON* con los resultados de la ejecución.

Esta acción combinada con la integración de un plugin llamado *Cucumber Reports*[24] que se encuentra dentro del mercado de extensiones de *Jenkins*, permite la creación de un informe con los resultados de la ejecución de las pruebas automáticas.

Este plugin, tal y como se puede observar en la figura 8, toma como archivo de entrada el generado al final de las pruebas con la extensión "*.json*" y lo convierte en un archivo del tipo "*.html*" el cual facilita la lectura de los resultados por parte del usuario final.

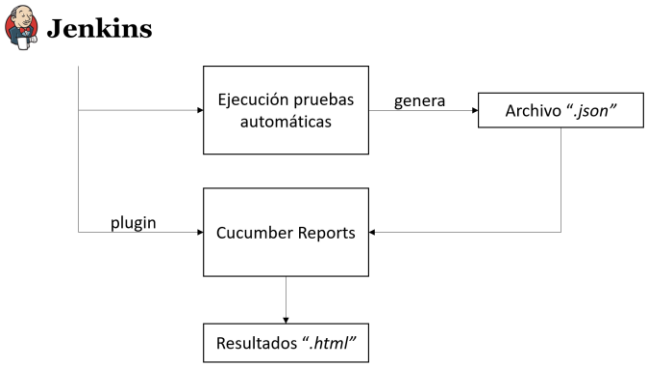

Figura 8. Generación del informe de resultados con Jenkins

Para verificar que todo funciona según lo esperado, se ha ejecutado una prueba automática dentro de la página de *Mango*, en el entorno de producción, que consiste en verificar que si se selecciona como país Francia, los dos botones que aparecen en un banner del apartado *Mango Likes You* tienen los atributos *html*[25] conocidos como *href*[26] con los valores correctos.

Para ello, se ha usado la tarea creada en la herramienta de integración contínua *Jenkins*, seleccionando como parámetros los que se pueden observar en la figura 9 y pulsando el botón de *build* a continuación.

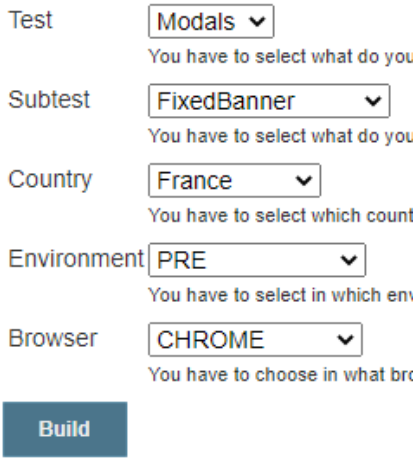

Figura 9. Parametrización de pruebas automáticas con Jenkins

Una vez esta tarea se ha ejecutado correctamente, podemos dirigirnos a *BrowserStack*[27] para visualizar el video que se habrá creado durante la ejecución, del cual se han hecho las capturas que se pueden observar en las figuras 10, 11 y 12.

Se puede afirmar que todas las capturas han sido tomadas durante la ejecución de las pruebas automáticas por la ventana emergente que aparece en google chrome con el siguiente mensaje: *Chrome is being controlled by automated test software*

En primer lugar, se ha seleccionado como país Francia y se ha pulsado el botón *enter*.

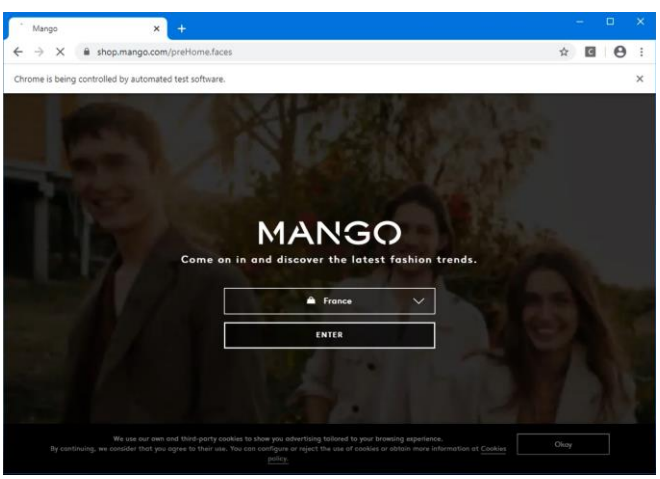

Figura 10. Reproducción de prueba automática con BrowserStack. Selección de idioma.

