Escuela Politécnica Superior

# Plan para la implantación del “testing” en una PYME

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Autor:
Jorge Adalberto Díaz Valdés

Tutor/es:
José Vicente Berná Martínez

Septiembre 2020

Universitat d’Alacant
Universidad de Alicante

# Resumen

En este trabajo de fin de máster se realiza una propuesta de mejora del proceso del testing para una PYME objeto de estudio. En concreto nos centramos en el proceso de ejecución de las pruebas. El departamento de calidad de la empresa en su labor diaria realiza pruebas de aceptación a una aplicación web de un grado de complejidad alto. Por este motivo necesita garantizar que su producto antes de salir al mercado tenga el menor número de defectos posibles y que cumpla con las expectativas de su cliente. Para esto se dedica parte de su presupuesto al diseño y ejecución de pruebas de aceptación, valiéndose para ello de un pequeño número de personas dedicado a la actividad del testing. El personal que realiza esta labor la ejecuta de forma manual. Esto conlleva a que los recursos humanos sean utilizados de forma ineficiente y como consecuencia la calidad del software se vea afectada ya que los trabajos manuales en este tipo de actividad suelen realizarse con poca eficacia y eficiencia debido a la naturaleza humana ya que, nosotros los seres humanos no siempre estamos al 100% de nuestra capacidad intelectual, física y psíquica.

En este documento se propone una nueva metodología de trabajo para lograr qué esta forma de proceder de la empresa mejore. Este método de trabajo debería mejorar la eficiencia y la productividad de la actual forma de proceder de dicha empresa en la realización del proceso de ejecución del testing puesto que a través del uso de herramientas se realizará una transición de ejecución de casos de pruebas de forma manual a automática, consiguiendo así explotar al máximo los recursos que actualmente dispone la empresa sin que ello implique un aumento en el presupuesto del producto software que se desarrolla actualmente.

Apoyándonos en la metodología propuesta y partiendo de una Propuesta de la solución en la que se realizará un estudio que demuestra los beneficios de la automatización en la ejecución de las pruebas de aceptación, se llevará a cabo un Plan de puesta en marcha que la empresa objeto de estudio deberá implantar para conseguir así los beneficios deseado en la calidad del software que desarrolla. Habiendo la empresa implantado este plan estará en condiciones para que, de una forma natural tenga la posibilidad de estandarizar su proceso de pruebas adoptando Marcos de mejora e implantación del testing. Para demostrarlo se tomará uno de estos Marcos de reconocido prestigio y se comprobará cuánto de similitud tiene la empresa en su forma de proceder al realizar el proceso de pruebas con este marco y cuánto nuestra propuesta se alinea también con él.

# Motivación, justificación y objetivo general

Durante mis primeros 3 meses como "tester" en una empresa cuyo nombre no es relevante mencionar, el equipo de calidad (QA) al cual formaba parte, hablaba mucho de la importancia de la automatización a la hora de ejecutar los casos de prueba, pero nadie se atrevía a, por lo menos, hacer un prototipo de un caso de prueba automatizado con la herramienta de la cual tanto hablaban, Selenium Webdriver (SeWe) [11]. Alrededor del cuarto mes aún se seguía debatiendo sobre el mismo tema, pero todo quedaba en el viento, ya que por muy de acuerdo que estuviere el jefe de equipo en implantar la automatización, cada vez que éste exponía el asunto a la alta directiva de la empresa, la propuesta terminaba siendo rechazada, ya que al proceso de pruebas lo terminaban dejando siempre en un segundo plano y por la parte del equipo de “testing” uno de los principales inconvenientes que se exponía era la falta de tiempo para automatizar los casos de prueba durante el período de desarrollo, comparado con el tiempo empleado para la realización y ejecución de los mismos de forma manual. Este escenario se suele repetir muy a menudo en muchas empresas de desarrollo de aplicaciones. Anteriormente, ya había trabajado también para otra empresa que, desgraciadamente aplicaba la misma lógica para justificar el porqué de no emplear la automatización. Incluso he llegado a conocer que existen algunas empresas que aun teniendo la posibilidad de tener un equipo de testing para realizar pruebas emergentes funcionales, no contratan a personal cualificado con conocimientos de realización de pruebas, ni tan si quiera de informática y ponen en manos de personas con disciplinas completamente ajenas a la anterior mencionada, informática, a realizar pruebas manuales antes de entregar el producto final. En esta empresa el único tipo de pruebas que se realiza son las pruebas emergentes funcionales de aceptación y la forma en que se realiza es manual 100%. Los testers prueban las interfaces de aplicaciones tal cual lo estuviera haciendo el usuario final. Estas pruebas requieren preparar datos previos y suelen ser muy repetitivas, llegándose incluso a repetir varias veces durante todo el tiempo que dura el desarrollo del producto.

En una ocasión, se hizo una planificación de entrega transcurrido 6 meses y llegada la fecha, los diferentes equipos de desarrolladores comenzaron a revisar la aplicación en las diferentes pantallas que habían trabajado. Fue en esos momentos cuando comenzó a suceder lo inesperado (o quizás lo contrario), cada vez que un equipo encontraba un error y lo corregía, esto generaba errores en el código que ya funcionaba, pero desarrollado por otro equipo. Así el segundo equipo de desarrolladores corregía sus errores, pero esto, curiosamente generaba un error distinto en el código de un tercer equipo de desarrollo. Dada esta situación, los testers

tuvimos que volver a probar de forma manual toda la batería de pruebas que habíamos hecho durante esos seis meses anteriores, pues la aplicación había dejado de ser estable. Muchas de las reejecuciones de pruebas que tuvimos que hacer eran muy repetitivas por el simple hecho de que cambia algún dato, pero el camino a recorrer para su ejecución seguía siendo el mismo al igual que las acciones realizadas. Otras veces había que rellenar un formulario muy extenso que tardaba cerca de 30min en completarse para luego verificar que el valor devuelto era el correcto. Como consecuencia la entrega se vio afectada con un retraso de 3 meses más tarde de lo previsto.

Durante mis estudios de grado impartí la asignatura “Programación y pruebas de sistemas software” (P.P.S.S.), en ella se demostraba y se hacía mucho hincapié en la importancia de la automatización, todas las ventajas que nos ofrecía, reducción del tiempo de ejecución de las pruebas, como se eliminaban los riesgos de introducción de errores en el código ya hecho mediante las pruebas automáticas de regresión, por otro lado se constataba cómo muchas empresas por no probar bien o no realizar pruebas sistemáticas perdían su reputación y su economía se veía tremendamente afectada. También ponían sobre la mesa el pensar de muchas empresas de que el realizar las pruebas consumía una gran parte del tiempo del desarrollo y que por eso esta actividad carecía de sentido, aunque toda la bibliografía que se recomendaba daba prueba fehaciente de que sí era viable, rentable y necesaria la actividad del testing y sin lugar a dudas la automatización.

Con toda esta experiencia vivida, habiéndome basado en los hechos del último párrafo y convencido de la gran ayuda que hubiera sido haber tenido pruebas automáticas durante esos momentos difíciles de alto riesgo en el desarrollo de la aplicación, he sentido la necesidad de buscar la forma de mejorar situaciones similares a las de entonces que me puedan o pueda volver a ocurrir en una empresa, pequeña y/o mediana, a través de la implantación de la automatización.

# Agradecimientos

Deseo empezar esta sección dando gracias a Dios por haberme permitido llegar al fin de este camino lleno de obstáculos e improvisos. Por haberme dado la sabiduría y la fuerza para atravesar todas y cada una de las dificultades encontradas. A él va mi primer agradecimiento. Segundo, agradezco a mi madre en ayudarme con lo que ha podido a pesar de su avanzada edad. A mi hija, porque a pesar de su corta edad ha colaborado dándome apoyo siempre que se lo pedí. Por otro lado a Andrés Montoyo quien me ayudó con la elección de este master, a la gran mayoría de profesores del master, que gracias a su comprensión y flexibilidad hicieron que mi paso por las asignaturas fuera más llevadera y posible de realizar: Virgilio Gilart, Carlos Perez, Higinio Mora, Faraón Llorens, Manuel Marco, José Luis Albentosa, Ramón Rubio, Juan Carlos Trujillo, José Norberto, algunos más que se me quedan por mencionar y finalmente mi profesor tutor José Vicente Berna Martinez que también ejerció de la misma labor durante mi carrera y de quien siempre he aprendido mucho y siempre me ha dado todo su apoyo. A todos ustedes muchas gracias.

# Citas

*La esperanza que se demora es tormento del corazón,*
*pero árbol de vida es el deseo cumplido.*
*Santa Biblia, Antiguo testamento, Proverbios 13:12*

*Bienaventurado el hombre que halla la sabiduría, y que obtiene la inteligencia; porque su ganancia es mejor que la ganancia de plata, y sus frutos más que el oro fino. Más preciosa es que las piedras preciosas, y todo lo que puedes desear no se puede comparar a ella. Largura de días está en su mano derecha; en su izquierda, riquezas y hora. Sus caminos son caminos deleitosos, y todas sus veredas paz. Ella es árbol de vida a los que de ella echan mano, y bienaventurados son los que la retienen*
*Santa Biblia, Antiguo testamento, Proverbios 3:14-18*

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# 1. Introducción

¿Por qué probar es necesario e importante para las empresas en cualquier proyecto de desarrollo software?

Probar, en inglés “testing”, es fundamental e imprescindible en todo desarrollo software principalmente porque quienes desarrollan aplicaciones son personas y cada individuo es distinto, no siempre tenemos el mismo estado de ánimo. Cualquier problema en la vida cotidiana de una persona, cualquier situación personal del día a día puede hacer que el equilibrio emocional y por tanto el nivel de concentración se desestabilice provocando que se cometan errores a la hora de desempeñar la labor profesional en el puesto de trabajo, por ejemplo, tomar requisitos, escribir código y por qué no, a la hora de probar aquello que se ha desarrollado cuando no se tienen los métodos y directrices necesarios. Estos errores cometidos serán los responsables de la generación de defectos en un producto acabado y como consecuencia pueden dar lugar a fallos que pudieran provocar desde pérdidas humanas hasta el cierre de una empresa. Un producto con un porcentaje de error superior al que los usuarios solemos tolerar afecta a la marca de empresa y esto a la larga crea una la mala reputación, haciendo que nadie más quiera ni comprarlo ni utilizarlo. Por esto, porque el ser humano no es perfecto y porque no somos capaces de probar de forma exhaustiva un producto al 100% en todas sus casuísticas, ya que aunque quisiéramos intentarlo no alcanzaría ni con el tiempo del universo para terminar de probar cada una de las posibilidades que se pudieran dar, se hace imprescindible que exista, una actividad durante el desarrollo del software que (1) garantice un mínimo grado de satisfacción y seguridad sobre el producto que se pretende sacar al mercado y que finalmente será usado por usuarios, (2) sea capaz de predecir defectos ocultos que comprometa el correcto funcionamiento del producto y/o la seguridad del usuario. Existen dos tipos de pruebas, estáticas y dinámicas. Dentro de esta última encontramos las *pruebas de entrega* en inglés “release test”, son éstas las que considero, basándome en mi experiencia académica y profesional, que como mínimo las empresas deben realizar para garantizar la seguridad y satisfacción mencionadas anteriormente. A este nivel estaremos probando todo el sistema, todas las interacciones de sus componentes, todas sus interfaces y es en este preciso momento donde emergen problemas que solo se pueden detectar al probar el sistema como un todo. No importa si a un nivel más bajo de pruebas (pruebas unitarias o de integración de componentes), nos hemos dejado algún error sin detectar, puesto que siempre saldrán a flote cuando ejecutemos algún caso de pruebas bajo un escenario concreto que interactúe con ese componente, objeto, clase, etc., que esté ocultando el error. Las “reléase test” se centra en lo

que el usuario ve, interactúa y usa, vela porque el sistema funcione tanto en condiciones normales como en circunstancias extremas que hacen que el sistema disminuya su rendimiento (sobrecarga del sistema por demasiadas peticiones, demasiados usuarios haciendo solicitudes a la misma vez, etc.). Sin perder de vista a estas pruebas que ponen al sistema al límite de sus posibilidades, es imperativo que las empresas centren su atención en hacer que el sistema funcione en condiciones normales con el mínimo número de errores posibles, preferiblemente cero, para asegurar así una buena experiencia de usuario. Un proyecto con una buena experiencia de usuario siempre crea una buena imagen de la empresa lo que al final se traduce en ganancias económicas que es al fin y al cabo lo que realmente interesa y buscan las empresas; ellas están y existen para ganar dinero. Por este motivo quiero centrar mi trabajo en las *pruebas dinámicas de entregas funcionales*.

Es muy importante tener en cuenta 3 factores fundamentales que nos permitirán hacer unas pruebas con una mayor o menor profundidad, estos son, el propósito del software que se quiere construir, las expectativas de los usuarios hacia el mismo (dependiendo de esto se le dedicará un mayor tiempo en cada una de las pruebas que se realicen), por último, el personal encargado de realizarlas. La persona que realice la actividad del "testing" debe conocer qué y cómo realizar las pruebas para, por ejemplo:

- Saber cuál es el mínimo número de pruebas necesario.
- Saber cuál es la cantidad suficiente de pruebas que garantiza la calidad del producto en un espacio de tiempo que, la mayoría de las veces está muy acotado.
- Garantizar que una vez puesto en marcha el producto y sea utilizado por el usuario todo funcione como éste espera.

La figura que realiza esta actividad es la del "tester". Es necesario entender que éste no debe de estar en contacto con el código fuente del aplicativo para que no resulte contaminado y esto influya en la eficacia de los casos de pruebas a desarrollar para detectar defectos, además su labor requiere desarrollar ciertas habilidades que solo se consiguen con la interacción de personas con varios roles en una empresa y debe de estar en constante comunicación con el Product Owner (jerga en la metodología Scrum) o cliente para detectar cualquier fallo en los requerimientos.

Ha de quedar claro que el "testing" nunca demostrará que el producto estará libre de errores ni que su comportamiento será el que se espera en todas y cada una de las distintas situaciones que se puedan dar en un escenario de uso determinado, pero sí sacará a la luz la existencia de errores…

*Program testing can be used to show the presence of bugs, but never to show their absence!*

*Edsger W. Dijkstra*

Por ejemplo, un fallo en la toma de requisitos o un requisito que no se haya tenido en cuenta será imposible de detectar háganse el número de pruebas que se quieran hacer.

La actividad de probar "testing" se utiliza para reducir el riesgo y mejorar la calidad del producto, demostrar a los desarrolladores, los stakeholders y el cliente, que el producto cumple con los requerimientos, lo que contribuye al éxito del producto de una empresa, siempre y cuando se tenga en cuenta la figura del tester, el mismo ha de trabajar muy de cerca, codo a codo con los diseñadores mientras se diseña el sistema y con los desarrolladores durante el proceso de construcción del software, debe estar involucrado desde el principio en la toma de requerimientos del producto a desarrollar, bien sea con el/los cliente/s o los stakeholders y en sus posteriores entrevistas de refinamiento para que se forje en él la visión global del producto. También es el tester quien debe verificar y validar que el producto desarrollado cumple con esas altas expectativas que el cliente y/o "stakeholders" tienen.

El gran problema que me lleva a escribir este trabajo es que la gran mayoría de las empresas se centran en buscar muchas veces buenos desarrolladores y sin embargo no son exhaustivos en buscar personal que sepa realizar pruebas, ya que esta tarea es tratada como residual, necesaria pero se hace en la medida que se puede, sin que sea sistemática y mucho menos automatizada, por lo menos en las empresas pequeñas, y es que es normal que una empresa con recursos limitados se centre primero en tener un producto que poder probar, y luego con el tiempo y recursos disponibles, las pruebas. La idea de este trabajo es, para estas empresas que ya son conscientes de la necesidad casi obligatoria de hacer pruebas, proporcionarles una metodología de cambio de un proceso manual a un proceso automático, donde deberán adaptar ciertas cosas a la hora de programar para facilitar luego la fase del test. Además, a la hora de implantar un sistema de pruebas se puede ir de lo más sencillo a lo más complicado, de forma que vayamos haciendo despliegues por pasos hasta lograr pasar de una situación manual a una totalmente automatizada. En este proceso se describen procedimientos, recursos y herramientas necesarios para lograrlo.

# 2. Estado del arte.

El testing es tan importante en el desarrollo del software tanto como el mismo desarrollo. Con él, mejoramos la calidad del producto, evaluamos y valoramos lo bueno que es nuestro producto [2], evitamos los riesgos y creamos confianza en el cliente que solicita el producto desarrollado [3]. No es lo mismo realizar pruebas para el desarrollo de una aplicación de video juegos, que para un sistema automático de control de vuelo o un sistema de control/soporte de signos vitales de un paciente en un hospital. Mientras más grande y complejo es el sistema más aumenta el riesgo de cometer fallos en su desarrollo. Esto implica que a mayor el riesgo en lo que se esté desarrollando mucho mayor será en número de tests que se necesite realizar y mejor ha de ser la calidad en la realización de esos tests [3]. En estos últimos ejemplos las empresas desarrolladoras deberían garantizar al 100% que el aplicativo funcionará tal y como se espera. Existen numerosos ejemplos que demuestran que de haber probado de una forma adecuada los diferentes sistemas puesto en producción, se hubiera evitado pérdidas humanas, económicas, de tiempo, de recursos y de reputación. A continuación, se mencionan 3 ejemplos que lo demuestran:

El primer ejemplo es la explosión del Ariane 5

*Figura 1. Lanzamiento y explosión del Ariane 5*
(Fuente http://www-users.math.umn.edu/~arnold)

El primer lanzamiento al espacio del cohete “Ariane 5” realizado por la Agencia Espacial Europea del inglés “European Space Agency” (E.S.A.) en junio de 1996, fracasó debido a un error software. Nada más despegar y tras 37s, el sistema de autodestrucción del aparato se activó haciendo que explotara al instante para evitar daños mayores. El motivo por el que sucedió esto fue que al reutilizar el módulo software de la versión anterior, Ariane 4, un error de bits hizo que un número pasara de ser positivo a negativo al alcanzar su valor máximo, causando un cambio de sentido en la trayectoria del cohete. Este hecho tuvo un alto coste económico para la E.S.A., las pérdidas fueron millonarias, solo el cohete más lo que portaba con él se calculó en unos $500 millones y el coste total de proyecto ascendió a $7 billones [4].

El segundo ejemplo es el fallo del Misil “Patriot”.

*Figura 2. Misil Patriot*
(Fuente http://www-users.math.umn.edu/~arnold)

El 25 febrero de 1991, durante la guerra del Golfo, un misil norteamericano falla la intersección con otro misil enemigo para su derribo por un defecto en el cálculo de la trayectoria. El no colisionar hizo que cayera en una barraca del propio ejército americano. El motivo de este accidente se debió a una corrección en el cálculo del tiempo en algunas partes del código, pero no en todas, provocó que la inexactitud en el tiempo no se tuviera en cuenta para abortar. El resultado final del cálculo del tiempo daba un número distinto al que debía ser. En este caso las pérdidas fueron humanas, pues 28 soldados perdieron la vida y otros 100 resultaron heridos [5].

El tercer ejemplo, se desarrolla en el sector de la salud, es el fallo del Therac-25.

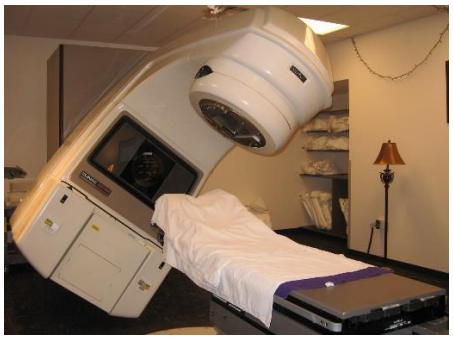

*Figura 3. Therac-25*
(Fuente https://fracasosdesoftware.wordpress.com)

El Therac-25 es un acelerador linear, un dispositivo acelerador de partículas que aumenta la energía de partículas atómicas cargadas de electricidad. Los hechos sucedieron entre el período de junio de 1985 y enero 1987, y se debió a un defecto en el software. 6 pacientes diagnosticados con cáncer recibieron una enorme sobredosis de radiación cuando recibían el tratamiento de radioterapia. Los accidentes se produjeron en varios centros hospitalarios, 5 de ellos en EEUU y 1 en Canadá. En este caso la empresa “Atomic Energy of Canada Limited” dueña del producto demostró no haber empleado ninguna técnica de control calidad del software. La

nueva versión del equipo Therac-25 usó módulos software de versiones anteriores Therac-6 y Therac-20. Estos módulos ya arrastraban errores no corregidos. La agencia norteamericana responsable del cuidado de la salud pública en U.S.A. “Food and Drug Administration” tras el resultado de su investigación en el caso, escribió que era increíble cómo los casos de pruebas hechos para comprobar que las primeras correcciones del Therac-25 eran correctas demostraban exactamente lo contrario. En esta ocasión también hubo pérdidas humanas, 3 de los pacientes perdieron la vida por esta sobredosis de radiación. Los 3 restantes acabaron con daños serios de salud [6].

## 2.1. Proceso de implantación del testing

Cualquier proyecto de creación de un producto software pasa por diferentes etapas en su desarrollo sin importar el modelo de desarrollo empleado (cascada, iterativo), se realizan actividades como la especificación de requisitos, de funcionalidad, el diseño, implementación y test. Esta última actividad juega un papel fundamental ya que verifica y valida que todo se está realizando correctamente, tal y como se espera, y que se está satisfaciendo realmente las necesidades del usuario respectivamente [3].

Es importante conocer y tomar la decisión de qué es lo que vamos a conseguir con la realización del test y establecer objetivos claros para cada test [3]. El testing no siempre se realiza de igual forma en todas las empresas, todo dependerá del contexto en el que nos hallemos [3], estos podrían ser algunos ejemplos de contextos:

- Modelo de ciclo de vida de desarrollo de software y metodologías de proyecto en uso.
- Niveles y tipos de prueba considerados.
- Riesgos de producto y de proyecto.
- Dominio del negocio.
- Restricciones operativas, incluyendo, pero no limitadas a:
    - o Presupuestos y recursos.
    - o Plazos.
    - o Complejidad.
    - o Requisitos contractuales y normativos.
- Políticas y prácticas de la organización.
- Estándares internos y externos necesarios.