En segundo lugar, se ha comprobado que se nos ha redirigido a la página principal de la tienda online de *Mango* utilizando como idioma el francés.

A continuación, se ha hecho una redirección hacia la página principal del apartado *Mango Likes You*.

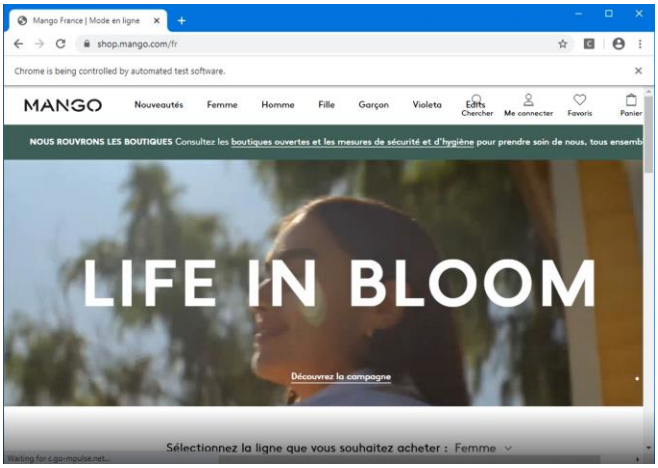

Figura 11. Reproducción de prueba automática con BrowserStack. Shop landing page en francés.

Por último, se ha comprobado que la página donde nos encontramos contenga un *modal* que responda al nombre de *FixedBanner*.

Dentro de éste, se ha comprobado que existan dos botones: "*Rejoindre le Club*" y "*Ouvrir session*".

Una vez verificado este paso, se ha procedido a extraer el valor del atributo *html* conocido como *href* de ambos botones, para comprobar que tienen el valor que deben tener.

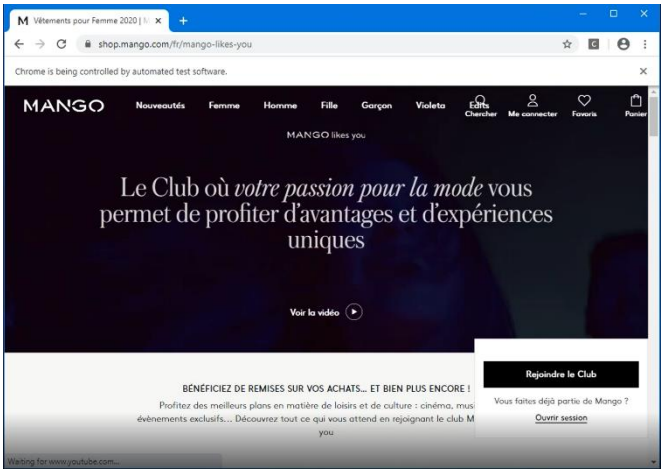

Figura 12. Reproducción de prueba automática con BrowserStack. Mango Likes You landing page en francés.

5 ARQUITECTURA

Para desarrollar este *framework* se han hecho servir varias tecnologías que se han tenido que integrar entre sí para dar lugar a un marco de trabajo robusto y fiable.

En primer lugar, se ha tenido que usar *Bitbucket* como plataforma de control de versiones. En él se ha almacenado la totalidad del código del proyecto de *Agile Testing Framework*.

En segundo lugar, se ha tenido que hacer uso de *Jenkins* como software de integración contínua.