Por todo lo anterior, es importante entender que es necesario tener las actividades que se desarrollan en todo proyecto software recogidas en procesos. Los procesos han de tener la característica de poder repetirse una y otra vez para obtener buenos resultados, de no ser así estaremos propiciando cometer los mismos errores cada vez que nos enfrentemos a las mismas o similares actividades [2]. Son muchos los beneficios que obtendremos al poder regirnos por procesos ya definidos, por ejemplo, que los mismos pueden ser reutilizados en nuevos proyectos, pueden ser evaluados usando métricas como coste, calidad y tiempo de entrega de un producto. Finalmente, un proceso se puede refinar y mejorar para obtener mejores resultados.

El marco de trabajo "Capabilty Maturiry Model" (CMM) reemplazado en la actualidad, aunque muchas organizaciones lo siguen usando, por el "Capabilty Maturiry Model Integration" (CMMI), es el modelo creado por el Instituto de Ingeniería del Software (SEI) en la universidad de Carnegie Mellon, utilizado por las empresas para evaluar sus procesos de desarrollos Softwares. El nivel de ***madurez*** de un proceso de desarrollo nos dice exactamente cuán capacitada está la organización para producir un software de alta calidad a bajo coste. El CMM, por otra parte, permite también a un cliente evaluar las capacidades de la organización y así saber cuánto de fiable es la organización en cumplir con la entrega de un producto que le ha sido solicitado. Se dice que una empresa carece de madurez cuando sigue procesos inmaduros. Una empresa inmadura puede no tener un proceso definido a seguir y aunque lo tuviere, podría no seguirlo. P. ej.: los profesionales (gestores, desarrolladores) reaccionan y atacan a los problemas a medida que van surgiendo, en vez de adoptar medidas de contingencias para prevenirlos o reducir su frecuencia de aparición. Para asegurarnos de que una empresa pueda seguir un proceso nuevo, el proceso ha de ser simple en sus inicios y fácil de seguir, puesto que para las organizaciones es mucho más práctico seguir un proceso creado con la posibilidad de que se pueda ir mejorando continuamente a encontrar el "ideal" que se ajuste a sus necesidades[2]. El CMM brinda esta posibilidad, permitiendo a las organizaciones ir implementado 5 niveles de forma ordenada y secuencial. Se debe empezar por el nivel 1, pasando de nivel a nivel, a medida que se van cumpliendo las metas que acarrea cada uno. Pasar de un nivel a otro es un indicador de que la capacidad de proceso de la empresa ha mejorado por lo que la lleva a producir software de mejor calidad a bajo coste. Otros modelos y estándares (ISO 9001, BOOTSTRAP, SPICE) también han sido desarrollados con similares objetivos, pero ninguno de ellos trata de forma adecuada los problemas del testing[13]. Por otra parte, debido a que el testing es considerado un subproceso completamente distinto o un proceso más dentro del proceso de desarrollo del software "The testing process is a component, or subprocess, of

the overall software development process...", ya que en él se recogen un conjunto único de actividades, técnicas, estrategias y políticas como algunas de las que se nombran a continuación [2]:

- El testing comienza casi al mismo tiempo que el proyecto se conceptualiza y seguirá existiendo mientras que el sistema exista. Su objetivo principal es demostrar la existencia de defectos en los aplicativos softwares y mostrar hasta qué punto un sistema tiene diferentes atributos de calidad (fiabilidad, rendimiento, estabilidad y escalabilidad).
- El testing se realiza por profesionales con diferentes responsabilidades en las diferentes fases y/o niveles del desarrollo del software.
- El testing se puede realizar de forma manual y automática.
- La política de la organización influencia el nivel de calidad alcanzado por el producto cuando se prueba, el tiempo que se le debe dedicar y cuánto del presupuesto total se le puede asignar al testing.

La comunidad del testing ha desarrollado varios modelos de madurez del test, similar al concepto de evaluar y mejorar el proceso de desarrollo del software, para que las empresas puedan complementar su capacidad de producir un software de mejor calidad a bajo coste alcanzando su nivel de madurez. Estos modelos se han creado con la idea de que el modelo se vaya adaptando a la empresa y no al revés. A continuación, se exponen algunos:

- Modelo de madurez del testing (TMM) del inglés "Testing Maturity Model"

Utilizado como una guía para implantar y mejorar el proceso del test. Se basa en el CMM y de igual forma implementa 5 niveles o estados, cada uno, a excepción del primero, con sus metas específicas a conseguir. El nivel 1, a diferencia del resto se alcanza de forma directa pues la empresa no necesita hacer nada en particular. Es el escenario donde el testing no se tiene en cuenta como una actividad crítica, como un proceso distinto en el desarrollo del software, no existe ningún proceso definido o no se tiene como importante la actividad del test. Cuando se pasa a los posteriores niveles ya comienzan a seguirse procesos más ajustados que hacen que la empresa alcance su madurez. Una vez que la empresa consigue estar en un nivel superior al 1, se siguen procesos de mejora acorde al entorno de la empresa [2].

- Modelo de mejora del proceso de pruebas, del inglés "Test process Improvement (TPI)"

Se utiliza para controlar y mejorar las pruebas basándose en el conocimiento y la experiencia de una empresa holandesa llamada IQUIP a finales de 1990. A través de un marco de referencia

permite determinar los puntos débiles y fuertes del proceso de pruebas de una organización. Se compone de unas 20 áreas claves, del inglés “key areas” que recogen distintos puntos de vistas del proceso de pruebas, p.ej. herramientas de pruebas, técnicas de especificación de las pruebas e informes. Las áreas, a su vez se clasifican por niveles de madurez (A, B, C, D) y cada nivel contiene puntos de verificación “check points” que permiten determinar la madurez. Por último, se proporciona unas sugerencias de mejoras, del inglés “improvement suggestions” para poder alcanzar el nivel de madurez deseado [14]. En el 2009 aparece su sucesor ***TPI Next***, éste a diferencias de su antecesor tienes 16 áreas claves y un conjunto de mejoras que lo hacen más fácil de entender y aplicar a cualquier proyecto, por ejemplo, ahora los niveles de madurez A, B, C y D cambian por nombres más descriptivos: Initial, Controlled, Efficient y Optimizing [15].

- Integración del Modelo de madurez del testing (TMMi) del inglés “Testing Maturity Model Integration”

El alcance de este marco va desde los niveles de pruebas (estáticas y dinámicas) hasta las pruebas estructurales (ciclo de vida del software, Técnicas, infraestructura y organización). Este marco permite su uso como modelo de referencia durante la realización de la actividad de mejora del proceso del test. Su desarrollo se fundamenta y tiene como principales pilares el TMM y el CMMI, siendo este último su complemento. Es un modelo de estado, centrado en todos y cada uno de los niveles de pruebas (componentes, integración, sistema y aceptación), incluyendo también las pruebas estáticas.

También utiliza el concepto de 5 niveles de madurez para dar soporte a las organizaciones en la evaluación y la mejora de los procesos de pruebas, lo que ayuda a dar soluciones a los problemas que les surgen tanto a los profesionales del testing como a los profesionales del desarrollo del software.

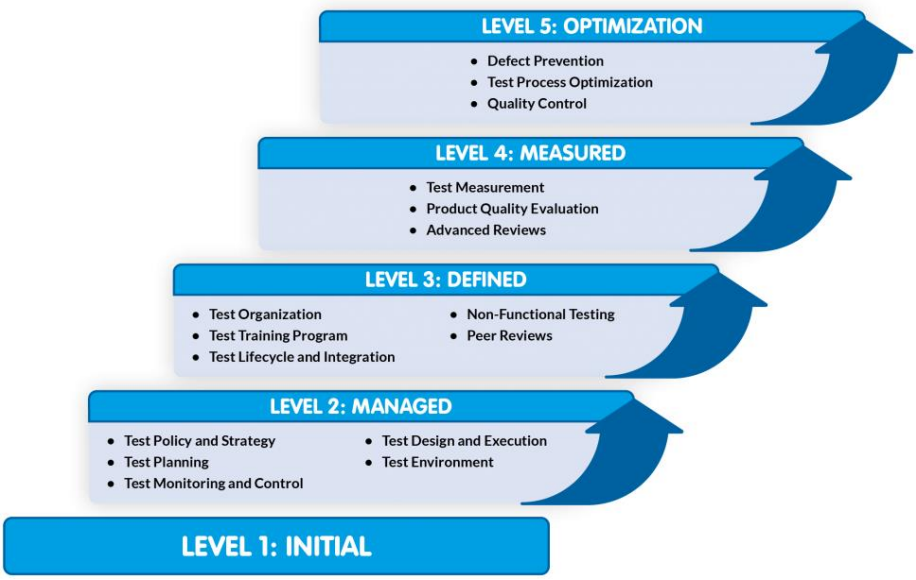


*Figura 4. Niveles de madurez y áreas de procesos del TMM*i

*(Fuente: https://www.tmmi.org/tmmi-model/)*

En general los niveles se componen de áreas de procesos que cada organización debe implementar para conseguir su madurez en ese nivel. Con cada área de proceso se consiguen unas metas (específicas y genéricas) y para conseguir dichas metas, cada una de las metas han de alcanzar un conjunto de prácticas (específicas y genéricas) que no son otra cosa que un conjunto de actividades a realizar. En la siguiente Figura 5 podremos apreciar la estructura de un nivel.

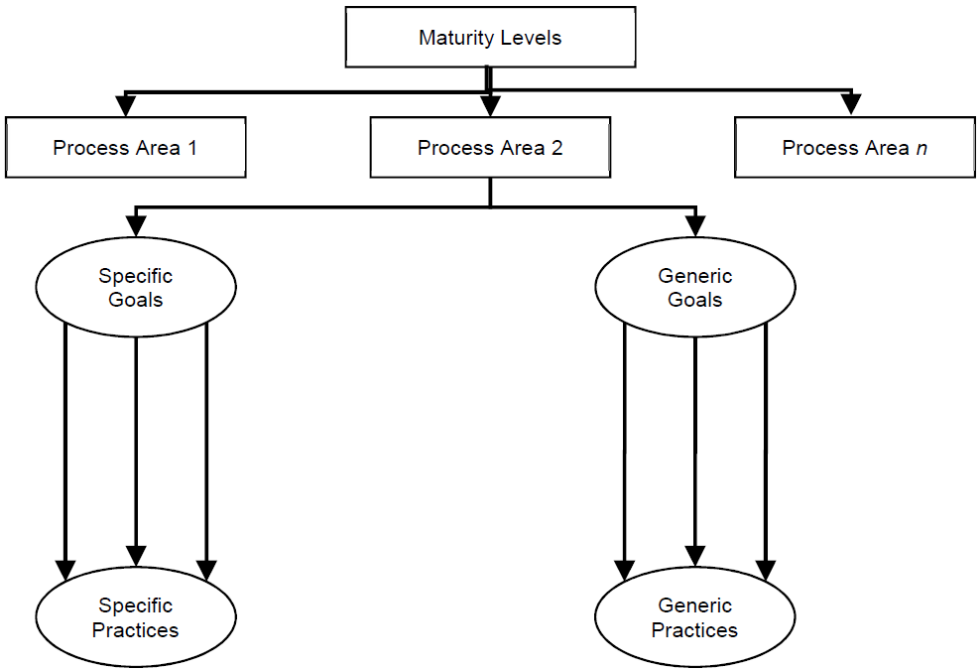


*Figura 5. Estructuras de un nivel en el modelo TMMi*

*(Fuente: TMMi-Framework-R1-2.pdf)*

## 2.2. Proceso del testing

El Comité Internacional de Certificaciones de Pruebas de Software del inglés (International Software Testing Qualifications Board: ISTQB) es la institución por excelencia reconocida como líder en el ámbito de las pruebas del software. La ISTQB es una organización sin ánimo de lucros que acredita y evalúa a los profesionales del testing. Gracias al trabajo voluntario de cientos de profesionales en el área del testing ha ido recopilando información de sus experiencias consiguiendo así una biblioteca de conocimiento en esta área y convirtiéndose en la certificadora del testing por excelencia para las empresas a nivel mundial [1]. Una vez alcanzada cierta madurez en la empresa, la ISTQB propone un proceso fundamental de pruebas a seguir, el "Test process" Figura 6 [3]. Éste recoge pasos fundamentales que nos asegura que no nos saltamos pasos críticos durante la realización de la actividad del testing en un proyecto y que hacemos las cosas en el orden correcto [3].

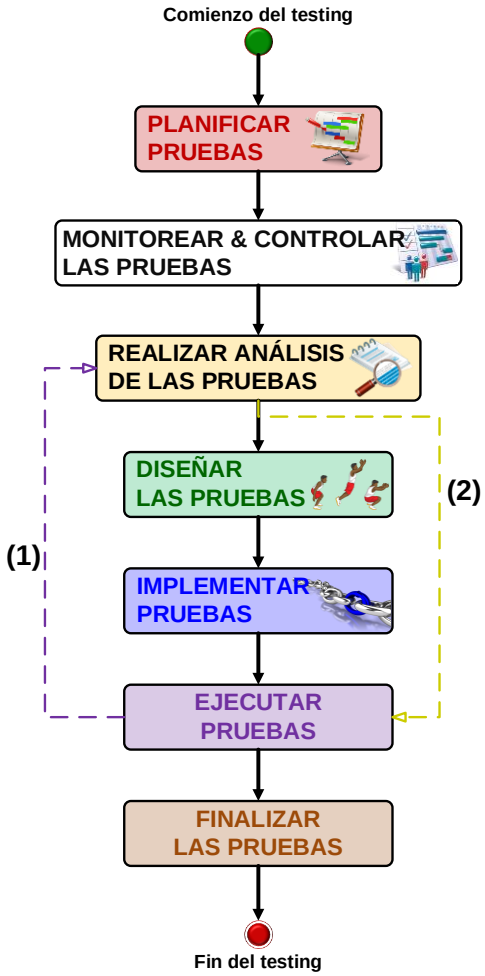


*Figura 6. Proceso básico del testing (Fuente propia)*

Para que el testing sea correcto hay que saber Cómo, Cuándo y Qué se va a probar. Aunque de la actividad del testing la parte que más evidente y llama la atención es la de *ejecución* de los

casos de pruebas, hay otras partes (o grupos de actividades) que no lo son tanto, pero que son las responsables de que lleguemos a ese momento de ejecución y después existen otros grupos de actividades que nos permite trabajar con los resultados de dicha ejecución.

Estos grupos de actividades no siempre se desarrollan de forma independiente, tampoco se siguen en un orden fijo, ya que un grupo de actividades ya visitado puede ser vuelto a visitar en repetidas ocasiones e incluso de forma simultánea con otros grupos, por ejemplo en la anterior figura 7, la línea discontinua (1) muestra cómo tras una ejecución de un caso de pruebas se llega a la conclusión de que se necesitan agregar nuevas funcionalidades, lo que provoca volver a analizar y a diseñar nuevas pruebas, mientras se continua el flujo de trabajo con el resto de casos de pruebas que están en ejecución. Por su parte la línea (2) supone otra situación en la que por falta de tiempo se necesita realizar la ejecución de las pruebas antes de haberlas tan siquiera diseñado [3].

Seguidamente se da paso a una breve descripción de cada uno de los grupos de actividades pertenecientes a la figura 7:

PLANIFICAR PRUEBAS

El “test manager” es la persona encargada de realizar la planificación. Al planificar se busca determinar qué es lo que se va a testear y cómo se va a hacer. Se crean documentos que son un plano de cómo y quién va a realizar cada una de las actividades de pruebas, se establece el alcance de las mismas (completion creteria) esto es, saber cuándo parar o dejar de probar y que ello no incurra en ningún riesgo para el cliente ni para la empresa. Estos documentos se irán corrigiendo durante todo el tiempo que dure el proceso. Está compuesto por un documento principal “Mater Plan” el cual define a alto nivel las actividades de testing que se van a planificar y otro conjunto de documentos denominados “Test level plans” que recogen en detalle las actividades de pruebas y sus estimaciones según el nivel en el que nos encontremos. P. el.: para las pruebas a nivel de sistema se creará el plan de pruebas del sistema, del inglés “Sistem test plan” [3].

MONITORIZAR Y CONTROLAR LAS PRUEBAS

Este grupo de actividad es realizada por el personal a cargo de llevar la gestión de las pruebas. El objetivo de monitorear es proporcionar “feedback” y visibilidad del progreso de las actividades de testing a los interesados (Product Owner, stakeholders,…) apoyándose del Plan de pruebas y en los resultados de la ejecución de las pruebas. Permite detectar irregularidades, desviaciones o cualquier problema que afecte el curso normal del proyecto. La obtención de los

datos para el monitoreo se puede hacer de dos formas, manualmente o de forma automática mediante herramientas de gestión de pruebas. El objetivo de controlar es llevar a cabo las acciones necesarias para la corrección de los problemas detectados durante el monitoreo, p. ej. Volviéndole a dar prioridad a los test en caso de retraso en la entrega del software, cambiar el calendario de las pruebas, volviendo a evaluar los criterios de fin y comienzo [3].

REALIZAR ANÁLISIS

En este grupo de actividad se realiza un conjunto de actividades cuyo propósito es saber qué es lo que se quiere probar, examinando exclusivamente la base de prueba. Ésta no es más que toda la información necesaria para poder crear los casos de pruebas (los requisitos de la aplicación que se ejecutará en un dispositivo, el mismo dispositivo). La información se puede encontrar en más de un documento. Se pueden utilizar algunas de las técnicas de pruebas como la de "caja negra" para identificar de forma más exacta y segura las condiciones de pruebas (lo que pasará como consecuencia de acciones o elecciones previamente hechas) que son las que nos permiten alcanzar los objetivos de pruebas específicos. Mediante esta actividad de análisis podemos anticiparnos en la detección de errores, ya que se pueden identificar defectos en la base de prueba, tales como inconsistencias, falta de claridad y/o funcionalidades en los requerimientos [3].

DISEÑAR LAS PRUEBAS

Si en el análisis se pretende saber qué es lo que se quiere probar, en el diseño se realiza el cómo probarlo. En este grupo de actividad es donde utilizamos las condiciones de pruebas (aquellas descripciones que nos detallan lo que va a pasar como consecuencias de alguna elección o acción realizada) para crear los casos de pruebas. Durante este proceso de creación de casos de pruebas, es muy habitual aparezcan defectos ya que además de saber los datos necesarios de entrada para realizar el test, también se necesita saber el resultado esperado tras finalizar la ejecución del test y en este punto suelen encontrarse errores, bien sea porque las especificaciones no lo detallan bien o bien porque existen incoherencias. Algunas actividades que se suelen realizar son las de (1) utilizar las condiciones de pruebas para crear los casos de pruebas, (2) dar prioridad a los casos de pruebas, (3) identificar los datos a usar en cada caso de pruebas [3].

IMPLEMENTAR PRUEBAS

En este grupo de actividad verificamos si tenemos todo lo listo para ejecutar las pruebas. Es el nexo entre la actividad de diseño y la de ejecución. Es saber el cómo ejecutar las pruebas una

vez la tenemos escritas. No tiene por qué ser una actividad individual, pues puede combinarse con la actividad de diseño de pruebas o incluso puede formar parte (y llevarse a cabo junto con el diseño de pruebas) de la documentación del grupo de actividad de ejecución. Ejemplo de las principales actividades que aquí se realizan son: (1) crear y dar prioridad a los procedimientos de pruebas (secuencia de casos de pruebas en orden de ejecución y cualquier pre-/pos-condición necesaria para la ejecución de las pruebas) y posiblemente crear script de test para automatizar. (2) crear suites de pruebas con los procedimientos de pruebas ya creados y con los scripts de pruebas. (3) preparar los datos de prueba y verificar si están listo para usar en el entorno de pruebas (4) crear el calendario de ejecución de prueba [3].

EJECUTAR PRUEBAS

Es cuando se ejecutan las pruebas, las suites de pruebas según lo previsto en el calendario de ejecución de pruebas. Como actividades más destacadas se encuentran (1) el registro de la identificación y la versión de lo que se está probando, los objetos de pruebas, la herramienta de prueba y cualquier cosa testeable que esté siendo usado para posteriores referencias en caso de encontrar defectos. (2) realización de la ejecución de las pruebas bien sea manual o de forma automática. (3) Comparación de los resultados reales con los esperados. Realización de informes basados en los defectos encontrados. (4) Registrando el resultado de cada test. (5) Reejecución de las pruebas, bien porque se le han corregido errores o como parte de una prueba de regresión [3].

FINALIZAR LAS PRUEBAS

Cuando el testing haya acabado se recolectan datos con el objetivo de tener una base de conocimiento para una posible reutilización de los artefactos de pruebas generados (scripts, procedimientos de prueba, etc.) en futuros proyectos. Esta actividad puede ocurrir en varios momentos, cuando se termina (o se cancela) un proyecto, al completar cualquier hito del proyecto, al realizar una entrega del sistema, cuando acaba algún nivel de pruebas, en proyectos ágiles cuando termina una iteración. El aspecto clave en este grupo de actividad es asegurarse de que la información no se pierda, incluyendo la experiencia de todos aquellos que estuvieron involucrados. Entre las actividades que aquí se realizan se encuentran: (1) verificar que los informes de los defectos se hayan cerrado, (2) crear un informe general de pruebas, (3) creación de un informe resumen de pruebas para comunicar los resultados de las actividades de las pruebas, (4) utilizar la información recolectada para mejorar la madurez del proceso de pruebas [3].

## 2.3. Clasificación del Testing

Las pruebas se clasifican en dos tipos que se complementan la una con la otra, pruebas estáticas y las dinámicas. Las pruebas estáticas son aquellas que se realizan sin la necesidad de ejecutar código y juega un papel importante a la hora de desarrollar las pruebas dinámicas. Las pruebas estáticas permiten ir haciendo una criba de aquellos errores que a simple vista se pueden detectar sin haber desarrollado código alguno. Es intentar sacar defectos del sistema a través del análisis de las especificaciones y requisitos del usuario. Hecho este primer filtro, el siguiente paso será pasar a trabajar las pruebas dinámicas. Estas pruebas necesitan imperativamente de código para poder realizarse y según qué fuente estemos investigando, estaremos hablando de niveles de pruebas[1][2] o de etapas de pruebas[12], pero en esencia todas ellas coinciden con las mismas actividades a realizar. En los subsiguientes apartados se abordarán estas 2 posibles opciones.

### 2.3.1. Clasificación por niveles

Se definen 4 tipos de niveles y según en el momento que nos encontremos en el proceso de desarrollo del aplicativo se emplearán unos u otros. El término “nivel” indica en qué nos vamos a centrar en un momento determinado del proyecto para llevar a cabo las pruebas y los tipos de problemas que son más susceptibles a suceder. Los niveles se clasifican como:

#### 2.3.1.1. Pruebas de unidad:

Por lo general son pruebas realizadas por el mismo desarrollador que ha escrito el código, aunque lo único necesario es que exista el acceso al código para que se puedan realizar. Se centran en hacer pruebas para verificar el correcto funcionamiento de los módulos que va creando, de forma individual.

#### 2.3.1.2. Pruebas de integración:

Llegado a este punto nos podemos encontramos con 2 subniveles, *integración de componentes* e *integración de sistemas*. De forma similar a las pruebas de unidad, las de integración de componentes corre a cargo del desarrollador, es éste el encargado de garantizar que sus componentes interaccionan bien con el resto de los componentes (también llamados módulos) y/o sistemas.

Las pruebas de integración de sistemas tienen como propósito descubrir los defectos en las interfaces. Esta actividad suele ser realizada por los testers.

#### 2.3.1.3. Pruebas de sistema:

Estas pruebas son muy importantes ya que vienes a garantizar aquello que las de integración no hicieron, probar los flujos de trabajo de principio a fin. El propósito principal en este nivel es probar el sistema como un todo, probar su comportamiento rigiéndose por lo escrito en las especificaciones del producto sin hacer caso del código. Su objetivo es encontrar errores entre todas las interacciones del sistema o aquellos provocados por su entorno. Además de las especificaciones de los requisitos del sistema y el software, otra materia prima para realizar estas pruebas son los casos de usos, informes de riesgos, manuales de sistemas, usuarios y operaciones.

Algunas fuentes en las que he basado mi investigación como “Software Engineering, por Ian Sommerville” y “ISTQB” no están del todo alineados en lo que respecta a estos últimos niveles de pruebas. Lo que para uno es solo pruebas de Sistemas, el otro lo termina dividiéndolo en 2, integración de sistema y sistemas (ISTQB).

#### 2.3.1.4. Pruebas de aceptación.

Aquí ya se prueba el sistema completamente integrado. El propósito de este nivel de pruebas es despejar de dudas al cliente sobre el producto, generar confianza demostrable acerca de la puesta en marcha y el funcionamiento de este. Que el cliente vea con sus propios ojos que el aplicativo cumple con sus expectativas y con todos los requisitos acordado desde el principio del proyecto. Por lo general es responsabilidad del usuario o cliente el llevar a cabo estas pruebas. Lo necesario para llevar a cabo estas pruebas pueden ser los requisitos de usuario, requisitos de sistema, informe de análisis de riesgo, el proceso de negocios entre otros documentos [3].

### 2.3.2. Clasificación por etapas

Esta otra forma o perspectiva de ver y desarrollar las pruebas dinámicas se basa en realizar las diferentes actividades en 3 etapas [12]:

#### 2.3.2.1. Etapa de pruebas de desarrollo:

Este proceso se centra en encontrar defectos y se compone de 3 tipos de pruebas.

- *Pruebas unitarias:*

  Se centran en probar los componentes de forma individual. Un componente equivale a una subrutina, función, procedimiento, una clase o una colección de esos elementos básicos para dar un servicio de más alto nivel [2].

- *Pruebas de componentes:*

  Se encargan de probar las integraciones de los componentes previamente desarrollados.

- *Pruebas de sistemas:*

  Es una prueba de integración de componentes e interfaces, pero a un nivel más alto, lo que significa que se prueba la integración de los sistemas por completo. Por ejemplo, se prueba la integración de dos sistemas desarrollados por equipos de desarrollo independiente.

#### 2.3.2.2. Etapa de pruebas de entrega (o pruebas funcionales):

El objetivo principal de este proceso es generar confianza hacia el producto, convencer a quienes lo adquieran que es lo suficientemente bueno para su uso. Esta etapa se centra en la realización de pruebas con el objetivo de encontrar defectos y también de validar el sistema una vez integrado. Es un proceso de pruebas de "caja negra" donde se tienen en cuenta todos los requisitos del sistema (no solo los del usuario final) para realizar las pruebas. Aquí se evalúan las pruebas funcionales y las no-funcionales.

Se pueden elegir entre 2 métodos para la ejecución de las pruebas funcionales y 1 para las no-funcionales: método de pruebas basadas en requerimientos, las basadas en escenarios y las basadas en rendimiento. A continuación, se describen cada uno de ellos.

- Método de pruebas basadas en requerimientos:

Los casos de pruebas se consiguen diseñar considerando cada requerimiento y de él sacamos un conjunto de pruebas. Este método se basa en pruebas de validación y no en pruebas con el objetivo de encontrar defectos. El objetivo principal es demostrar que el sistema ha implementado de forma correcta los requerimientos. Un requerimiento genera varias pruebas cada una de ellas garantiza su cobertura o sea que se está cubriendo por completo a necesidades funcionales de ese requerimiento. También se deberá mantener un registro de trazabilidad de las pruebas basadas en requerimientos lo que conecta al conjunto de pruebas con los requerimientos [12].

- Método de pruebas basadas en escenarios:

Se crean escenarios para a través de ellos generar los casos de pruebas. Son como una especie de historia que recoge lo habitual que el usuario real suele hacer realizando una actividad típica en su trabajo diario. Estas historias deben ser motivadoras para qué las personas interesadas cuando las lean vean la importancia de que el sistema pase las pruebas.

En un escenario nos podemos encontrar reflejados implícitamente varios requisitos por lo que además de estar verificando requisitos individuales también estamos verificando que la combinación entre ellos no cause fallos [12].

- Método de pruebas basadas en el rendimiento:

Para ejecución de las pruebas no-funcionales se evalúan propiedades emergentes como el rendimiento y la fiabilidad. Se les llama “emergentes” porque los fallos surgen solo si el sistema está integrado en su totalidad. El objetivo fundamental es verificar y validar que el sistema puede procesar la carga para lo que fue diseñado. Para esto se desarrollan pruebas de estrés, las cuales trabajan en los valores límites para conocer el comportamiento del sistema cuando llega al límite de su carga de procesamiento y cuando la sobrepasa.

Nos ayuda a encontrar cuando bajo una elevada carga el rendimiento del sistema comienza a degradarse. Conocido este dato podremos introducir una protección al sistema para que se deshaga de esa carga al pasar ese umbral [12].

#### 2.3.2.3. Etapa de pruebas de usuario:

Esta etapa es un proceso informal centrado solo en la validación del sistema, son los usuarios o clientes de la aplicación quienes prueban el aplicativo haciendo uso de él y dando sus opiniones. Se comprueba que el sistema le gusta al usuario y que satisface sus necesidades. Esta etapa es muy importante ya que es prácticamente imposible para los desarrolladores y testers probar el sistema en el entorno real del cliente bajo situaciones peculiares que pueden dar paso al uso incorrecto del sistema de forma totalmente inconsciente e ilógica. Que se realicen pruebas en un entorno real da una mayor sensación de confianza en aspectos como la fiabilidad, el rendimiento, la usabilidad y la robustez del sistema. Esta etapa se compone de tres tipos de pruebas usuarios conceptualmente diferentes entre ellas. A continuación, se describen cada una de ellas [12].

- Pruebas Alpha:

Mientras el producto se está desarrollando los usuarios de la aplicación trabajan en combinación y de forma directa con el equipo de desarrollo probando el producto en el mismo entorno utilizado por los desarrolladores. Esta forma de trabajar trae como beneficio que los usuarios pueden identificar problemas que para el equipo de pruebas (los testers) pasan desapercibidos, además, gracias a la información suministrada por los usuarios se pueden diseñar pruebas más reales acorde al uso real de la aplicación [12].

p. ej.: Windows Insider Canal de Desarrollo.

- Pruebas Beta:

Culminado el desarrollo del aplicativo o a veces faltando poco para ello se les facilita una primera “release” a un grupo selecto de usuarios de la aplicación para que evalúen el producto en sus propios entornos. Como no todos los usuarios trabajan el mismo entorno, se pueden detectar problemas distintos de interacción entre el aplicativo y las características del entorno usado en cuestión. De este tipo de pruebas se benefician los aplicativos que se corren en diferentes entornos trabajo, ya que es imposible para estos fabricantes de softwares tener réplicas de todos los entornos posibles en el cual el software va a ser usado [12].

p. ej.: Windows Insider Canal beta, Windows Insider Canal de vista previa de versión

- Pruebas de Aceptación:

El cliente prueba el sistema y decide si aceptarlo y pagar por él o no. Estas pruebas se llevan a cabo justo después de terminada la etapa de “release”. Es un proceso compuesto por seis etapas.

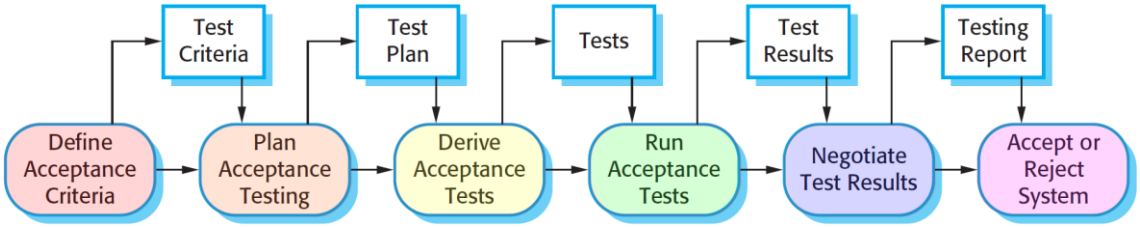


*Figura 7. Proceso de pruebas de aceptación*
(Fuente *Software Engineering,* ed. 10, Ian Sommerville)

Puede parecer que después de todo el trabajo hecho hasta el momento, que el cliente tenga la opción de no querer pagar por el producto, sea un punto de inflexión y decisivo a favor del cliente y en contra del fabricante software. Pero sucede que llegado a este punto el cliente está tan implicado en el proyecto que su mayor interés es que el producto se ponga en marcha cuanto antes, aunque con algunos defectos, pues el coste de no aceptar el producto sería mayor que el de dar solución a cualquier problema existente en el producto. En este momento se acuerda (proveedor y cliente) cómo ir subsanando estos errores o mejoras, mientras el producto está en producción. A continuación, se explica brevemente cada una de las fases y el artefacto que genera [12].

*“Define Acceptance Creteria”*: esta fase debe formar parte del contrato del proyecto y es donde se intentan definir los requisitos en detalle del proceso de desarrollo, tanto el cliente como el desarrollador deben de estar de acuerdo con ellos. En la práctica estas definiciones de requisitos son difíciles de realizar ya que los requisitos suelen cambiar mucho a medida que se

avanza en el desarrollando el sistema. El artefacto generado es el listado de los criterios de aceptación para realizar las pruebas [12].

*"Plan Acceptance Testing"*: Fase donde se planifican aspectos importantes para el desarrollo de las pruebas de Aceptación, tales como los recursos disponibles, el tiempo y el presupuesto a emplear. Se crea el calendario de aplicación de las pruebas y se plantea la cobertura de los requisitos y el orden que se van a realizar las diferentes pruebas a las distintas características del sistema. Se estudian además los riesgos del proceso de las pruebas y se realiza un plan de contingencia. El artefacto generado es el documento de planificación de las pruebas [12].

*"Derive Acepptance Tests"*: Llegado a esta fase, se diseñan las pruebas tanto funcionales como no funcionales para verificar que el sistema cumplirá con los criterios de aceptación del cliente. Las pruebas se diseñarán de forma tal que cubran por completo los diferentes requisitos del sistema, aunque en la práctica es muy difícil establecer de forma objetiva un criterio de aceptación. El artefacto generado son los tests [12].

*"Run Acepptance Tests"*: Esta fase da comienzo a la ejecución de los tests diseñados con anterioridad. El entorno de ejecución debería ser el mismo donde el usuario está trabajando o va a trabajar, pero esta práctica es inapropiada puesto que obstaculiza la labor del trabajador, por tanto, se suele crear un entorno de ejecución de usuario independiente para ejecutar las pruebas. En este caso el artefacto generado son los resultados de la ejecución de cada uno de los casos de pruebas [12].

*"Negotiate Test Results"*: En esta fase se pueden dar dos situaciones, la primera es que todas las pruebas de aceptación hayan pasado y que no haya habido ningún problema con el sistema. Si es así entonces todo acaba de forma exitosa y el sistema puede ser entregado. En caso contrario si se descubre algún tipo de problema o algún fallo en el sistema, entonces el proveedor y el cliente habrán de ponerse de acuerdo y negociar si el sistema es lo suficientemente bueno para ser usado con ese defecto y acordarán el cómo el desarrollador va a arreglar esos defectos. Esta actividad genera como artefacto el informe de los resultados de la ejecución de los casos de pruebas [12].

*"Accept or Reject System"*: En esta última fase se reúnen los desarrolladores con el cliente para decidir si éste acepta o no el sistema. Si el sistema no es lo suficientemente bueno para ser puesto en producción, entonces se realiza un segundo período de desarrollo para arreglar ese problema y al finalizar el mismo se repite la fase de prueba de aceptación [12].

Después de todo lo investigado en este apartado del estado del arte, se ha podido comprobar que el testing no es un proceso ajeno, independiente al proceso de desarrollo del software si no que es una parte complementaria con la cual el ciclo de vida del software tiene que contar. Se ha podido comprobar que ya existes marcos de trabajos formales y estándares que nos informan de cómo hacer un buen testing en una empresa.

## 2.4. Antecedentes.

A la empresa de la cual formaba parte del equipo de QA formado por un conjunto de alrededor de 6 personas para verificar y validar un proyecto de tamaño considerable, le es difícil encontrar recursos humanos con perfiles de tester con el conocimiento apropiado para poder llevar a cabo todas y cada una de las actividades que se necesitan para que todo fluya según lo que establecen las diferentes fuentes de renombres en estos escenarios, principalmente el ISTQB. Por ello, cuando no les queda otro remedio, suelen contratar a personal con escasos conocimientos de pruebas de software o en ocasiones ninguno. Los responsables de gestionar los proyectos a menudo desconocen en profundidad lo que esconde la teoría del test, por lo que esperan que cualquier persona que ocupe el puesto de tester pueda llevar a cabo su función de forma productiva a medida que va ganando experiencia en el proyecto. Es aquí donde empiezan los problemas, pues una vez que los directivos asumen y entienden la necesidad de que el producto a desarrollar sea testeado, creen que con probar la aplicación de forma manual es suficiente, confían en que estas pruebas serán ejecutadas de forma rápida y esperan que esta actividad no atrase a los desarrolladores en la construcción del producto.

La aplicación sobre la cual se trabaja es una aplicación web de tamaño considerablemente grande. El tipo de pruebas que oficialmente se realiza son pruebas emergentes funcionales de aceptación y de forma manual porque la automatización no se plantea, ya que es un tema tabú.

La metodología de desarrollo de software utilizada es Scrum y el método utilizado por los testers para probar la aplicación consiste en elegir de un listado, suministrados por el “Product Owner”, historias compuestas cada una por criterios de aceptación para después, introducirlas en un sistema gestor de pruebas y a partir de entonces, los criterios se analizan, se diseñan (en este momento se escriben, se detallan por pasos, todas las variantes que se estimen convenientes e importantes) los diferentes casos de pruebas correspondientes a cada criterio según la historia, finalmente los casos de pruebas se ejecutan uno a uno, paso por paso, de forma manual, comprobando que la aplicación se comporta en cada momento según van sucediendo los pasos que va mostrando el sistema gestor de pruebas. Durante este último paso pueden producirse las siguientes situaciones: (1) los casos de pruebas pasan satisfactoriamente

según lo diseñado. (2) existe alguna situación de bloqueo al llegar a un paso determinado. (3) en algún paso de ejecución el test falla. La forma de gestionar dichas 3 posibilidades se realiza de la siguiente forma:

(1) Esta situación es la más deseada por todo el equipo, al terminar la ejecución manual todos los pasos del caso de prueba pasan satisfactoriamente y este resultado queda registrado de forma automática en el sistema gestor de pruebas, llagando así al fin de la ejecución y dando paso al siguiente caso de pruebas en la lista.

(2) Dado alguna situación de bloqueo, bien sea por falta de información en las especificaciones, porque el desarrollo depende de un segundo módulo que está siendo desarrollado por otro integrante del grupo o por un error encontrado ajeno al desarrollo actual, se buscará la causa del problema, se registrará el bloqueo en el sistema gestor de pruebas, se informará al responsable pertinente y se esperará a que el caso quede desbloqueado. Llegado dicho momento, se volverá a ejecutar el caso de pruebas desde el principio.

(3) Cuando un caso falla, directamente se registra el fallo en el sistema gestor de pruebas, se informará al desarrollador involucrado en el fallo y se esperará a que el fallo se corrija. Hecho esto, se volverá a ejecutar el caso de pruebas desde el principio.

La siguiente Figura 8 intenta resumir todo el proceso de pruebas:

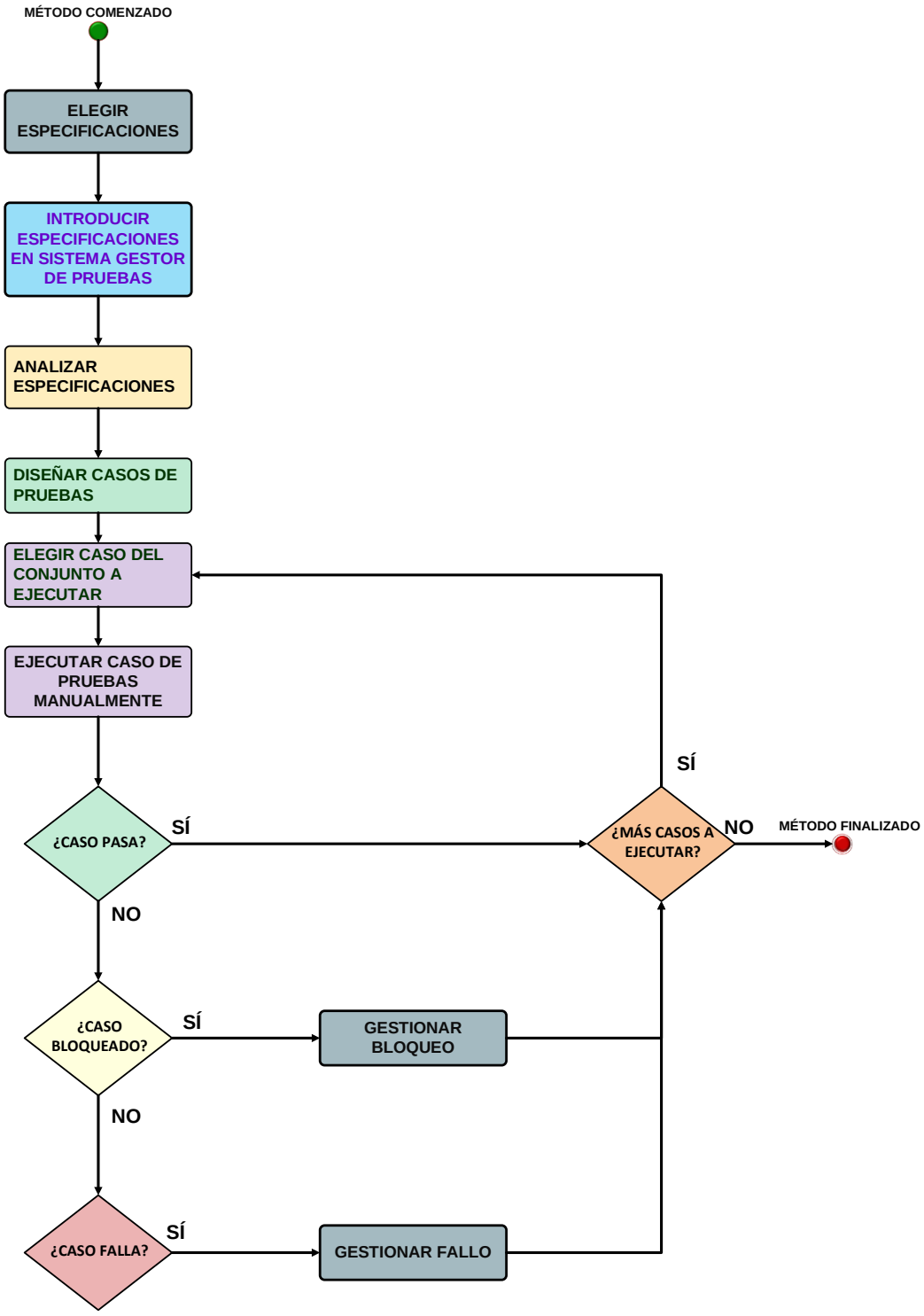


*Figura 8. Modelado AS-IS del proceso de pruebas de la empresa* (Fuente propia)

Las situaciones (2) y (3) hacen que se tenga que ejecutar el caso más de una vez y no siempre que se solucionan alguna de las situaciones es garantía de que todo vaya a ir bien, especialmente con los fallos. A menudo cuando el desarrollador corrige un error, se genera otro que sale a flote cuando el tester vuelve a ejecutar el caso y cuando el desarrollador vuelve a corregirlo, entonces aparece otro error en la siguiente ejecución del caso de prueba y así sucesivamente hasta que se llegan a corregir todos los defectos. Una ejecución de un caso de pruebas puede tardar entre

10min o 30min, pero una buena parte de las veces no es así, pudiendo tardar una media de entre 5 ó 6 horas aproximadamente su ejecución total. Esto sucede porque por cada posibilidad de prueba que tenga el caso, se tienen que configurar los entornos de ejecución de formas completamente distintas. Pongo un ejemplo más cotidiano, imaginémonos un coche de Fórmula 1 al cual se le quieren cambiar los neumáticos según las condiciones meteorológicas, si está lloviendo se emplearán neumáticos más anchos, si el asfalto está seco se utilizarán otros menos anchos para que aumente la velocidad sin perder la estabilidad, en definitiva, el coche se comportará de forma distinta según qué neumáticos tenga, irá más rápido o más lento, será más estable o menos, etc. Para muchos es sabida la velocidad con que se cambian las ruedas de esos coches en plena carrera, un equipo formado por un conjunto de alrededor de 20 personas es capaz de cambiar las 4 ruedas en menos de 2s [16].