En tercer lugar, se ha construido el proyecto a través de la herramienta *Maven*.

Por último, se ha hecho uso del plugin *Cucumber Reports* que ofrece la herramienta Jenkins para facilitar la lectura del informe de los resultados al usuario, usando un archivo "*.html*".

Las integraciones entre todas estas tecnologías se han llevado a cabo a través del software de integración continua *Jenkins*, ya que desde él se recupera el código que se encuentra en *Bitbucket*, se parametriza el comando a ejecutar con *Maven* y se genera el archivo "*.html*" con los resultados de la ejecución de las pruebas automáticas.

Esta arquitectura puede verse reflejada gráficamente en la figura 13.

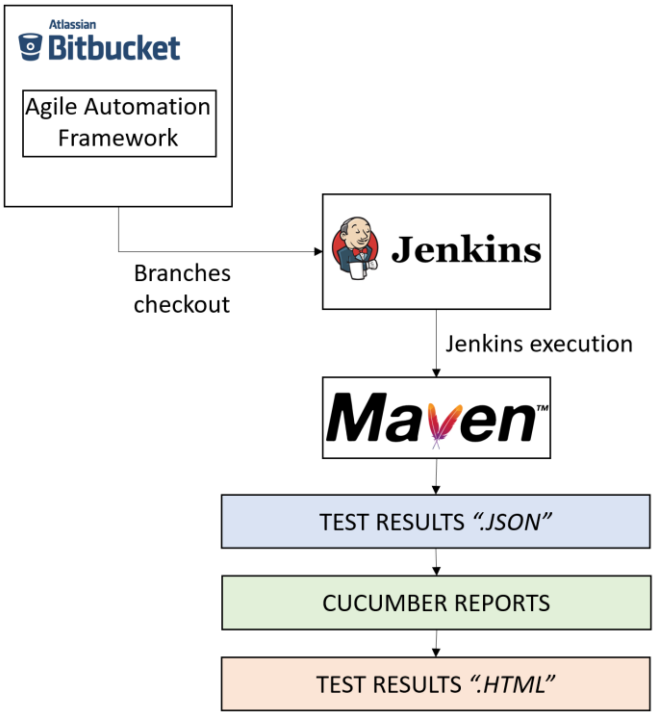

Figura 13. Arquitectura del proyecto

6 METODOLOGÍA

Para el desarrollo de este proyecto se ha hecho uso de una metodología ágil conocida como *Kanban*[28] combinada con un modelo de desarrollo en espiral.

Esto nos permite controlar el flujo de trabajo y flexibilizar la entrada de nuevas tareas que no estaban previstas en un principio, así como su seguimiento y priorización dentro de la planificación que se había elaborado al inicio del proyecto.

Para aplicar el modelo en espiral nos apoyaremos en círculos crecientes de cuatro fases que consistirán en: planificación, análisis de riesgo, implementación y evaluación.

Este esquema se puede observar en la figura 14.

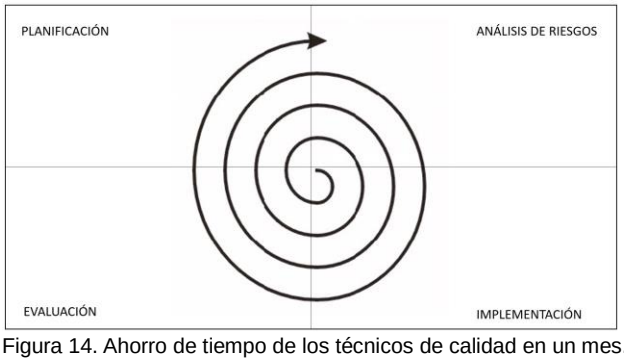

Figura 14. Ahorro de tiempo de los técnicos de calidad en un mes.

7 RESULTADOS

Gracias a estas pruebas automáticas se ha conseguido reducir el número de horas que el equipo de calidad debía dedicar a hacer las comprobaciones pertinentes antes de desplegar una nueva versión de la página web de *Mango* en el entorno de producción.