*Figura 9. Equipo de Fórmula 1 cambiando neumáticos*
*(Fuente http://www.ellatinoarkansas.com/content.cfm?ArticleID=12552)*

Ahora bien ¿qué pasaría si en vez de ese conjunto de personas, las ruedas las cambiara una sola persona? Aunque en la competición estuvieran de acuerdo en esperar a que los neumáticos se cambiaran, el público presencial y el televidente se aburriría de tener que esperar por cada cambio que se hiciera y el espectáculo deportivo perdería toda la emoción que suele transmitir. De este mismo modo es como se siente uno cuando tiene que volver a ejecutar estas pruebas manuales con diferentes contextos, entornos, usuarios, licencias, varios campos a cumplimentar, etc. para obtener el resultado esperado. En cambio, si estas pruebas estuvieran automatizadas con alguna herramienta disponible hoy en día como bien aconsejan los diferentes marcos en la actualidad, entonces la velocidad con que tales pruebas se ejecutarían aumentaría. Como mencioné más arriba en la motivación, durante mis estudios impartí la asignatura P.P.S.S y en ella conocí la herramienta Selenium Webdriver que nos permite automatizar pruebas funcionales en aplicaciones web. Ya fuera con esta herramienta u otra similar, tendríamos la ventaja de estar ejecutando un procedimiento de pruebas, mientras el

programa, por su parte estaría ejecutando el mismo procedimiento de pruebas en otra instancia, pero con otros parámetros al mismo tiempo que yo. Y si ese programa pudiera lanzar varias instancias a la vez con diferentes parámetros y en diferentes entornos, estaríamos hablando de resultados tan eficientes como los del cambio de neumáticos en las carreras de Fórmula 1.

# 3. Objetivos

En esta sección se va a describir el objetivo general de la propuesta y se determinarán los subobjetivos.

Los marcos globales de trabajo (según se desprende del estado del arte) consideran el testing una fase más del ciclo de vida del software, por tanto, debe estar integrado en su desarrollo y se le deben dotar de mecanismos que permitan su realización de forma eficiente y eficaz.

El objetivo principal de este TFM es automatizar la actividad manual de ejecución de los casos de pruebas de una PYME. Esto conlleva a alcanzar los siguientes subobjetivos:

- Mejorar el proceso del testing en la empresa definiendo cómo se han de realizar los procesos de testing apoyándome en marcos de referencia y estándares que me permitan ver en qué punto se pueden realizar las mejoras.
- Mejorar la eficiencia en la forma de trabajar de los tester, reduciendo el tiempo empleado por los ellos en la ejecución de los casos de pruebas.
- Aumentar la calidad del software, ya que los testers tendrán más tiempo libre disponible para desarrollar y ejecutar más pruebas en el mismo período de tiempo.
- Minimizar los riesgos existentes a la hora de ejecutar los tests de forma manual.
- Buscar formas de formalizar el testing, estableciendo normas y reglas, para evitar que el desarrollo del software tenga un impacto mínimo en los tests scripts ya almacenados.
- Definir el escenario y entornos de trabajo, herramientas y protocolos.
- Analizar las posibles herramientas para la automatización y seleccionar la más adecuada.
- Realizar una validación de la propuesta (midiendo los indicadores clave de rendimiento (KPIs) y demostrando que mejoran después de automatizar).
- No generar gastos económicos extras a la empresa.

# 4. Metodología

El proyecto de la empresa a la cual yo formaba parte estaba formado principalmente por un grupo de desarrolladores y otro de testers, al cual yo pertenecía, encargado de validar el desarrollo que se hacía, pero el método que voy a emplear para la realización de esta mejora será la de ver a mi grupo como una empresa externa, independiente a la empresa real y para comenzar el grupo lo integrarán 2 personas, un ingeniero informático y un técnico informático. Los motivos que me llevan a hacer esto es porque me es imposible hacer el análisis del testing de forma integral en la empresa ya que no trabajo en otras secciones, no tengo la visión general de todo el proyecto para poder controlarlo, ni tampoco sé cómo lo gestionan los gestores, pero lo que sí controlo es la gestión de mi departamento de testing. De esta forma podré ejercer un mayor y mejor control en los análisis que será necesario realizar.

Los grandes marcos de trabajo afirman que una de las formas de mejorar el proceso del testing en la empresa es mediante la automatización en la fase de ejecución de los casos de prueba. Por ejemplo, el TMM*i* aconseja, en la cuarta área de su nivel 2 "Diseño y ejecución del test" usar como herramienta de soporte a dicha área la automatización utilizando test scripts (código que conforma el test) predefinidos:

*"According to the defined execution schedule, the test cases are run either manually using documented test procedures and/or via test automation using pre-defined test scripts.[7]"*

La metodología de trabajo que voy a emplear se dividirá en 3 fases. La primera estará relacionada con la automatización y verificación del testing, la segunda con su validación, o sea comprobar que realmente se han conseguido satisfacer y cumplir los objetivos deseados, y la tercera se relacionará con la implantación de la automatización del testing en la empresa llevando a cabo un Plan de puesta en marcha que dé solución al actual problema.

## 4.1. 1ª Fase – Iniciación tecnológica

Partiendo del supuesto que mi empresa externa se encuentra en el nivel 2 del marco TMM*i* y dado que se necesita una herramienta que sirva para automatizar las pruebas de aceptación, se elegirán herramientas adecuadas al tipo de software sobre el que la empresa realiza el proceso de testing, pues el tipo de software condiciona qué herramientas se pueden utilizar. Tras la elección se dedicará un tiempo para documentarnos sobre la herramienta y la preparación del entorno para integrar la herramienta. Una vez hecho esto se procederá a hacer pruebas preliminares en el entorno de trabajo para comprobar que la herramienta se ajusta a nuestro objetivo. Verificado y validado este importante paso solo quedará aprender bien la

herramienta y coger un poco de destreza. El motivo de esto último es porque no tenemos a nuestra disposición un experto en la herramienta, pero sí tenemos al personal capacitado para que la curva de aprendizaje sea lo más empinada posible.

## 4.2. 2ª Fase – Implementación tecnológica

Una vez que se adquiera un poco de destreza con la herramienta se procederá a la creación de un documento de tipo "manual de buenas prácticas". Será una especie de contrato interno entre el departamento de desarrollo y el de testing. El propósito principal de este documento es evitar que una vez que se realicen los tests scripts y queden validados, estos no se vean perjudicados por un cambio de nombre o estructura de los elementos que conforman la página web ya que esto conllevaría a que los test scripts ya creados con anterioridad fallecen en posteriores ejecuciones. Los desarrolladores deberán comprometerse a actualizar dicho documento cada vez que vayan a hacer modificaciones que afecten a algún identificador, estructura de la página o elementos de ella.

Terminada la redacción del manual, se realizar un estudio comparativo donde se vea la eficiencia de la automatización frente al uso manual en la ejecución de los casos. Para esto se establecerán KPIs que nos permitirán medir los efectos de la ejecución de los casos de pruebas antes de automatizar (modo manual) y después al implantar la automatización, verificando de esta forma si lo que se propone es adecuado e influye o no en la mejora de la calidad del software. Una vez obtenidos los resultados, entonces en posteriores mejoras del proceso del test se podrá ir extrapolando esta forma de trabajar hacia los diferentes niveles de prueba que existen en las fases de desarrollo del proyecto para ir realizando el diseño de pruebas en paralelo, p.ej. desde las etapas tempranas en la toma de requisitos hasta la fase de entrega, como mandan los grandes marcos de trabajo. Tras todo esto, se realizará un informe donde se destaquen aspectos positivos y negativos encontrados en el proceso de la implantación de la automatización y el uso de la herramienta.

## 4.3. 3ª Fase – Plan de implantación de la automatización del testing en la empresa

Momento en que se llevará a la práctica toda la teoría y comprobaciones vistas y realizadas. Se seguirán unas pautas para que la implantación de la automatización del testing sea lo más aceptada posible entre los integrantes del proyecto, especialmente los testers que tendrán que lidiar día a día con esta nueva forma de trabajar. La resistencia al cambio será uno de los puntos más importantes y difíciles a gestionar. Esto es debido a que muchos recursos humanos no

siempre están de acuerdo en dejar su zona de confort, para adentrarse en nuevas formas de trabajar que podrían poner en compromiso su estatus profesional. Por ello, se instruirá al trabajador dotándole del conocimiento necesario para que la implantación sea lo menos traumática posible para el personal que no esté muy de acuerdo y también para los que lo estén.

También se realizarán informes en aquellos casos en los que el testing automático no esté proporcionando el rendiendo esperado y se buscarán formas de realizar las mejoras oportunas.

Por último, se hará una validación de los pasos seguidos en la propuesta con uno de los marcos de trabajo estándares para la mejora e implantación del testing vistos en el estado del arte, el TMMi. De mostrando así que la propuesta va alineada tanto con la empresa como con los maracos.

## 4.4. Planificación

Para llevar a cabo el seguimiento de las actividades de las faces crearemos un calendario. Esto será lo que determine el resto de las actividades de la Propuesta de la Solución. Una segunda herramienta será utilizada, MS Project Professional 2019. El motivo de su elección es porque ya poseemos conocimiento del funcionamiento de la misma y por otro lado nos permitirá llevar un control exhaustivo de las tareas a realizar.

# 5. Propuesta de la solución

En este apartado comenzaré a explicar **cómo** dar solución a los objetivos que se pretenden alcanzar, para ello se propone transformar el actual proceso de testing, en un proceso cercano a los estándares de los marcos de desarrollo, introduciendo cambios en el proceso del testing. Después de revisar los grandes marcos, se adoptará uno de los puntos recomendados para mejorar el testing en el proceso de ejecución de las pruebas, la automatización de los procedimientos de pruebas.

Nuestra empresa dará solución a otra empresa que necesita que la calidad del software que está construyendo sea la mayor posible con costes mínimos. Nosotros hemos sido contratados por dicha empresa para realizar las pruebas de aceptación de su producto. Se trata de una aplicación web de gestión de trabajadores para diversos sectores. En ella se han de rellenar campos, comprobar accesos a diferentes pantallas si se cumplen ciertos parámetros de permiso, simular horarios y un sinfín de funcionalidades. Este aplicativo, objeto de prueba (lo que está siendo probado), tiene la característica de tener que correr en 3 tipos de navegadores distintos: Chrome, Firefox y Edge. Esto significa que deberá existir compatibilidad 100% con esos navegadores ya que los usuarios de la aplicación solo utilizan esos navegadores. La forma actual de realizar las pruebas de aceptación para validar las diferentes funcionalidades de la aplicación es de forma manual, esto es, si hay que probar un formulario en una pantalla con un navegador dado, esta actividad tendrá que repetirse en los 2 tipos diferentes de navegadores restantes para asegurarse de que no exista incompatibilidades entre ellos, ya que en muchas ocasiones las hay.

Ya que nosotros, como empresa externa nos hemos marcado el objetivo principal de automatizar la gran mayoría de pruebas que hagamos, comenzaremos nuestra actividad eligiendo una herramienta que satisfaga las necesidades de automatización del tipo de pruebas que necesitamos realizar.

## 5.1. Elección de herramientas y documentación.

Este apartado da comienzo a la 1ª Fase – Iniciación tecnológica del apartado Metodología. Como allí se indica, en esta sección realizaremos la elección de la herramienta para llevar a cabo la automatización de las pruebas funcionales. Esta herramienta deberá ser capaz de conseguir en cada ejecución la simulación del flujo de trabajo que un usuario sea capaz de hacer con la aplicación y de devolver informes sobre cada prueba ejecutada. Con estos informes se podrá llevar un control de la evolución del desarrollo del proyecto como se menciona en el grupo de

actividades del proceso básico del testing “Monitorear y Controlar las Pruebas” mencionado en el Estado del arte.

En la comunidad del testing encontramos una gran variedad de herramientas que nos permiten realizar nuestra tarea de automatización y de forma gratuita, p. ej. Puppeteer[8], Katalon Studio[9], TestNG[10], Selenium Webdriver, etc. Vista la complejidad en el aprendizaje que presenta cada una de ellas, optaremos por quedarnos con la más conocida para el equipo, SeWe ya que tenemos conocimientos previos de su funcionamiento. Además, esta herramienta es una a las que más le hacen referencia en la comunidad del testing. Creada como código abierto de la unión de los proyectos Selenium y Webdriver en 2009, soporta a la mayoría de los navegadores más usados, como Chrome, Firefox, Edge, IE, etc. Está respaldada por una gran comunidad de usuarios y desarrolladores. Permite ser implementada principalmente en los siguientes lenguajes de programación, Ruby, Python, C#, JScript y Java. Por comodidad, haremos la implementación con el lenguaje de programación Java.

También utilizaremos la biblioteca “library” JUnit, un API de java que a través de sus clases y métodos nos ayudará a implementar y ejecutar las pruebas. Será la encargada de devolver el informe del estado de los test una vez ejecutados y de preparar las “pre” y “pos” condiciones de cada test.

Trabajaremos con un proyecto base centralizado desde donde los testers se encargarán de descargarse el nuevo proyecto, subir actualizaciones y actualizarse con los nuevos cambios existentes. Para la gestión de cambios nos apoyaremos de la herramienta control de versiones Git [20], en ella se alojará la información del proyecto base que, a continuación, en el siguiente apartado se describirá.

Con respecto a la documentación se ha recopilado bibliografía para generar conocimiento, libros como “Selenium WebDriver Practical Guide [10]” y también los innumerables blogs de la comunidad de SeWe nos han servido como referencia de apoyo para, con todo ello, crear una página web en nuestro sistema centralizado de administración de conocimiento empresarial Confluence [19] que nos sirva de guía de referencias de ayuda. La herramienta Confluence permite a los equipos crear documentación de forma colaborativa online. Además, se ha creado un directorio compartido llamado “\materialGeneradoSelenium” donde se alojará todo el material electrónico de interés y ayuda (libros en pdf, Word, etc.). De esta forma todo el personal interesado podrá tener acceso a la información y podrá también aportar conocimiento agregando más información.

## 5.2. Diseño del proyecto base

Hemos diseñado un proyecto base que será el punto de partida para poder desarrollar los tests. El objetivo principal es que todos los implicados en el desarrollo de las pruebas automáticas trabajen con el mismo proyecto y que conforme se van creado los tests de las diferentes funcionalidades, todos tengan acceso al código ya generado. El proyecto se creará en un Entorno de Desarrollo Integrado, más conocido como "IDE" y como la experiencia es también un plus, utilizaremos Eclipse en su versión Sprint Tool Suite 4. Nos ha parecido buena idea que el proyecto tenga un formato estándar, para ello lo construiremos utilizando la herramienta Maven la cual creará un esqueleto (arquetipo) estándar con la estructura necesaria para ejecutar test. La Figura 10 muestra dicha estructura.

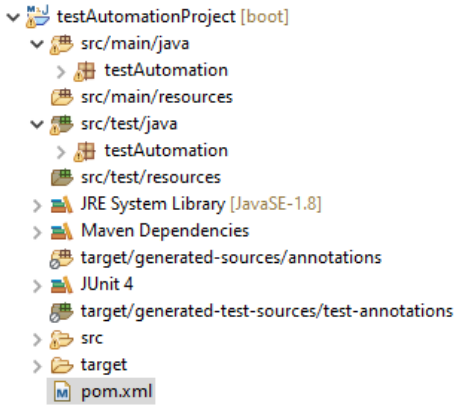


*Figura 10. Estructura estándar del proyecto generada con Maven (Fuente propia)*

Al generar un nuevo proyecto Maven en su raíz se crea un fichero de configuración denominado "pom.xml" el cual habrá que modificar para agregar todos los conectores (en forma de dependencias) necesarios para que se pueda ejecutar el proyecto. Como dependencias principales tendremos el API JUnit y a Selenium que nos permitirá utilizar sus interfaces y en nuestro caso la de Webdriver que simula diferentes tipos de navegadores. La Figura 11 muestra las dependencias mencionadas dentro del fichero pom.xml.

```
<!-- ================ SCOPE TEST ================ -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-api</artifactId>
    <version>3.10.0</version>
    <scope>test</scope>
</dependency>
<!-- ================ end of SCOPE TEST ================ -->

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-api</artifactId>
    <version>3.10.0</version>
</dependency>
</dependencies>
```

*Figura 11. Dependencias de JUnit y Selenium en el fichero pom.xml (Fuente propia)*

Se ha de destacar que esta actividad será realizada por ambos integrantes del grupo, aunque el mayor peso recaerá en el técnico. El ingeniero en ocasiones bridará ayuda de tipo informativo al técnico.

## 5.3. Prueba de la herramienta y adquisición de destreza.

Ya con el entorno preparado, comenzaremos a crear el código para probar y verificar que la herramienta, SeWe funciona y se integra bien con la aplicación objeto de prueba. Para una primera toma de contacto con SeWe se ha realizado un experimento sencillo. Hemos utilizado como objeto de prueba la página web de Google. Se ha implementado un script test simple que se encarga de abrir la página web de Google y hacer una búsqueda simple dada una cadena de caracteres. Esto ha sido nuestro primer programa "Hola mundo". La Figura 12 muestra conceptualmente la prueba ejecutada.

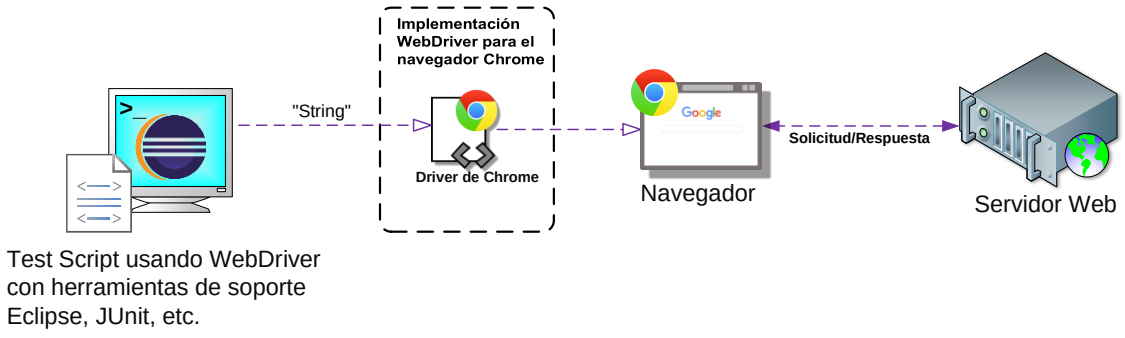


*Figura 12. Automatizando una búsqueda simple en Google (Fuente propia)*

Una vez conseguida esta primera prueba, hemos procedido a realizar una segunda, esta vez con los 3 navegadores de forma secuencial Google, Firefox y Edge. De esta forma nos hemos asegurado de que todo marchará bien cuando tengamos que hacer lo mismo con la aplicación de nuestro cliente. La Figura 13 muestra conceptualmente las pruebas ejecutadas.

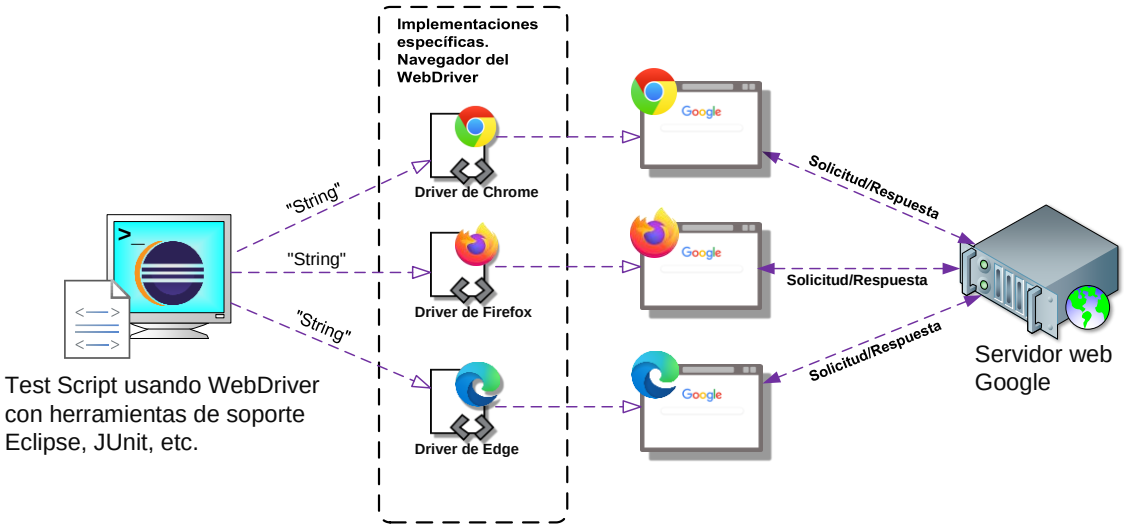


*Figura 13. Automatizando 3 navegadores - Búsqueda simple en Google (Fuente propia)*

Superada esta primera fase y habiendo comprobado que la herramienta se ajusta a nuestros objetivos y satisface nuestras expectativas, esto es, sirve perfectamente para hacer pruebas en navegadores webs, principalmente los que nuestro cliente demanda, solo nos queda comenzar la codificación de nuestros tests con la aplicación web del cliente, leer la suficiente documentación para adquirir destreza en la realización de los tests y la puesta en marcha de la implantación del testing en el proyecto de nuestra empresa.