El ahorro en cuanto al tiempo se refiere, ha sido cuantificado teniendo en cuenta que había que realizar cincuenta y cuatro flujos distintos en la página web del entorno de pre-producción para verificar que el nuevo desarrollo funcionaba tal y como se esperaba y que nada de lo que ya estaba en producción se había visto afectado.

Se ha estimado que la media de tiempo de un técnico en realizar cada una de estas pruebas de forma manual era de cinco minutos, de manera que para completar los cincuenta y cuatro tests, se hubieran necesitado dos cientos setenta minutos, lo cual corresponde a cuatro horas y media.

En este último mes en que el *framework* de pruebas automáticas ha estado en funcionamiento, se han podido ejecutar cuarenta pruebas automáticas, diez por semana.

Teniendo en cuenta que la dedicación de tiempo a las pruebas manuales de un técnico de calidad antes de existir este *framework* era del cien por cien, podemos afirmar que, desde este momento, disponen de aproximadamente un 15.6% más de tiempo, el cual se ha podido emplear para añadir nuevos flujos de revisión de la página web, lo cual ha permitido reducir la tasa de errores introducidos en la versión de producción y por consiguiente, mejorar la experiencia de las usuarias dentro de la tienda online de *Mango*. Esta comparativa se puede observar en la figura 15.

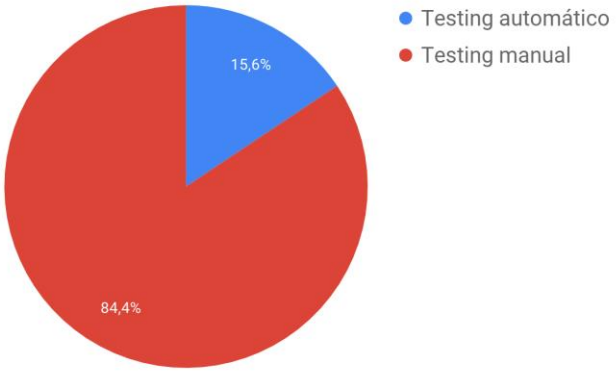

Figura 15. Ahorro de tiempo de los técnicos de calidad en un mes.

Se estima que, en un futuro, este *framework* conseguirá automatizar el suficiente número de pruebas como para reducir en un cuarenta por ciento el trabajo de un técnico de calidad en cuanto a las pruebas manuales se refiere.

Por otra parte, la ejecución de estas pruebas automáticas ha supuesto la detección de una serie de errores en el último despliegue que estaba previsto hacer en el entorno de producción, lo cual ha permitido desestimar la subida del nuevo código al entorno productivo para que los desarrolladores puedan arreglar todo aquello que ha fallado. De esta manera se ha conseguido reducir a cero el impacto que iban a tener estos desarrollos en la venta de producto en línea.

De cuatro despliegues mensuales que se realizan, se ha conseguido detectar errores que suponían un impacto alto en uno de ellos, tal y como se puede observar en la figura 16.

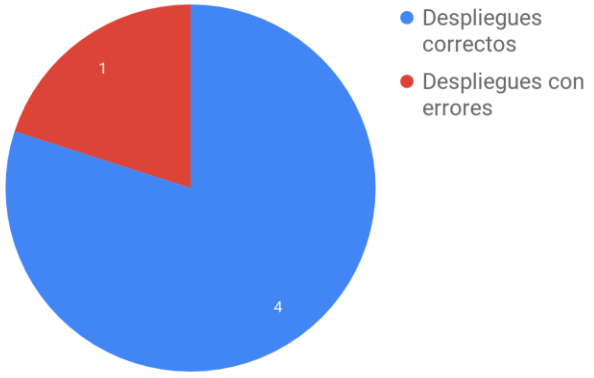

Figura 16. Comparativa de despliegues correctos y con errores en un mes.