## 5.4. Creación de la "Guía del desarrollador para la automatización del proyecto"

Llegado a este punto se inicia la "2ª Fase – Implementación tecnológica" con la creación de un manual que será clave en el proceso de desarrollo del testing automático. Para lograr un proceso de testing homogéneo, sistemático y automatizable, el departamento de desarrollo ha de seguir un conjunto de formalidades y convenciones que nosotros, desde el departamento de testing les vamos a suministrar a través del acceso a nuestra plataforma de conocimientos Confluence. Similar al apartado creado para nuestra documentación interna de ayuda, crearemos otro donde marcaremos las pautas a seguir… les suministraremos una especie de manual de buenas prácticas denominado "Guía del desarrollador para la automatización del testing en el proyecto". Será una especie de contrato que cada desarrollador tendrá que comprometerse a cumplir. Allí se recogerán un listado de reglas que se deberán respetar durante el desarrollo y 3 tablas en las que se podrá editar para agregar y/o modificar información relativa a los elementos de la página web, sus identificadores, sus eventos y su estructura. En la siguiente Figura 14 podemos apreciar un ejemplo de un identificador "input" dentro de la estructura de la página web de inicio al campus de la Universidad de Alicante:

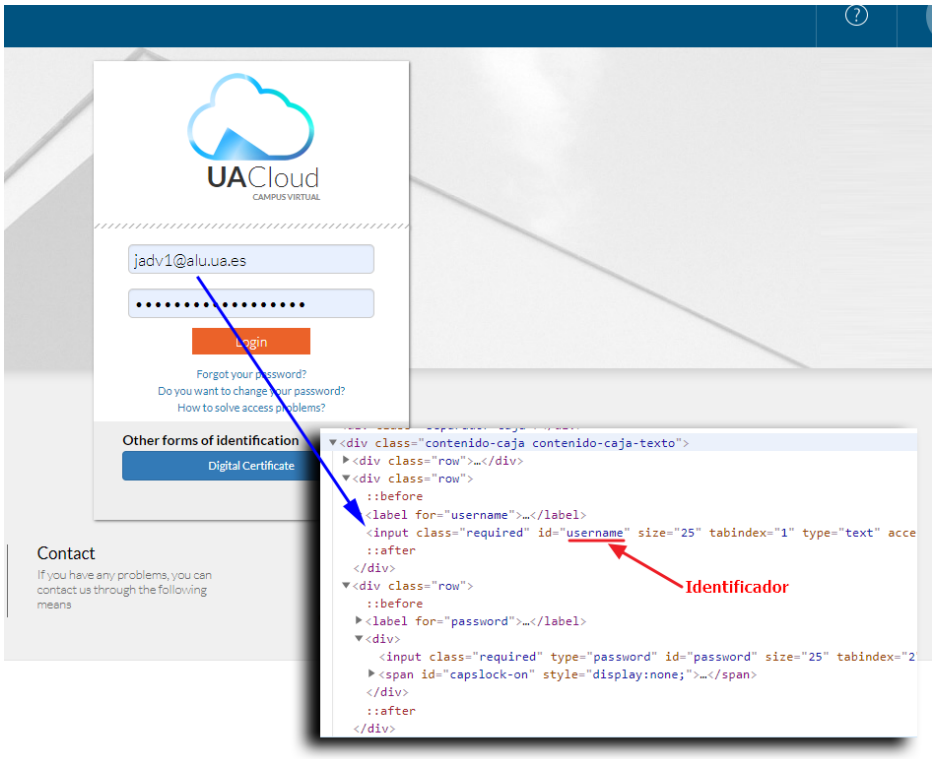


*Figura 14. Página de inicio del campus de la Universidad de Alicante (Fuente propia)*

A continuación, se enuncian las reglas y seguidamente muestran los prototipos de las tablas:

R1. Todos los identificadores de nombres como clases, ids y atributos de los elementos webs (div, inputs, span, etc) que pertenezcan a algún test script creado deberán de figurar en la Tab.ID.

R2. Los identificadores de nombres que aparezcan en la Tab.ID. deberán permanecer inmutables, pero de ser necesaria su modificación o eliminación deberá la Tab.ID. deberá recoger esos cambios.

R3. Siempre que se pueda y en la medida de lo posible se deberán usar identificadores de nombres (atributos ids) en los principales elementos webs para que con SeWe se pueda identificar a los elementos webs de forma más rápida y eficiente.

R4. Los cambios o eliminación de las estructuras de los diferentes componentes que pertenezcan a algún test script creado deberán ser recogidos en la Tab.E.C.

R5. Se deberá identificar en la Tab.E.E. qué evento es lanzado por la acción sobre un cierto elemento y la localización del elemento, si el test script desarrollado depende del evento generado.

Las siguientes tablas serán rellenadas de forma progresiva por los testers al crear los tests. A partir de entonces, posteriores modificaciones serán llevadas a cabo por los desarrolladores o

testers. Cada vez que un test, p.ej. haga referencia a algún elemento a través de su identificador para su localización, a partir de ese momento el identificador se convierte en inmutable a los efectos del desarrollador. El desarrollador pues, tendrá que asegurarse de que cada vez que vaya a modificar un identificador id, el mismo no exista ya en la tabla de identificadores, de existir tendrá que reflejar el cambio. De esta forma se podrán evitar los riesgos de fallos en la ejecución de los tests provocados por las modificaciones realizadas por los desarrolladores durante la construcción del código "html" y sus futuras modificaciones.

La Tabla 1 se denominará "Tabla de identificadores" (Tab.ID.), en ella se recogerán los elementos webs a los que, una vez desarrollados los tests, se hace referencia cuando son ejecutados. Por ejemplo, si un test necesita localizar de la Figura 14 el elemento input con id "username" o un div con clase "row", en la tabla se recogerá de la siguiente forma:

| Tabla de identificadores | | | |
|---|---|---|---|
| Elementos | Id | Clases | Referencia de localización |
| input | username | - | Página de inicio |
| div | - | Row | Página de inicio |
| … | … | … | … |

*Tabla 1. Prototipo Tab.ID.*

La Tabla 2 "Tabla de estructura de componentes" (Tab.E.C) creada y rellenada en un principio por los testers reflejará los cambios llevados a cabo por un desarrollador en la estructura de los componentes existentes en la página. Un componente será todo aquel conjunto de elementos web que puede ser reutilizado en distintas páginas de la aplicación y cuyos componentes siguen siempre el mismo orden jerárquico entre ellos. En la columna "Componentes" se escribirá el nombre identificativo del componente en cuestión. Si la estructura interna de un componente cambia, se indicará en la columna "Modificada" con una 'x'. Si el componente es eliminado, la columna "Eliminada" será la que sea marcada con una 'x'.

| Tabla de estructura de componentes | | | |
|---|---|---|---|
| Componentes | Estructura | | Localización |
| | Modificada | Eliminada | |
| Modal Alerta de actividades | x | | Ventana de actividades |
| … | … | … | … |

*Tabla 2. Prototipo Tab.E.C.*

En la Tabla 3 "Tabla de Elementos y Eventos" (Tab.E.E.) se mostrará la relación que existe entre el elemento web y el evento que éste lanza. Por ejemplo, en ocasiones interesa saber si al

pasar el ratón por encima de un elemento etiqueta “label” se mostrará cierto tipo de pantalla, ventana o mensaje de ayuda. En este caso un tester recogería esta información una vez creado el test script de la siguiente forma:

| Tabla de Elementos y Eventos | | | |
|---|---|---|---|
| Elementos | Acción | Evento que lanza | Localización |
| label | Pasar puntero por encima | Abrir ventana de ayuda emergente | Ventana de inicio |
| … | … | … | … |

*Tabla 3. Prototipo Tab.E.E.*

## 5.5. Análisis del proceso de ejecución del testing en un caso de estudio

Una vez creado el manual de buenas prácticas, realizaremos una simulación de cómo afectarían los tiempos en la automatización y ejecución de las pruebas más frecuentes realizadas por los testers, frente a la forma manual actual. Para eso vamos a describir cómo se realizaría la ejecución de un típico procedimiento de pruebas actualmente en nuestra empresa modelo. Para realizar este proceso de ejecución de los tests, los testers acceden al sistema gestor de pruebas llamado “Squash TM” [18] que almacena todos los casos de pruebas a ejecutar. Para dar comienzo se accede a la pantalla de ejecución como muestra la Figura 15:


*Figura 15. Sección Ejecución “Campaign Workspace” en Squash TM (Fuente propia)*

Es aquí donde comienza el proceso manual de ejecución tan mencionado en este trabajo y es aquí también donde se almacenará el resultado de la ejecución de cada test, la frecuencia de fallos de cada uno de ellos, la persona que ejecutó el test, la hora, etc.

### 5.5.1. Proceso de ejecución manual

Para ejecutar un test hay que proceder de la siguiente forma:

En la sección Ejecución “Campaign Workspace” damos un clic en el botón “play” como muestra la Figura 16:

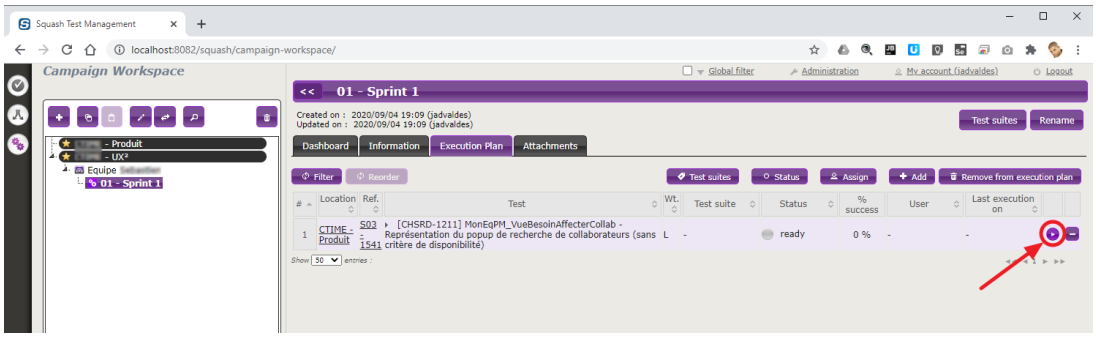


*Figura 16. Inicio del Proceso Ejecución (Fuente propia)*

Esto genera otra pantalla, Figura 17, desde la cual el test diseñado es llamado y ejecutado paso a paso. A continuación, damos comienzo a la ejecución presionando el botón “Begin” lo que nos llevará a la interfaz de la Figura 18:

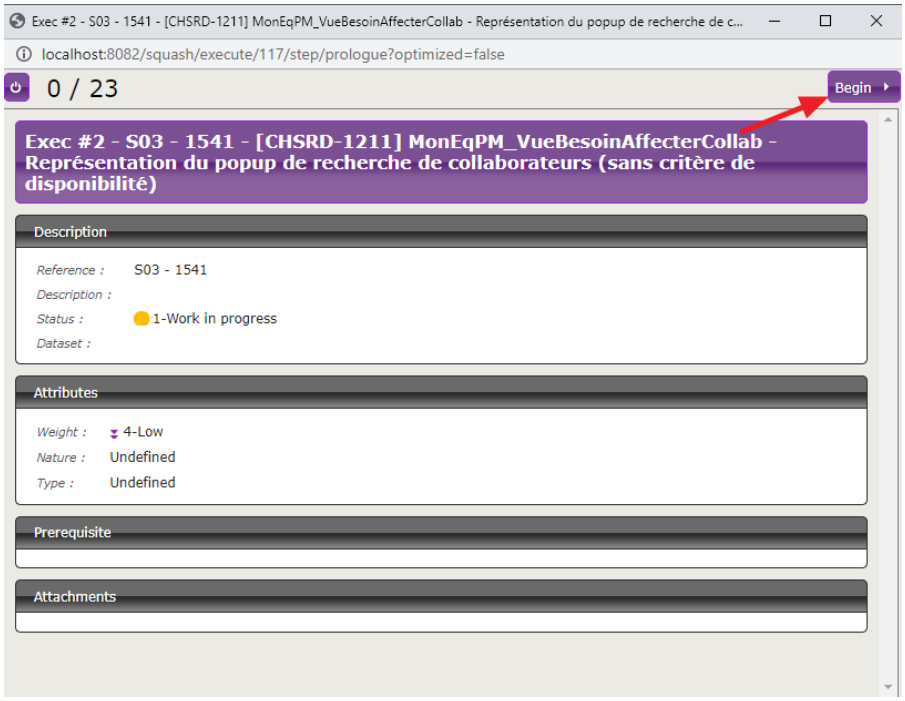


*Figura 17. Ventana de llamada al test diseñado (Fuente propia)*

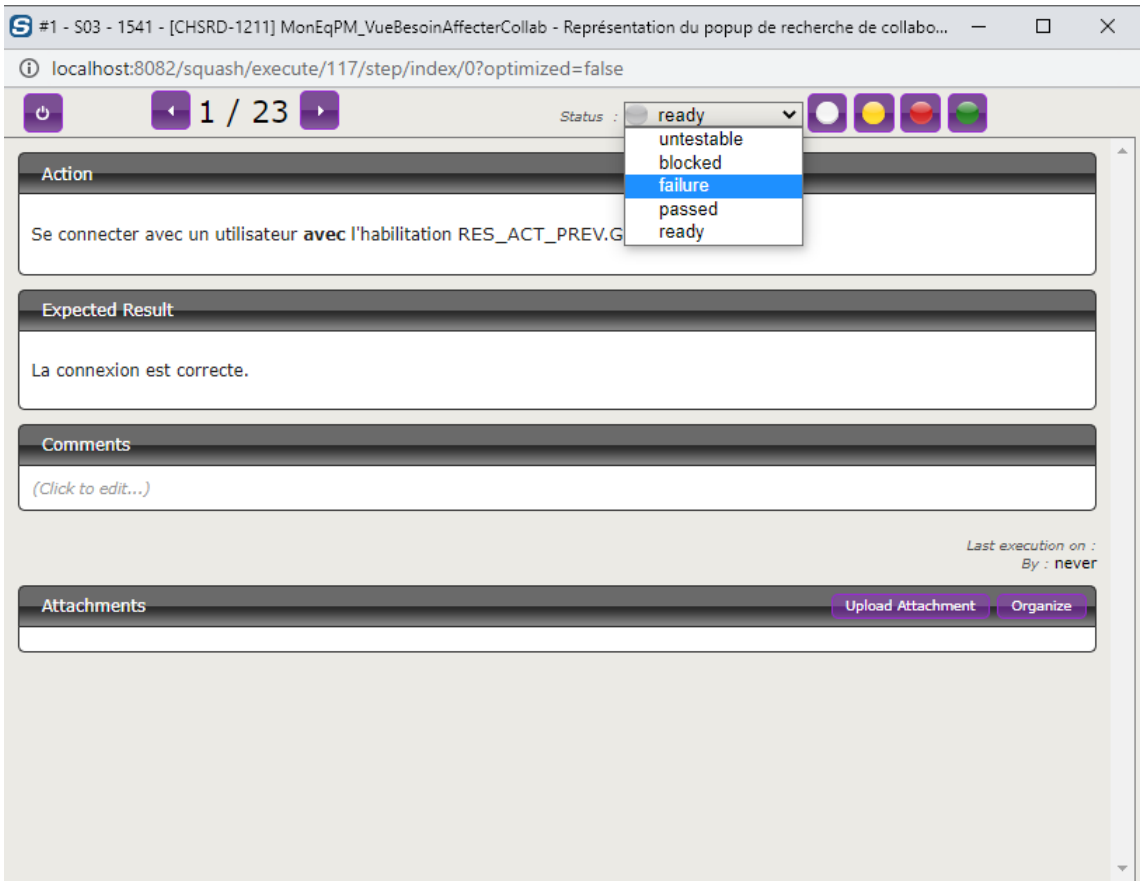


*Figura 18. Primer paso del caso de pruebas a ejecutar (Fuente propia)*

En esta pantalla se aprecian 4 secciones principales que son las que marcan el ritmo de trabajo del tester: “Action”, “Expected Result”, “Comments” y “Status”.

**Action**: es la acción a realizar en la aplicación y como consecuencia habrá un resultado.

**Expected Result**: es el resultado esperado de esa acción, o sea lo que debería suceder según las especificaciones cuando se realiza dicha acción.

**Comments**: En caso de que la acción genere un resultado no esperado, se introduce información explicando el porqué el tester afirma que es un fallo.

**Status**: Son los estados por los que el caso de prueba puede atravesar. El tester ha de etiquetar el caso según su juicio.

- Un caso puede que sea imposible de ejecutar p.ej., porque expone tan solo una precondición dentro del procedimiento del test y en este caso se le asigna el estado “Untestable”.
- Un caso puede que sea imposible de ejecutar p.ej., por falta de información en las especificaciones y en este caso se le asigna el estado “blocked”.

- Un caso puede que sea imposible de ejecutar por un error en la aplicación y en este caso se le asigna el estado “failure”.
- Si el caso ejecutado devuelve el resultado esperado, entonces se le asigna el estado “passed”.
- “ready” es el estado inicial.

Es necesario destacar que este proceso de ejecución lleva consigo implícitamente posibles puntos de errores. Dado que es el propio usuario “tester” quien siguiendo los pasos de los casos de pruebas deberá introducir datos, en su mayoría no especificados en el diseño, en la aplicación objeto de prueba. Tomemos p.ej. una prueba con las mayores tasas de ejecución del proyecto “Probar el acceso a determinadas ventanas, con permisos y sin ellos”.

Esta prueba lleva consigo precondiciones y postcondiciones, esto es, cada vez que tenemos que realizar la prueba habrá que preparara ciertos parámetros para después poder probar lo que queremos, después de haberlo hecho hay que volver a dejarlo todo como estaba antes.

#### 5.5.1.1. Precondición configuración de parámetro:

Primero tendremos que logeamos en la aplicación como se puede apreciar en la siguiente Figura 19:

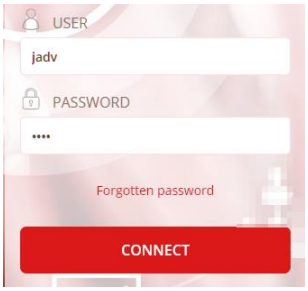


*Figura 19. Ventana de acceso a la aplicación (Fuente propia)*

Esta acción nos traslada a la ventana principal de la aplicación mostrada en la Figura 20:

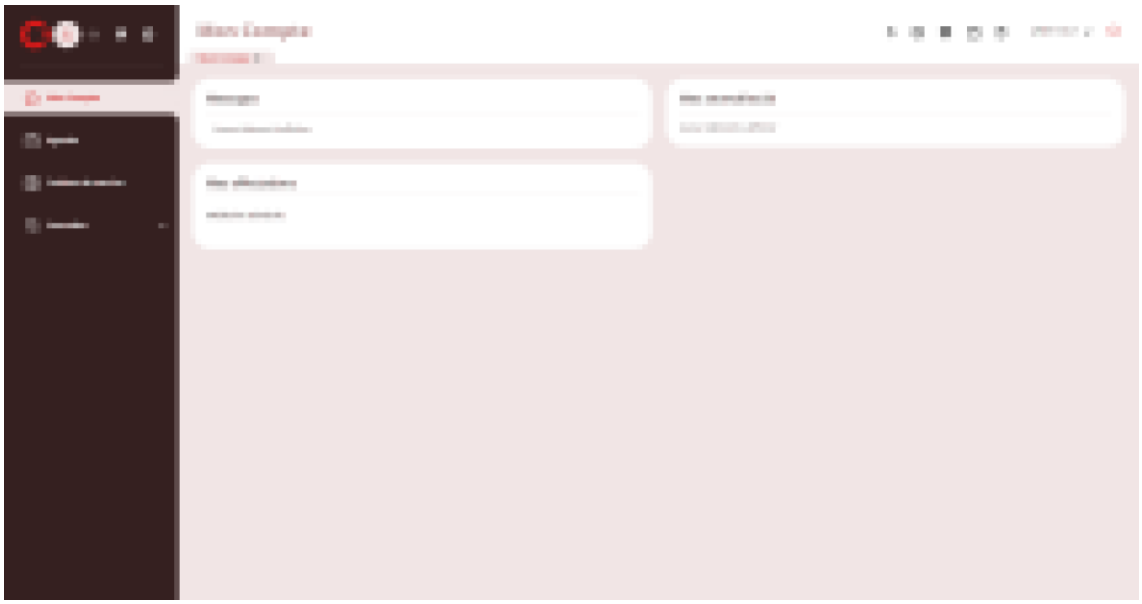

*Figura 20. Ventana principal de la aplicación (Fuente propia)*

Una vez dentro debemos acceder a la página de configuración de permiso Figura 21, allí seleccionar tu perfil, dentro de tu perfil seleccionar de un listado, el nombre del parámetro deseado y en su panel de configuración, agregar el valor necesario p.ej. “1” para que permita el acceso a la ventana de validación.

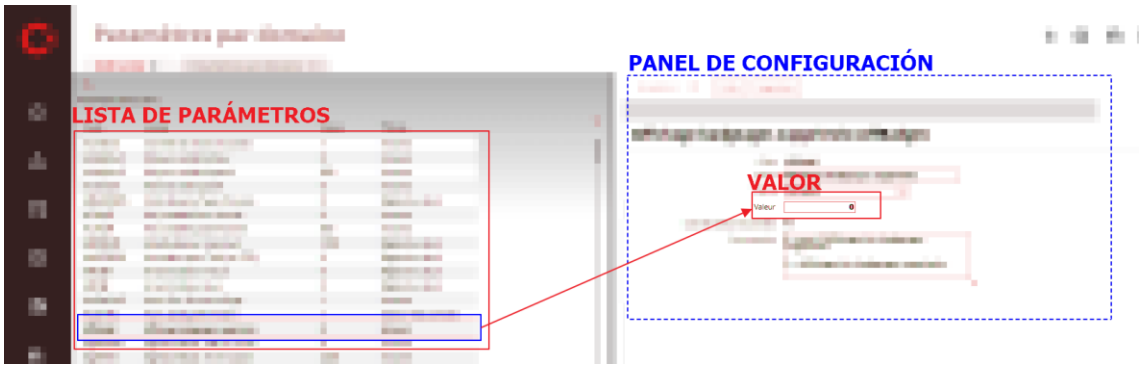


*Figura 21. Página de configuración (Fuente propia)*

Seguidamente hay que desconectarse de la app para que los cambios surtan efectos.

### 5.5.1.2. Caso de prueba:

Nos volvemos a conectar, accedemos a la página de validación Figura 22 y se comprueba que es posible el acceso.

*Figura 22. Página de validación (Fuente propia)*

Para probar el segundo caso, cuando no es posible en acceso, habrá que volver a la ventana de configuración realizar el proceso de selección en el listado del parámetro, cambiarlo por el valor apropiado p.ej. “0” y aceptar los cambios. De nuevo nos desconectamos de la app para que los cambios surtan efectos. Nos volvemos a conectar, accedemos a la página de validación y comprobamos que no existe el acceso a esa página.

#### 5.5.1.3. Postcondición configuración de parámetro:

Comprobado este último paso habrá que restablecer el parámetro a su estado inicial ya que su configuración afecta a toda la aplicación y otro equipo de desarrollo que pudiera estar necesitando trabajar con esa configuración, quedaría incapacitado para continuar con su desarrollo.

La siguiente Tabla 4 muestra la frecuencia con que se ha realizado la prueba, o sea el número de veces que se ha ejecutado (*f* eje.), frente al tiempo que se tarda en realizar esta prueba en 3 navegadores (*t*x3 Nav.) en un momento ideal (o sea sin ningún tipo de interrupción durante el proceso).

| Prueba manual | |
|---|---|
| *f* eje. | *t*x3 Nav. |
| 1 | 15min |
| 2 | 30min |
| 3 | 45min |
| 4 | 60min |
| 5 | 75min |
| 6 | 90min |

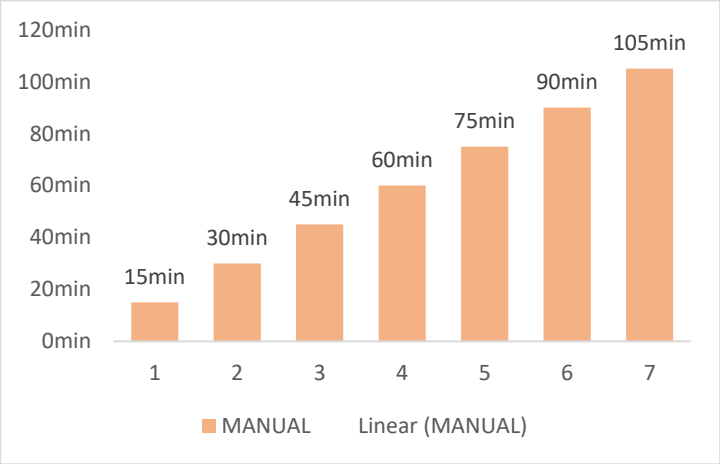


*Tabla 4. Prueba manual*

### 5.5.2. Proceso de ejecución automática

Una vez visto la realización de la ejecución manual de uno de los test más frecuentes, veamos cómo sería implementar el mismo ejemplo llevado a la automatización, utilizando el planteamiento y metodología descritos, pero antes de empezar, conviene destacar los siguientes conceptos y recomendaciones al programar con SeWe:

- Todo lo que se puede hacer con esta herramienta, estará basado en la localización de elementos en la página web y la realización de acciones sobre ellos. Para esto la herramienta pone a nuestra disposición clases e infinidad de métodos con los cuales se podrá simular cualquier flujo de trabajo que un usuario pueda realizar sin importar el grado de complejidad que esté realizando.

- El tester deberá tener conocimientos de Programación Orientada a Objetos para gestionar los tests scripts.

- Es recomendable diseñar los tests como frameworks para que cualquiera que utilice el código lo pueda utilizar de forma fácil e intuitiva.

- Para garantizar la mantenibilidad de los tests ya escritos se deberá diseñar las pruebas desde el principio usando el patrón de diseño “PageObject pattern” [10]. Las principales ventajas son garantizar que los futuros cambios que se vayan a realizar en los tests sean los mínimos y es lo ideal para diseñar “frameworks” de tests. La forma más sencilla de conceptualizar este patrón es la siguiente:

    - Cada página web corresponderá a una clase en java.

    - Todos los elementos de la página web equivalen a los miembros de la clase.

- Todas las acciones que se puedan realizar en la página web equivaldrán a métodos de la clase.

#### 5.5.2.1. Precondición configuración de parámetros

Para comenzar con este proceso de automatización, utilizaremos el proyecto base "testAutomationProject" alojado en nuestro repositorio Git y que se ilustró en la Figura 10 del apartado "5.2 Diseño del proyecto base". Este proyecto será la estructura común que los desarrolladores del testing utilizarán como plantilla para llevar a cabo las ejecuciones de los tests. Desde la herramienta Eclipse se irán ejecutando cada uno de los pasos que nos ha ido mostrando la herramienta "Squash TM" en la ejecución manual.

Nuestra primera tarea será lograr abrir la página de inicio de la aplicación, la cual nos mostrará un pequeño formulario que será rellenado con el nombre de usuario y la contraseña para poder acceder al aplicativo pulsando el botón de acceso "CONNECT", como se pudo mostrar en la Figura 19. Para ello será necesario la interacción entre varias clases en el proyecto y que a continuación nombramos. La clase "Helper" (una de las más usadas, a través de ella se refactoriza el código creando y reutilizando clases y funcionalidades respectivamente) crea a la clase "LoginPage" que es el equivalente a la página de inicio de la aplicación (estamos siguiendo las recomendaciones del patrón "PageObject" la cual plantea que por cada página haya una clase que sea su equivalente). Esta clase, se encarga de interactuar con los elementos del formulario de la página de inicio, pero antes debe saber qué navegador va a utilizar para entonces prepararlo acorde a lo que necesite su configuración, ya que según qué navegador se vaya a utilizar será necesario una determinada configuración. En la carpeta "resources" del proyecto se han almacenado los ejecutables encargados de lanzar el tipo de navegador en el que se quiere hacer las pruebas. Cada fabricante de navegador ha de implementar el estándar W3C WebDriver[17] para que SeWe pueda ejecutarse en él. Por tanto, el fabricante ha de proporcionar un medio desde donde los usuarios se puedan descargar dicho ejecutable, el cual será necesario para que el test pueda saber qué tipo de navegador lanzar en el momento de la ejecución. En nuestro caso vamos a necesitar de 3: chromedriver.exe para Chrome, geckodriver.exe para Firefox y msedgedriver.exe para Edge. Una vez hayamos preparado la configuración del navegador a utilizar, procederemos con la apertura de la página inicio. Ya con la página abierta y utilizando las diferentes funcionalidades de SeWe se logra rellenar el formulario y acceder a la aplicación a través del botón "CONNECT". En la siguiente Figura 23 se muestran los diferentes identificadores que serán utilizados por la herramienta SeWe y que al final se recogerán en la tabla Tab.ID:


*Figura 23. Identificadores de los elementos - página Inicio (Fuente propia)*

Una vez que hayamos accedido a la aplicación, el código del test crea una instancia de la clase "Referentiel" equivalente a la página de configuración para a continuación, realizar las correspondientes llamadas a sus métodos que nos permitirán llevar a cabo las diferentes acciones:

*ref.abrirPaginaConfiguración*: el navegador se redirige hacia la página configuración.

*ref.realizarBusquedaElementos*: utilizando la interfaz de SeWe y sus método localizaremos los elementos en la página web para realizar las diferentes acciones sobre ellos.

*ref.introducirDatos*: se encarga de escribir los datos en el campo a modificar

*ref.aceptarCambios*: confirma que los cambios sean realizados con éxito.

*ref.desconectar*: no desconectamos "logout" de la aplicación.

La Figura 24 muestra el diagrama de secuencia UML que representa esta primera parte del proceso, la precondición:

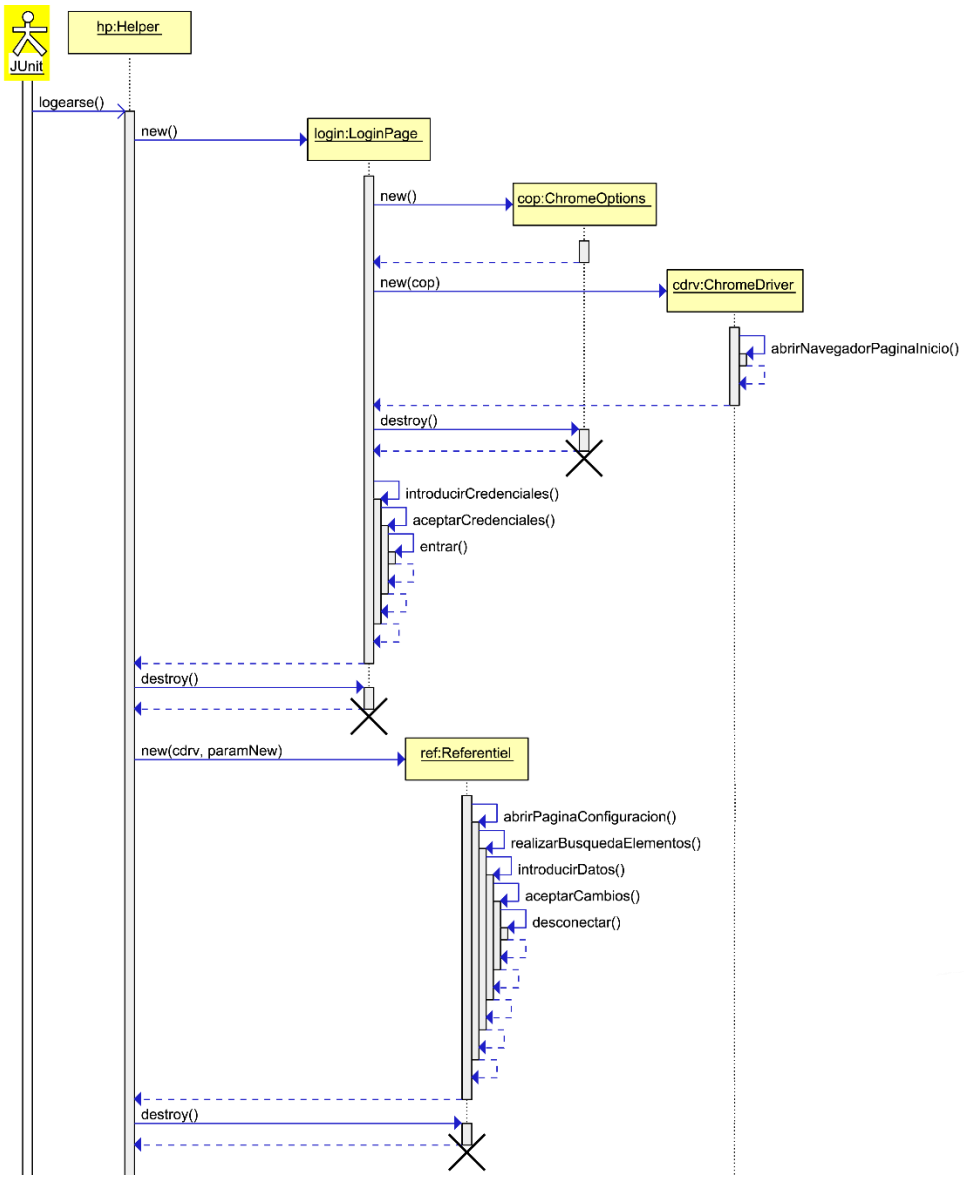


*Figura 24. Diagrama de secuencia UML - Precondición configuración de parámetro (Fuente propia)*

#### 5.5.2.2. Caso de prueba

Una vez que las precondiciones se establecen, procedemos a realizar nuestro caso de pruebas, para ello realizamos las mismas acciones llevadas a cabo relacionadas con el acceso a la aplicación. Una vez dentro accedemos a la página que queremos comprobar creando una instancia con la clase equivalente "ValidationPage". Verificado que existe el acceso a la página de Validación, podemos dar por válida la prueba, como nos especifica la historia en los requisitos.

Lo siguiente será probar si no existe el acceso con el parámetro apropiado. Para ello realizamos los mismos pasos que en el momento de la precondición para hacer las modificaciones en la página de configuración. Seguidamente nos desconectamos y volvemos a realizar las mismas llamadas hechas anteriormente, solo que esta vez no debería de existir el

acceso. Estas llamadas las podemos ver en la siguiente Figura 25 representada como diagramas de secuencia UML, a notar que las segundas llamadas a las clases “LoginPage” y “ValidationPage” de comprobación de no acceso, han sido omitidas.

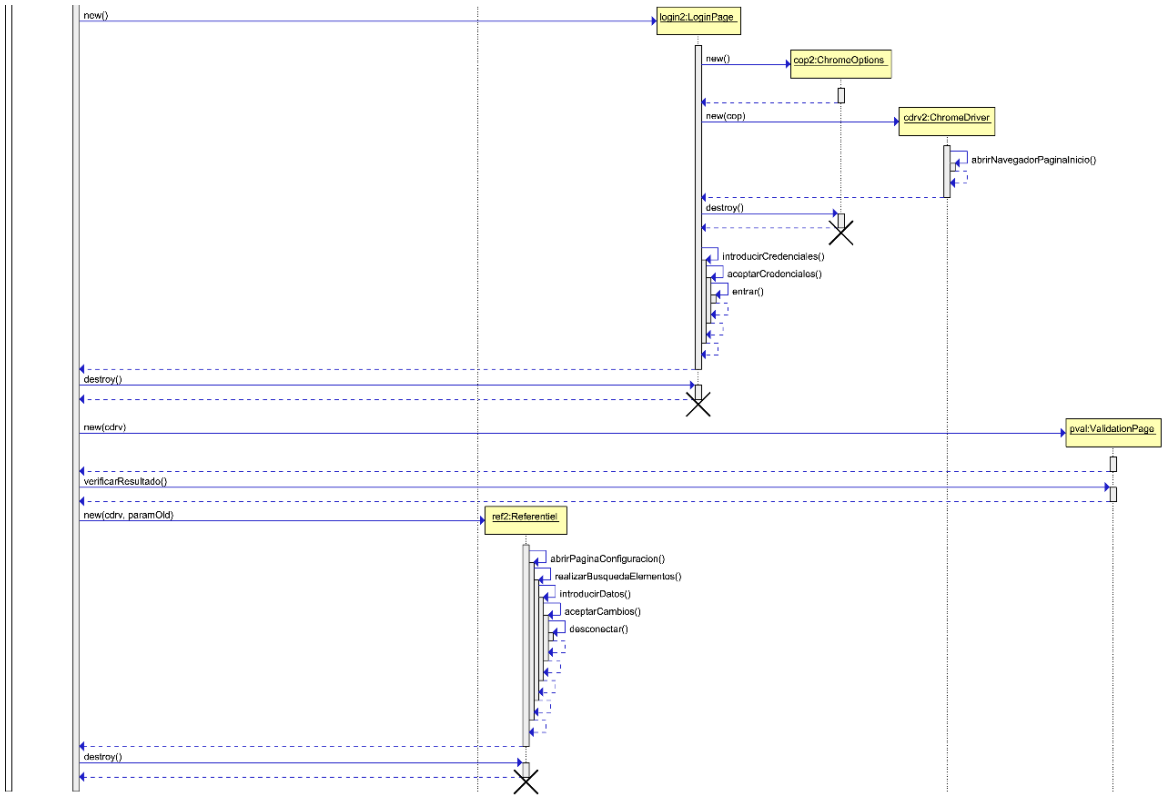


*Figura 25. Diagrama de secuencia UML - Caso de prueba: Acceso a la página validación (Fuente propia)*

#### 5.5.2.3. Postcondición configuración de parámetro

El proceso de restablecimiento de parámetros al igual que su diagrama es idéntico al proceso de precondición, solo que esta vez el valor del parámetro cambia. Finalizado el test se deberán rellenar las tablas en la guía del desarrollador con la información requerida. En este caso solo hemos necesitado de la Tab.ID. de la que mostramos una porción en la Tabla 5:

| Tabla de identificadores | | | |
|---|---|---|---|
| Elementos | Id | Clases | Referencia de localización |
| - | #user_login | - | Página de inicio |
| - | #user_pass | - | Pág. de inicio |
| - | #wp-submit | - | Pág. de inicio |
| - | #phx-btn-zone-ref | - | Pág. de configuración |
| button | title="Filtrer" | - | Pág. de configuración |
| span | - | ui-button-icon-primary<br>ui-icon<br>ui-icon-zoomout | Pág. De configuración |
| - | #paramdivers_filter | - | Pág. De configuración |
| button | - | filter-view-btn | Pág. De configuración |
| … | … | … | … |

*Tabla 5. Tab.ID. - Procesos de ejecución automática*

El rellenado de las tablas Tab.E.C y Tab.E.E no ha sido necesario en este ejemplo.

Por otro lado, en el código se ha creado una clase llamada “Consts” que recoge todos los identificadores usados en la identificación de los elementos y los reemplaza por variables constantes. Con esto se consigue que las modificaciones en los identificadores tengan el menor impacto posible en los tests scripsts desarrollados. Veamos un listado de cómo quedarían las variables constantes con su equivalencia de identificadores perteneciente a la anterior Tabla 5 siguiendo el siguiente formato ( ***VariableConstante*** = “Cadena de caracteres identificadora” ):

- ***usrID*** = "#user_login"
- ***usrPw*** = "#user_pass"
- ***lgnBtn*** = "#wp-submit"
- ***pageConf*** = "#phx-btn-zone-ref"
- ***elemBtnConfigPgFltr*** = "button[title=\"Filtrer\"]"
- ***elemSpanConfigPgZoom*** = "span.ui-button-icon-primary.ui-icon.ui-icon-zoomout"
- ***elemBtnfltrvw*** = "button.filter-view-btn"
- ***xamdvr*** = "#paramdivers_filter"

Si fuera necesario modificar algún identificador en la Tab.ID, la actualización de ese identificador en el o los tests se realizaría en la clase que contiene las variables constantes y no en todos los tests scripts donde se estuviera usando ese identificador, en caso de no haber empleado esta clase “Consts”.

La siguiente Tabla 6 y diagrama muestran la frecuencia con que se ha ejecutado la prueba, o sea el número de veces que se ha ejecutado (*f* eje.), frente al tiempo que se tarda en realizar esta prueba (*t*) incluyendo el de complementar las tablas que serán luego usadas por los desarrolladores durante el desarrollo. Los encabezados (*t* eje.) y (*t* dev.) equivalen al tiempo de ejecución y el tiempo de desarrollo respectivamente.

| Prueba automática | | | |
|---|---|---|---|
| *f* eje. | *t* eje. | *t* dev. | *t* |
| 1 | 90s | 2400s | 41,5min |
| 2 | 90s | 0s | 43,0min |
| 3 | 90s | 0s | 44,5min |
| 4 | 90s | 0s | 46,0min |
| 5 | 90s | 0s | 47,5min |
| 6 | 90s | 0s | 49,0min |

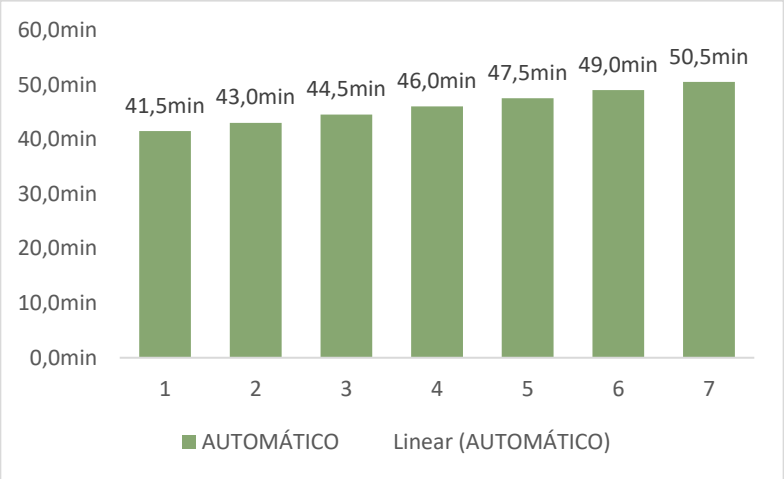


*Tabla 6. Prueba automática*

## 5.6. Informe de resultados obtenidos

La siguiente Figura 26 muestra la gráfica de la unión de los resultados de ejecución de las 2 graficas anteriores. En ella se puede apreciar cómo a partir de la 3 ejecución del caso de pruebas se empieza a notar una pequeña mejora en la ejecución automática del procedimiento de pruebas. Por otro lado, se puede apreciar que, a partir de la 7ª ejecución se logran reducir los tiempos a la mitad.