8 CONCLUSIONES

Para concluir este proyecto, se puede decir que se ha conseguido crear un marco sobre el que definir cualquier prueba relativa a un navegador y automatizarla de una manera fácil y entendible tanto por la parte de las máquinas como por la parte de las personas.

Con lo cual, podemos afirmar que este nuevo *framework* de pruebas automáticas ahorrará mucho tiempo a aquellos técnicos de calidad y desarrolladores de *Mango* que prueban que sus implementaciones funcionen bien de forma manual, y a todos aquellos que usan el proyecto de automatización de pruebas que existía anteriormente en la empresa, con el que se tardaba mucho más tanto en definir los tests como en implementarlos.

Con el *framework* prácticamente finalizado, se considera que aportará mucho valor dentro de la empresa, tanto a los desarrolladores como a aquellos que se encargan de validar que todo siga funcionando después de integrar un desarrollo nuevo en producción, ya que si todo aquello que deben validar se define dentro del marco de trabajo a través del lenguaje *Gherkin*, la próxima vez podrán hacerlo de forma automática.

Además, el hecho de usar el lenguaje común *Gherkin* con el área de *negocio* supone un punto de inflexión muy grande a la hora de definir los criterios de aceptación y escenarios de una nueva historia de usuario, ya que ayuda a que se produzca un mejor entendimiento entre esta área y los desarrolladores.

Con lo cual, desde el momento en el que se define una nueva tarea y se implementa el archivo *.feature* para añadir al *framework* aparece un nuevo punto de apoyo para los desarrolladores, el cual permite entender de una forma mucho más fácil, cuáles son los criterios de aceptación que requiere el área de negocio.

9 TRABAJO FUTURO

Como trabajo a realizar en un futuro se encuentra el desarrollo de una herramienta de generación de informes en lenguaje *html* que sea independiente de *Jenkins* para hacer más legible los resultados de las ejecuciones de los tests y la integración de este *framework* con la herramienta de administración de tareas de proyectos *Jira*.

También, añadir más flujos a los ya existentes de manera que el trabajo del equipo de calidad se reduzca a comprobar solamente aquellos cambios que se introduzcan en la nueva versión de la página web y no todos los flujos que comprueban que nada de lo que había anteriormente falle.

Además, se ha planteado implementar un túnel en la herramienta *BrowserStack* que permita ejecutar las pruebas contra entornos de desarrollo que no estén publicados fuera de la red interna de *Mango*.

Por último, se encuentra la creación de una interfaz de usuario en *javascript* que reduzca la dificultad de ejecución de las pruebas automáticas. Esta interfaz se conectará con el software de integración continua *Jenkins* para que éste realice la ejecución de las pruebas y publique el informe resultante.

10 AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a la empresa *Mango*, la oportunidad que se me ha brindado para desarrollar este framework de automatización de pruebas bajo su supervisión, con especial mención a Francesc Josep Muñoz, por su ayuda y dedicación en todo el proceso de desarrollo.

En segundo lugar, a mi tutora en la universidad, Ana Oropesa, por exigir siempre el buscar la mejor versión de mi para hacer el Trabajo de Fin de Grado y por acompañarnos siempre de una manera excelente en su desarrollo.

Por último, a mi familia, por haberme acompañado y animado en los momentos más difíciles de este proyecto.