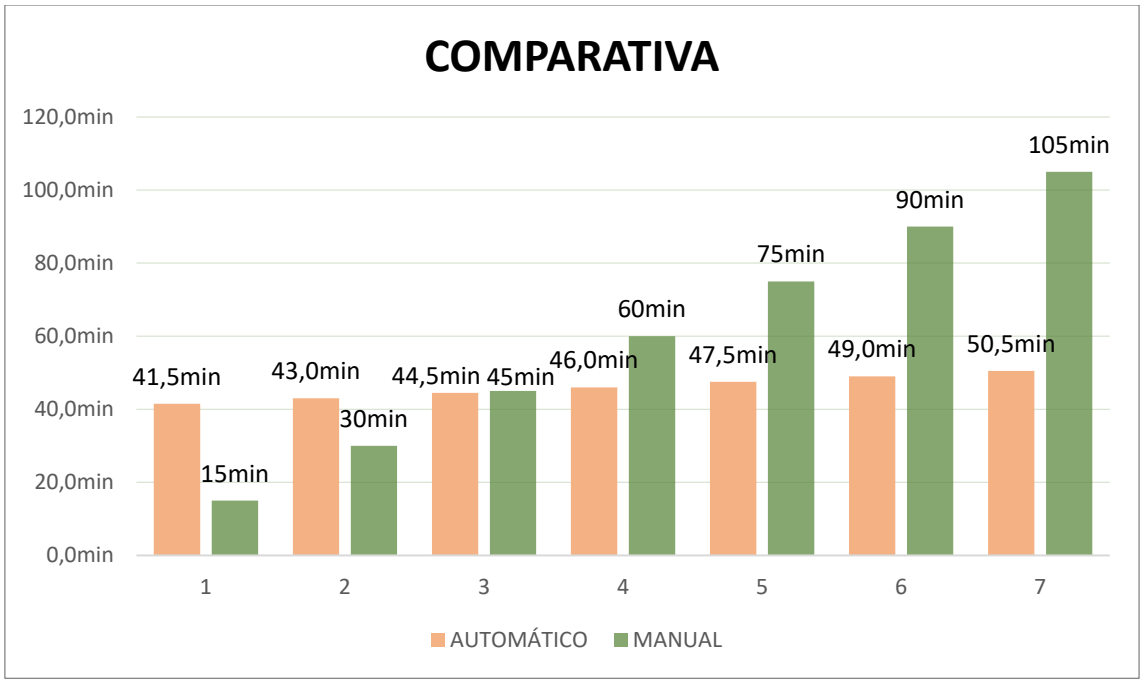


*Figura 26. Comparativa prueba manual vs automático*

Las siguientes gráficas de "Ratio de rendimiento entre la ejecución manual y la automática" nos dan una mejor visión respecto a la mejora en porcentajes. Los resultados de rendimiento se han obtenido utilizando la fórmula siguiente:

$$n = 100\frac{tiempo\ ejecución_y - tiempo\ ejecución_x}{tiempo\ ejecución_x}$$

Si queremos saber en qué porcentaje es una forma de ejecución A más rápida que la otra B, asignaremos a $tiempo\ ejecución_x$ el valor del tiempo que ha tardado A y a $tiempo\ ejecución_y$ el valor del tiempo que ha tardado B.

La siguiente Figura 27 muestra la gráfica del ahorro de tiempo que supone el haber automatizado el procedimiento de pruebas con cierto grado de complejidad y habiendo superado en número de ejecuciones en 3.

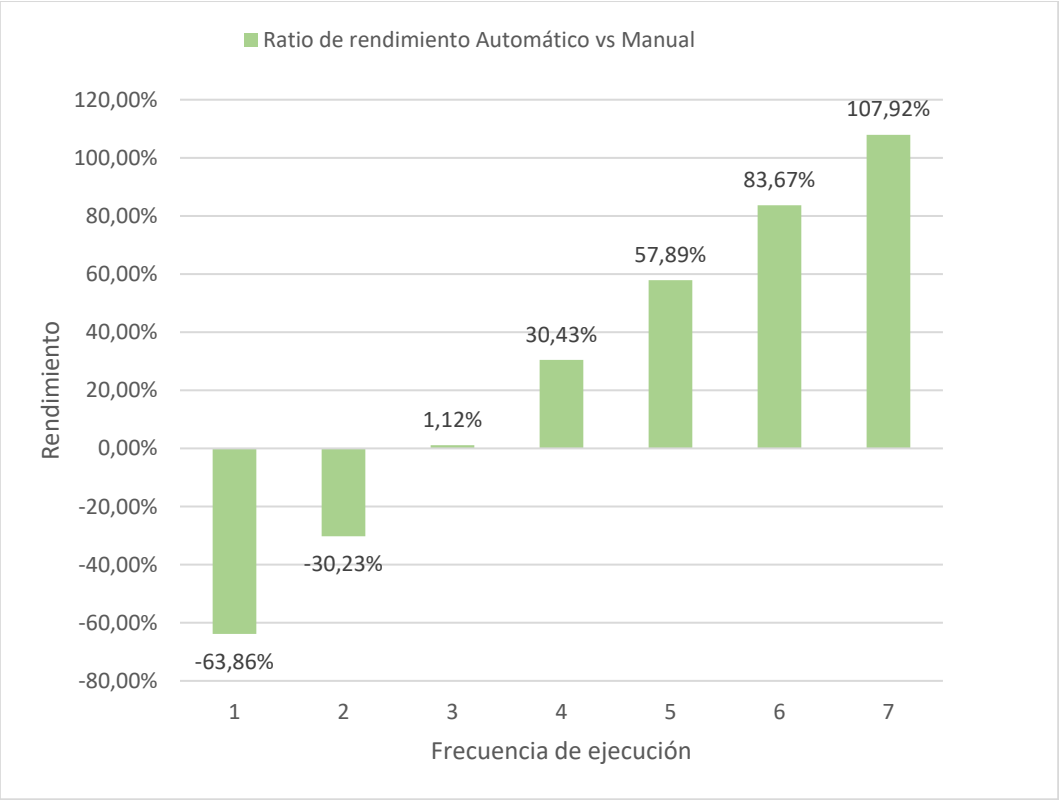


*Figura 27. Ratio de rendimiento automático respecto al manual (Fuente propia)*

En la gráfica de la siguiente Figura 28 se puede apreciar el derroche de tiempo que supone continuar haciendo las pruebas manuales a partir de tener que ejecutar un procedimiento de pruebas con cierto grado de complejidad y habiendo superado en número de ejecuciones en 3.

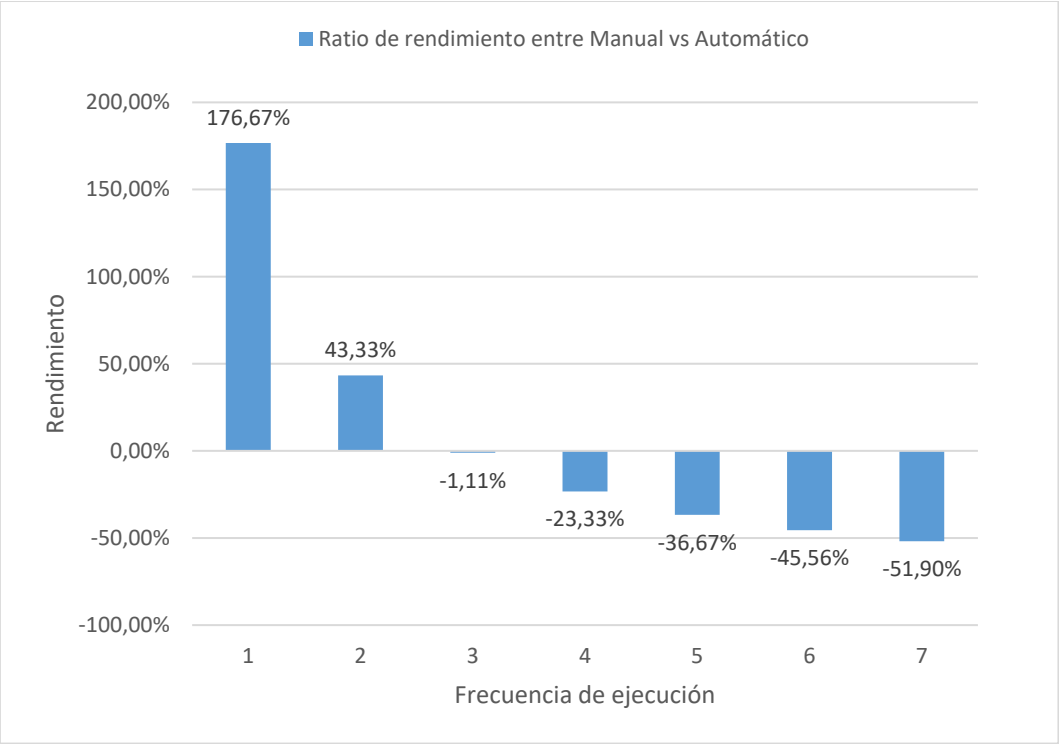


*Figura 28. Ratio de rendimiento manual respecto al automático*

Es Importante destacar que los beneficios de esta automatización pueden verse aumentados ya que esta misma prueba de acceso a la página de validación verificando la existencia de ciertos parámetros en la página de configuración, se suele repetir en prácticamente todas las ventanas de la aplicación. La mayoría de las ventanas tiene el mismo comportamiento que la página de validación en cuanto a configuración de parámetros se refiere, por tanto, el haber desarrollado una prueba para la página de validación servirá de igual forma para la comprobación del resto de ventanas existentes y futuras a desarrollar. Solamente habrá que desarrollar la clase equivalente a esa ventana para que se realice su respectiva llamada de acceso y con esto se reduce drásticamente el tiempo empleado en de desarrollo de las pruebas.

Por otro lado, los resultados evidencian también que la automatización de testing es adecuada solo en las pruebas que vayan a ser repetidas varias veces y que no presenten un grado de complejidad de desarrollo demasiado alto, ya que de ser así existiría poca o ninguna rentabilidad en el tiempo de desarrollo invertido puesto que el número de ejecuciones tendría que ser demasiado alto para que comenzara a haber un rendimiento favorable. P.ej. si se conoce que una prueba va a ser ejecutada una o dos veces durante todo el tiempo que dure el desarrollo y además se estima que la complejidad en ser automatizada será bastante alta, entonces esa prueba no sería una buena candidata a elegir, quedando descartada del proceso de automatización. Con esto se deduce que la elección correcta de los casos a automatizar es clave para no emplear tiempo en desarrollo que no compensará.

A causa de lo anterior mencionado es fundamental desarrollar el código, los scripts tests, de forma eficiente, como un esqueleto en el cual solamente habrá que introducir llamadas a clases equivalentes a páginas, y cada clase tendrá sus métodos equivalentes a las funcionalidades de la página en la aplicación objeto de pruebas. Dicho de otra forma, el desarrollo de las pruebas se hará en formato plantilla o “framework” para facilitará el trabajo a los demás testers en el momento de creación de alguna prueba.

## 5.7. Plan de puesta en marcha

Este punto da comienzo a la "3ª Fase – Plan de implantación de la automatización del testing en la empresa" enunciada en la metodología. Se realizará un plan de puesta en marcha para la implantación de la automatización del testing en una pequeña empresa. Para ello me basaré en mi propia experiencia tomando como modelo una empresa en la cual eres empeñado mi labor de tester. Partimos suponiendo que hemos obtenido "carta blanca" para esta propuesta por parte de los directivos, pues sin su consentimiento sería muy difícil esta gestión. La situación actual del departamento de calidad de la empresa caso de estudio es la de un equipo de calidad (Q.A.) formado por 3 integrantes. Este personal es el responsable de garantizar que un producto software desarrollado por un grupo de alrededor de 30 desarrolladores sea entregado a un ente desconocido, con el menor número de defectos posibles.

Durante el período de tiempo planificado, alrededor de 35 horas, se intentará hacer que este personal sea capaz de realizar sus propias pruebas de automatización de forma autónoma y que produzcan los beneficios esperados. Por otra parte, se medirá el grado de eficiencia que se ha alcanzado al introducir la automatización en aquellas pruebas que así se presten. Durante los primeros días y hasta que el personal vaya adquiriendo conocimiento y destreza, un ingeniero informático encargado de hacer la implantación de la automatización del testing suministrará ayuda ante cualquier duda.

### 5.7.1. Creación de la figura del responsable del testing (test manager)

El primer punto a tratar será el de designar un responsable del testing (si no existe) con autoridad conferida para llevar a cabo esta puesta en marcha de mejora del testing. Esta persona será la responsable gestionará todo lo concerniente a:

- Las herramientas a utilizar, que en este caso las concernientes al testing serán las seleccionadas en la propuesta, Selenium WebDriver.
- La comunicación con la directiva.
- La preparación de los entornos de trabajos (repositorio local donde alojar los proyectos de testing, gestión de la configuración y versionado).
- La gestión del personal profesional para la actividad de la automatización del testing.
- Registro de incidencias en las pruebas.
- La prioridad de las funcionalidades susceptibles automatizar.

- La toma de decisiones sobre qué puede (o no) ser automatizado.
- La estimación de los tiempos de desarrollo de los tests scripts.
- La comunicación con el equipo de desarrollo y los testers.
- El control y la monitorización del proceso de mejora implantado.
- La evaluación de los resultados.

También resulta interesante y útil la creación de una segunda figura que apoye al "test manager" en su labor diaria. Servirá de ayuda en momentos de análisis y evaluación del rendimiento de la automatización. En la interacción con el personal profesional, tendrá también capacidad de en la toma de decisiones en la gestión del testing si no se encuentra disponible la figura del "test manager". A esta persona la acuñaremos con el nombre de "Tech-leader".

### 5.7.2. Plan de formación

Para que pueda establecerse la automatización en el proceso de ejecución del testing con la herramienta SeWe, es necesario la realización de un plan de formación para todo el equipo Q.A. Se desarrollará un método de aprendizaje progresivo para que no afecte al rendimiento actual de la empresa ni a la labor diaria de los trabajadores en la empresa. Se contratará a un ingeniero con conocimiento en la materia para servir de guía en todo el proceso de implantación.

#### 5.7.2.1. Formación del "Test manager"

La figura del "test manager" será la primera en recibir la formación, es ella quien distribuirá posteriormente todo el conocimiento necesario al resto de integrantes del equipo. Durante una semana y en períodos de 2 horas diarias recibirá lecciones sobre cómo llevar a cabo la automatización de forma eficiente.

En el primer día:

- Se le explicará la importancia de la automatización y sus beneficios. A través de diapositivas se le mostrará cómo la sociedad actual va rumbo hacia la automatización en la inmensa mayoría de campos. Desde las ciudades inteligentes, las industrias cada vez más robotizadas, hasta la comunicación con las máquinas a través de chatbots y la tendencia que tienen los trabajos "manuales repetitivos" a desaparecer en un futuro no muy lejano a causa de la transformación digital en las empresas. Destacar que en los primeros momentos quizás no se vean resultados favorables en el rendimiento pero que a largo plazo se obtendrán los beneficios como lo afirma la ley de Amara "nuestra tendencia es sobrestimar los efectos de una tecnología en el

corto plazo y subestimar el efecto en el largo plazo" (asignatura del master "Dirección y estratégica de las tecnologías de la información")

- se le dará acceso a materiales con la base de conocimiento sobre SeWe
- se realizará un pequeño experimento con SeWe en una página web real.

En el segundo día:

- Se creará un de proyecto plantilla desde el cual se irán agregando de manera progresiva los casos de pruebas a automatizar.
- Se llevará a cabo la primera prueba de automatización con el producto de la empresa.
- Se detallará las diferentes formas que tiene la herramienta SeWe de identificar los elementos en la página web y ejecutar acciones sobre ellos simulando a un usuario real.

En el tercer día:

- Se demostrará con estudios estadísticos los beneficios y desventajas de automatizar procedimientos de pruebas.
- Se explicarán los riesgos concernientes al desarrollo del testing automático, pues el hecho de desarrollar un test script puede implicar la introducción de fallos en la misma prueba.
- Recomendaciones sobre cuándo es rentable la automatización y cómo elegir los casos automatizar (p.ej. casos muy repetitivos, que el grado de complejidad en el flujo de trabajos del usuario final no sea muy alto, los niveles de riesgos que existan en la realización de un flujo de trabajo). Se ha de tener en cuenta que automatizar pruebas muy complejas puede generar errores ya que aumenta la probabilidad de generarlos.
- Se explicarán los riesgos que existen de que los tests scripts dejen de funcionar si no se implica al departamento de desarrollo en la automatización.

En el cuarto día:

- Se llevarán a cabo pruebas y comprobaciones para verificar que el alumno ha adquirido los conocimientos básicos necesario para desarrollar sus propios tests scripts.

- Habilitación del entorno para repositorio común. Se alojará el proyecto en un repositorio de la empresa y se hará la configuración necesaria para que pueda ser accedido su contenido desde otro ordenador en la empresa a través de una herramienta de control de versiones.

En el quinto día:

- Se realizará un pequeño examen al "test manager" (teórico y práctico) sobre los conocimientos adquiridos.

#### 5.7.2.2. Formación del tester

Una vez que el "test manager" haya pasado el entrenamiento, el mismo será el encargado de elegir una vez por semana a un integrante del equipo Q.A. para transferir los conocimientos adquiridos durante el curso. El test manger irá dando instrucciones para preparar el entorno de trabajo del tester e ir automatizando casos de test. De forma similar a como "test manager" fue instruido, el tester recibirá lecciones por parte de su responsable durante 5 días con 2 horas de duración. Una vez finalizada la semana, el tester con conocimiento de automatización quedará libre para continuar con su labor y a la espera de órdenes del "test manager" para automatizar sus primeros casos de pruebas.

Por cada caso de prueba que el tester automatice, deberá subirlo al repositorio común para que sirva de base de conocimiento a los demás testers y a su vez se pueda reutilizar en posteriores pruebas de regresiones. Antes de subir cualquier nueva versión al repositorio de testing, el tester deberá comprobar que el proyecto, al menos compila compila.

Es test manager tendrá la labor de monitorizar, el trabajo realizado de su personal a cargo y de aclarar cualquier tipo de duda que vaya surgiendo hasta que todo el equipo Q.A. adquiera la destreza necesaria.

### 5.7.3. Implicación de grupos externos

Para garantizar la integridad de los test scripts desarrollados, o sea para que una vez creados sigan funcionando en futuras ejecuciones. Se ha de garantizar que el desarrollo de nuevas funcionalidades por parte de los desarrolladores no implique cambios en las referencias de los elementos y o estructuras que ya han sido identificados por los tests scripts en las pruebas automáticas con SeWe. Por este motivo el "test manager" junto con el equipo de Q.A. deberá convocar una reunión informativa dirigida al departamento de desarrollo.

Los puntos a tratar serán los ya vistos en el apartado ' "5.4 Creación de la "Guía del desarrollador para la automatización del proyecto" ', al equipo de desarrollo se le informará de

las nuevas normas a seguir a partir del momento que el departamento de calidad informe del establecimiento de la automatización. Desde entonces los desarrolladores deberán cumplir con un conjunto de reglas y de actualizar 3 tablas que mantendrán la integridad referencial entre las pruebas automáticas y los elementos que conforman la aplicación web cuando se realicen modificaciones en el código después de que haya sido testado y por tanto existan pruebas automáticas.

### 5.7.4. Otras responsabilidades del test manager

El responsable del testing tendrá la responsabilidad de estimar los tiempos de desarrollo de los tests scripts comparándolo con el tiempo de realización manual del test para comprobar que no hay desvíos en el rendimiento de la actividad del testing. Comparará la ganancia con los tiempos de test manual frente al automático para verificar que la elección del caso de test a automatizar es correcta.

# 6. Planificación

La planificación será un aspecto clave para la implantación y puesta en marcha de la automatización en la empresa. Nos permitirá hacer una estimación de cuánto tiempo llevará implantarlo todo y que funcione dentro de un periodo acotado. Este proceso nos permitirá explicar a los directivos de forma más clara, concisa y formal en qué tiempo se llevará a cabo nuestro plan de conversión, pudiendo identificar y prever en todo momento los retrasos en las fechas de finalización de cada actividad. Pretendemos también causarles una buena impresión para ganar un poco más de confianza y apoyo.

El planning será también un resguardo que nos ayudará a comprender cómo ha ido evolucionando todo el proceso para su posible reutilización en futuros proyectos de implantación de la automatización del testing. La Figura 29 muestra el planning general la Figura 30 y Figura 31 muestran el mismo planning, pero más de cerca.