BIBLIOGRAFIA

[1] Página oficial de Java. [En línea] Disponible en: https://www.java.com/es/download/faq/whatis_java.xml

[2] Introducción a Selenium. [En línea] Disponible en: https://www.digital55.com/desarrollo-tecnologia/herramientas-testing-introduccion-selenium/

[3] Porqué usar TestNG. [En línea] Disponible en: https://www.tutorialselenium.com/2018/03/07/usar-testng-selenium/

[4] Explicación de que es Gherkin. [En línea] Disponible en: https://cucumber.netlify.app/docs/guides/overview/

[5] Documentación oficial de Cucumber [En línea] Disponible en: https://cucumber.io/docs/cucumber/

[6] Documentación oficial Jenkins. [En línea] Disponible en: http://jenkins.io/doc/

[7] Explicación del término framework. [En línea] Disponible en: https://jordisan.net/blog/2006/que-es-un-framework/

[8] Documentación oficial de Python. [En línea] Disponible en: https://www.python.org/doc/

[9] Explicación del lenguaje C++. [En línea] Disponible en: https://blanchardspace.wordpress.com/2013/05/06/introduccion-a-c-que-es/

[10] Navegadores mayor cuota uso. [En línea] Disponible en: https://www.w3schools.com/browsers/

[11] Sistemas operativos mayor uso. [En línea] Disponible en: https://www.w3schools.com/browsers/browsers_os.asp

[12] Herramienta de gestión de código Git de Atlassian Bitbucket. [En línea] Disponible en: https://www.atlassian.com/es/software/bitbucket

[13] Explicación de qué es Git. [En línea] Disponible en: https://medium.com/@jclopex/que-es-git-desde-0-7678f4c3598d

[14] Página oficial de Maven. [En línea] Disponible en: http://maven.apache.org/

[15] Página oficial de IntelliJ. [En línea] Disponible en: https://www.jetbrains.com/es-es/idea/

[16] Explicación sobre el archivo pom.xml. [En línea] Disponible en: https://pierfinazzi.wordpress.com/2011/04/14/%C2%BFque-es-un-pom-xml/

[17] ¿Qué es un commit? [En línea] Disponible en: https://codigofacilito.com/articulos/commits-administrar-tu-repositorio

[18] WebDriver: Herramienta para testing de aplicaciones web [En línea] Disponible en: https://unpocodejava.com/2010/11/30/webdriver-herramienta-para-testing-aplicaciones-web-2/

[19] PageObject en las pruebas automatizadas. [En línea] Disponible en: https://www.dataart.com.ar/news/pruebas-automatizadas-page-object/

[20] PageFactory en el testing. [En línea] Disponible en: https://www.tutorialselenium.com/2019/02/05/page-object-model-selenium-webdriver/

[21] El formato de archivo xml. [En línea] Disponible en: https://www.online-convert.com/es/formato-de-archivo/xml

[22] Documentación sobre testng.xml. [En línea] Disponible en: https://testng.org/doc/documentation-main.html

[23] Pipelines en Jenkins. [En línea] Disponible en: https://www.paradigmadigital.com/dev/pipelines-de-jenkins-evolucion-continuous-delivery/

[24] Plugin Cucumber Reports. [En línea] Disponible en: https://plugins.jenkins.io/cucumber-reports/

[25] Explicación de html. [En línea] Disponible en: https://www.w3schools.com/html/

[26] Explicación del atributo href. [En línea] Disponible en: https://www.w3schools.com/tags/att_a_href.asp

[27] Documentación oficial de BrowserStack. [En línea] Disponible en: https://www.browserstack.com/docs

[28] Qué es la metodología Kanban. [En línea] Disponible en: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/

ANEXO I:

Listado de casos de uso automatizados:

- Realizar un registro de un nuevo usuario.
- Realizar una compra con un usuario existente
- Realizar una compra con un usuario que no existe y que se pida registrarse para finalizarla.
- Comprobar que aparecen los menús básicos (Mujer, Hombre, Niño, Niña, Violeta)
- Comprobar que existen submenús para estos menús.
- Buscar una prenda por el código de referencia.
- Añadir N productos a la bolsa de compra y comprobar que se han añadido.
- Eliminar productos de la bolsa.
- Comprobar que el modal de adhesión al club Mango Likes You funciona correctamente y redirige a los formularios donde debe redirigir.
- Comprobar que el modal fijado en la parte inferior de la página de Mango Likes You redirige correctamente a las clientas.