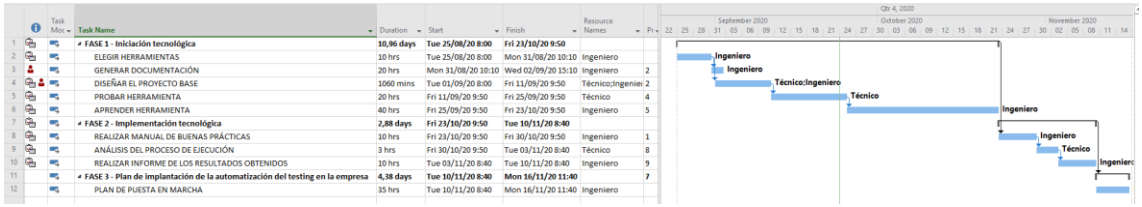


*Figura 29. Planificación - vista completa (Fuente propia)*

| | Task Mod | Task Name | Duration | Start | Finish | Resource Names | Pr |
|---|---|---|---|---|---|---|---|
| 1 | | **FASE 1 - Iniciación tecnológica** | **10,96 days** | **Tue 25/08/20 8:00** | **Fri 23/10/20 9:50** | | |
| 2 | | ELEGIR HERRAMIENTAS | 10 hrs | Tue 25/08/20 8:00 | Mon 31/08/20 10:10 | Ingeniero | |
| 3 | | GENERAR DOCUMENTACIÓN | 20 hrs | Mon 31/08/20 10:10 | Wed 02/09/20 15:10 | Ingeniero | 2 |
| 4 | | DISEÑAR EL PROYECTO BASE | 1060 mins | Tue 01/09/20 8:00 | Fri 11/09/20 9:50 | Técnico;Ingenier | 2 |
| 5 | | PROBAR HERRAMIENTA | 20 hrs | Fri 11/09/20 9:50 | Fri 25/09/20 9:50 | Técnico | 4 |
| 6 | | APRENDER HERRAMIENTA | 40 hrs | Fri 25/09/20 9:50 | Fri 23/10/20 9:50 | Ingeniero | 5 |
| 7 | | **FASE 2 - Implementación tecnológica** | **2,88 days** | **Fri 23/10/20 9:50** | **Tue 10/11/20 8:40** | | |
| 8 | | REALIZAR MANUAL DE BUENAS PRÁCTICAS | 10 hrs | Fri 23/10/20 9:50 | Fri 30/10/20 9:50 | Ingeniero | 1 |
| 9 | | ANÁLISIS DEL PROCESO DE EJECUCIÓN | 3 hrs | Fri 30/10/20 9:50 | Tue 03/11/20 8:40 | Técnico | 8 |
| 10 | | REALIZAR INFORME DE LOS RESULTADOS OBTENIDOS | 10 hrs | Tue 03/11/20 8:40 | Tue 10/11/20 8:40 | Ingeniero | 9 |
| 11 | | **FASE 3 - Plan de implantación de la automatización del testing en la empresa** | **4,38 days** | **Tue 10/11/20 8:40** | **Mon 16/11/20 11:40** | | **7** |
| 12 | | PLAN DE PUESTA EN MARCHA | 35 hrs | Tue 10/11/20 8:40 | Mon 16/11/20 11:40 | Ingeniero | |

*Figura 30. Planificación - Listado de tareas (Fuente propia)*

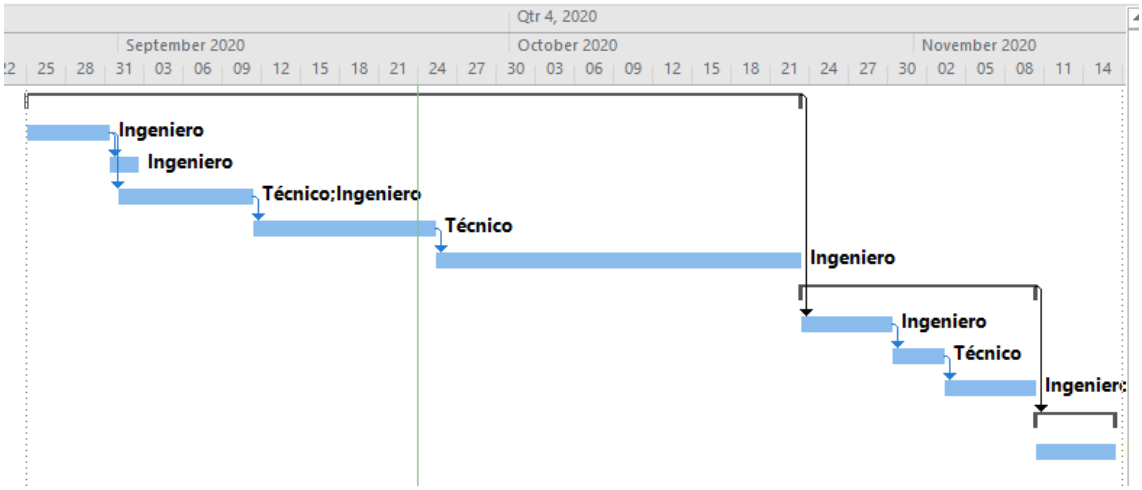


*Figura 31. Planificación - diagrama de Gantt (Fuente propia)*

La implantación de un proyecto de estar magnitudes en una empresa tendría un mínimo aproximado de duración de 3 meses. La siguiente figura muestra esta duración empleando un calendario estándar, esto es empleando 8hrs al día y trabajando de lunes a viernes.

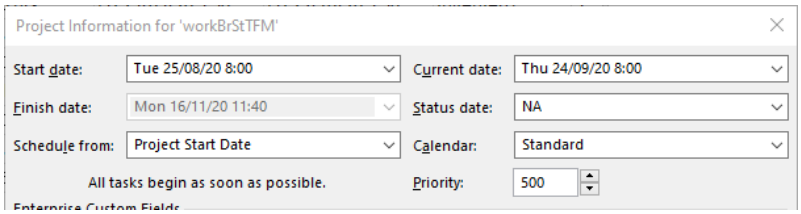


*Figura 32. Duración mínima del proyecto bajo un calendario estándar (Fuente propia)*

# 7. Propuesta de adopción de marcos formales

Una vez que nuestra empresa objeto ha llevado a cabo el anterior "Plan de puesta en marcha" estará en condiciones (y de forma natural) para afrontar nuevos retos ya que en el entorno empresarial sus factores ambientales internos favorecen a la implantación de nuevos los marcos de mejora del testing. De uno de estos Marcos de mejora e implantación del testing en la empresa se hará una simulación de los puntos a implementar y se comprobará que una buena parte de ellos ya estaría siendo implementado por la empresa a través de la propuesta de solución expuesta en este trabajo. El marco objetivo será el TMMi.

Se debería de comenzar por el nivel más bajo el 2, como en el marco se aconseja ya que el nivel 1 es aquel que cualquier empresa que no esté implementado ningún marco de mejora está implementando por defecto. El nivel 2 se denomina "gestionado", porque es una forma de trabajar muy acoplada a un proyecto en concreto dentro de una empresa (en un nivel 3 todos los proyectos de la empresa siguen una misma forma de trabajar). Por tanto, tomaremos ciertas prácticas pertenecientes a cada una de sus áreas de procesos y la ajustaremos a nuestra empresa objetivo para así, de una forma estándar y formal, lograr un plan para la implantación del testing siguiendo las recomendaciones de los expertos, pero ajustado a las limitaciones de la empresa. No existe un orden para la implementación de las áreas de procesos pues el objetivo final es el de implementarlas todas según convenga a la empresa.

El marco aconseja que finalizado el trabajo con todas las áreas de procesos se necesitará volver a repasar todos los puntos para poco a poco ir corrigiendo cada apartado y seguir agregando más actividades (denominadas prácticas) para finalmente lograr alcanzar todas las metas que marca cada área de proceso. Solo así se podrá pasar a un nivel superior, en este caso el siguiente nivel 3.

## 7.1. Área de proceso "Diseño y ejecución del testing"

El propósito del diseño y ejecución del testing es el de mejorar la capacidad del proceso de testing durante el diseño y la ejecución. Para esto se establecen especificaciones de diseño, se emplean técnicas de diseño de prueba, se realiza un proceso estructurado de la ejecución del test y se gestiona el cierre o la continuidad de las incidencias.

### 7.1.1. Meta genérica: Institucionalizar el proceso gestionado

Para conseguir cumplir con esta meta se desarrollarán un conjunto de actividades (prácticas genéricas) que a continuación se enunciarán.

- Asignar autoridad y responsabilidad para realizar el diseño de las pruebas y su ejecución.
- Realización de entrenamientos dentro de la empresa.
- Proporcionar los recursos para la ejecución de las pruebas, como pueden ser las herramientas de ejecución.
- Monitorear y controlar el proceso de ejecución automática del testing basándose en la planificación hecha con anterioridad y tomar medidas de corrección en caso de desvío en los tiempos.

Como se puede comprobar este conjunto de prácticas genéricas ya están recogidas en Plan de puesta en marcha propuesto.

### 7.1.2. Meta específica: realización del análisis y el diseño de las pruebas usando técnicas de diseño.

Para conseguir esta meta se llevarán a cabo las siguientes prácticas específicas:

- Se identificarán y se darán prioridad a las condiciones de prueba. Tal y como recomienda el "Grupo de Actividad – Realizar análisis" del Proceso del testing propuesto por el ISTQB estudiado en Estado del arte., nuestra empresa realiza este análisis de las historias que en cada sprint se suministran. Como la empresa se interesa por las pruebas de aceptación, la técnica de diseño que se emplea es la de Caja negra, utilizando métodos como los de particiones equivalentes, análisis del valor límite, tablas de decisiones y diagramas de transición de Estado). Empleando estas técnicas se extraen las condiciones de pruebas de las historias suministrada. Las condiciones de pruebas son documentadas de tal forma que se sepan los criterios por los que se cumple o no una determinada condición. Los testers son los responsables de extraer las condiciones de pruebas de las historias.
- Se identificarán y se darán prioridad a los casos de prueba. Como recomienda el "Grupo de Actividad – Diseñar las pruebas" del Proceso del testing propuesto por el ISTQB, los testers de nuestra empresa objeto de estudio desarrollan los casos de pruebas a partir de las condiciones de pruebas y por su parte el responsable del testing asigna prioridades a los casos de pruebas basándose en los riesgos identificados del producto software a desarrollar. Los casos de pruebas se documentan en la herramienta de gestión de casos de pruebas "Squash TM", de esta

forma se podre asignar identificadores a cada uno de ellos para un posterior seguimiento.

- Para mantener una trazabilidad horizontal con los requisitos “las historias” y las condiciones de pruebas se creará una matriz de trazabilidad. El responsable del testing será el encargado de llevar el mantenimiento de esta matriz. El objetivo de esta matriz es la de facilitar la monitorización de la cobertura de requisitos durante la ejecución de las pruebas.

De estas prácticas específicas solamente la última no se recoge en el Plan de puesta en marcha.

### 7.1.3. Meta específica: Implementación de las pruebas

Siguiendo el Proceso del testing propuesto por el ISTQB durante la implementación de las pruebas se generan los procedimientos de pruebas, en nuestro caso las pruebas son creadas en la herramienta de gestión “Squash TM”. Esta meta no presenta ningún cambio en el actual proceso de la empresa pues actualmente se siguen las prácticas específicas que recomienda el marco.

El responsable del testing planificará la secuencia con que los procedimientos de pruebas serán ejecutados. Se encargará de investigar las dependencias entre los procedimientos de prueba, elegirá el tester que realizará la transformación de la prueba manual a automática y le asignará los casos de prueba a automatizar.

### 7.1.4. Meta específica: Ejecución de las pruebas

En esta meta es donde se realizará la ejecución de las pruebas, de acuerdo con los procedimientos de prueba ya escritos. Los problemas que se encuentren serán informados y registrados en los medios correspondientes. Las siguientes prácticas específicas serán realizadas para alcanzar esta meta:

- Ejecutar los casos de prueba. De acuerdo con la planificación se ejecutarán de forma manual y/o automática utilizando procedimientos de pruebas ya documentados. Posteriormente se registrarán los resultados. Se comparan los resultados actuales con los resultados esperado. Se realizan las pruebas de regresiones que procedan.

- Reportar las incidencias creando un informe de incidencias. Se deberá informar de las incidencias encontradas a los interesados correspondientes, los desarrolladores, y se llevará un registro de cada una de ellas. Se establecen las causas del incidente

por parte de los testers. Los incidentes serán almacenados en un repositorio central para que todos puedan tener acceso a ellos.

- Escribir un log de test. El objetivo de esta práctica es proporcionar un informe Cronológico con detalles relevantes relativo a las ejecuciones se las pruebas. Actualmente la empresa ya recoge esta práctica con la herramienta Squash TM pues al momento de la ejecución de los tests se quedan un registro de forma automática de todo lo sucedido con esa prueba.

De estas prácticas, la primera de ellas, el Plan de puesta en marcha propone la mejora con las pruebas de regresiones automáticas, el resto ya se está implementando.

### 7.1.5. Meta específica: Finalización de los incidentes encontrados en las pruebas.

En esta meta se describe la forma de gestionar y se dar solución a los problemas encontrados. Las siguientes prácticas específicas son necesarias implementar:

- Realizar acciones apropiadas para corregir las incidencias. Para ello se realizan las siguientes actividades registrar información de la incidencia preparada en la base de datos de incidencias y actualizar el informe de incidencias. Realizar un retesteo y/o prueba de regresión para confirmar que el problema ha sido solucionado. Cerrar de forma formal la incidencia informando que la prueba de retesteo y/o prueba de regresión se ejecutó de forma satisfactoria.

- Llevar un seguimiento de las incidencias encontradas. para ello se realizan las siguientes actividades proporcionar un informe de estado de las incidencias a los interesados. Tomar acciones pertinentes caso de que una incidencia qué necesita ser reparada mantenga el mismo estado durante mucho tiempo.

Ambas prácticas ya se realizaban en la empresa aún antes de la Propuesta de la solución.

## 7.2. Área de proceso "Políticas y estrategias del testing"

El propósito de esta área de proceso es desarrollar y establecer una política y estrategia institucional de testing. También se definen los indicadores de rendimiento de pruebas.

### 7.2.1. Meta genérica: Institucionalizar el proceso gestionado

Lo que se pretende con esta meta es que haya un compromiso por parte de la directiva en mantener estos procesos y que los mismo formen parte y se tengan en cuenta en el conjunto de procesos realizado en el desarrollo del proyecto. De tal forma que en tiempos de estrés no

se abandone la forma de trabajar. De igual forma que en las metas específicas, aquí se desarrollarán un conjunto de actividades (prácticas genéricas) que a continuación se enunciarán.

- Asignar responsabilidad. En este caso se designa a un grupo con autoridad y conocimiento que sea responsable definir las políticas de pruebas, las estrategias y los indicadores de rendimiento de pruebas. En nuestra propuesta hemos asignado esta responsabilidad a la figura del "test manager".

- Plan de formación. Al responsable de definir estos procesos de política y estrategia se le entrenará para que adquiera conocimientos básicos del testing. Se les enseñar a prácticas de medición en el desarrollo de las pruebas. En el entrenamiento de nuestra propuesta se le indica al responsable del testing qué casos de test son susceptibles de automatizar. Se le enseña en base a qué deberá tomar esta decisión (p.ej. los procedimientos de pruebas que más repiten, de los que más se repiten, los que más tiempo requieran de desarrollo del test script, los riegos de cada caso de pruebas).

- Monitorizar y controlar el proceso. El responsable deberá monitorizar y controlar que las políticas y estrategias se cumplan según lo planificado, de lo contrario deberá tomar acciones rectificadoras.

Una vez más las prácticas mencionadas son llevadas a cabo en el Plan de puesta en marcha.

### 7.2.2. Meta específica: Estableciendo las políticas de test

La política del testing define los objetivos generales de la organización. En nuestro caso hemos definido un objetivo principal y subobjetivos (en al apartado **Objetivos**) que son nuestros indicadores de rendimiento claves del test (KTPIs) y que están alineados con la política de calidad de la empresa.

### 7.2.3. Meta específica: Estableciendo las estrategias

La estrategia del testing se basa en la política del testing. Analiza y gestiona los riesgos generales del producto buscando formas de mitigar esos riesgos mediante procesos que brinden solución. En general se incluye la descripción de los niveles de pruebas (p.ej. unitarias, de integración, de sistema y de aceptación) a aplicar y por cada nivel como mínimo se definen objetivos, responsabilidades, tareas principales y criterios de entrada y de salida. El nivel de prueba de la empresa para la cual trabajamos es el de aceptación y para este nivel se deben cumplir los siguientes objetivos, responsabilidades y tareas principales:

- Verificar que el sistema satisface los criterios de aceptación definidos.

- Validar si el sistema es “apto para el uso” del inglés “fit for use”
- Alcanzar cierto nivel de cobertura en los requisitos del usuario.

Estas 3 responsabilidades corren ya en la empresa objeto de estudio a cargo del tester y su responsable.

Otra estrategia a usar es la de buscar modos de automatización para cada nivel de pruebas. Es nuestra “Propuesta de la solución” la estrategia encargada de gestionar este punto.

Otra estrategia será emplear “técnicas de diseño del test” según el nivel de pruebas. En nuestro caso empleamos la técnica de prueba de caja negra, puesto que para realizar el test nos basamos en las historias que se suministran durante el desarrollo.

### 7.2.4. Meta específica: Estableciendo indicadores del testing

Para implementar esta meta se establece y se despliega un conjunto de indicadores de rendimiento de procesos de prueba orientados a conseguir un propósito para medir la calidad del proceso de prueba. El siguiente conjunto de prácticas específicas se han de me llevar a cabo:

- Para establecer los indicadores de rendimiento se definirán y seguidamente se pondrán en marcha mediciones que permitirán analizar y realizar informes. En nuestro caso se analiza los tiempos de desarrollo y ejecución de los test automáticos, calculando el esfuerzo y lo que cuesta realizarlos, como se mostró en el apartado “5.5.2 Proceso de ejecución automática”. Por otra parte, se contabilizará el número de defectos encontrados y el porcentaje de detección. Otro indicador importante a definir será conocer la cobertura del test con respecto a los criterios de aceptación. La herramienta “Squash TM” que utiliza la empresa se encarga de estas 2 últimas tareas de forma automática.

- Para la puesta en marchas de los indicadores se obtendrán datos específicos de indicadores para un posterior análisis, p.ej. tablas y gráficas obtenidas. De forma periódica se enviarán informes de los resultados a todos aquellos interesados y se asistirá en la comprensión de los informes a los interesados. Esta última práctica también se implementa en nuestro Plan de puesta en marcha, aunque el asistir a los interesados en la compresión sería algo nuevo a incluir ya que los interesados tienen acceso a la información, pero esto no asegura que la comprendan.

## 7.3. Área de proceso "Planificación del testing"

El propósito de la planificación del testing es el de definir un método basado en los riesgos identificados y en las estrategias definidas y de también establecer y mantener una buena planificación para gestionar, llevar a cabo y controlar las actividades del testing. Definirá qué pruebas se necesitarán hacer, cuándo, cómo y por quién. Por cada nivel de pruebas que se identifique se necesitará un plan de test.

### 7.3.1. Meta genérica: Institucionalizar un proceso gestionado

Para conseguir cumplir con esta meta se debe ejecutar las siguientes prácticas genéricas:

- Establecer y mantener una política organizacional para planificar y llevar a cabo el proceso de planificación del testing. Algunas políticas que generalmente se especifican:
    - Cada proyecto Lelio definirá un plan de test que incluya un método de prueba, acompañado del esfuerzo y las estimaciones de llevar a cabo esas pruebas.
    - Cada método de prueba del proyecto derivará de la estrategia de prueba planteadas.
    - Los planes de prueba se deberán desarrollar usando procesos estándares y plantillas.
    - Se deberán usar herramientas estándares cuando se realice la planificación del testing.
- Planificar el proceso. se proporcionará tiempo suficiente a los gestores del testing para realizar las actividades en la planificación del testing. Las personas con experiencia en el aplicativo desarrollado en la empresa ayudarán dando soporte en la creación del plan de test. Se deberán usar herramientas del tipo:
    - herramientas de planificación de proyectos.
    - herramientas de estimación.
    - herramientas de análisis de riesgo.
    - herramientas para la gestión de pruebas.
    - herramientas de gestión de la configuración.

- Asignar responsabilidades. Se deberá inferir de autoridad y responsabilidad a una persona para que se encargue del proceso de planificación de las pruebas, de la documentación generada y que proporciona el servicio de planificación del proceso de pruebas. Es la figura del “test manager” la encargada de esta asignación. Entre sus actividades principales está la de negociar los compromisos con los interesados y de desarrollar y coordinar el plan de prueba.
- Entrenar al personal encargado de la planificación. Algunos tópicos que deberán aprender serán:
    - los principios de la planificación
    - la estrategia del testing
    - cómo organizar las pruebas.
    - una introducción a las técnicas de diseño de prueba.
    - las herramientas de soporte de planificación de pruebas.
    - planes de contingencia.
- Gestión de la configuración. El objetivo es colocar la información generada bajo niveles de control de configuración. Herramientas como Git, nos servirán para mantener versiones de copias de documentos generados y modificados durante el proceso de desarrollo. La herramienta MS Project, nos ayudará con el mantenimiento de la Estructura de Desglose del trabajo (EDT). Ejemplos de material generado puesto bajo control de configuración.
    - EDT
    - Datos de la estimación del tesing
    - Datos de evaluación de riesgos del producto software
    - El plan de prueba

Ya para cumplir con esta meta genérica se necesitará acudir al marco formal TMMi para su implementación puesto que muchas de las actividades no coinciden al 100% con nuestra proposición del Plan de puesta en marcha.

### 7.3.2. Meta específica: Realización de la evaluación de los riesgos

El objetivo de esta meta es el de realizar una evaluación del producto software a desarrollar con el objetivo de identificar las áreas críticas a probar. Esta meta no está del todo implementada en nuestro Plan de puesta en marcha, con lo cual la misma constituiría una buena práctica a implementar. A continuación, se enuncia un conjunto de prácticas específicas a cumplir para alcanzar esta meta:

- Crear listas ordenadas por categorías de los riesgos del producto. Las categorías de riesgos a definir serán las siguientes:
    - Riesgos funcionales.
    - Riesgos relacionados con el cambio, p.ej. regresiones.

  La realización de estas listas se consigue llevando a cabo las siguientes actividades: determinar las categorías de riesgo del producto software. Definir criterios consistentes para la evaluación y cuantificación de la probabilidad de riesgo del producto a desarrollar y los niveles de impactos.
- Identificar y analizar los riesgos del producto a desarrollar. Para esto se utilizarán las siguientes técnicas:
    - Brainstorming
    - Entrevistas con expertos
    - Experiencias aprendidas de otros productos

### 7.3.3. Meta específica: Establecer un método de prueba

Esta meta se lleva a cabo para, según los riesgos identificados se establecerá un método de prueba con el objetivo de identificar y mitigar los riesgos del producto. Esta meta no está del todo implementada en nuestro Plan de puesta en marcha, con lo cual la misma constituiría una buena práctica a implementar.

De entre las prácticas específicas a realizar tenemos la de realizar dos listas.

- lista de elementos susceptibles a ser probados y a no ser probados.
- lista de funcionalidades sensibles a ser probadas y a no ser probadas.

Con estás lista se definirán métodos de revisión de pruebas. Por ejemplo, para todos los elementos de alto riesgo se realizará un retesteo completo, es decir se reejecutará todo el

procedimiento de test por cada error que se encuentre. Para los elementos con bajo riesgo, los errores encontrados serán vuelto a probar, pero de forma aislada, es decir no se reejecutar todo el procedimiento de test sino ese error en concreto.

Otras prácticas a intentar implementar serían las de definir criterios de entrada y de salida, también definir los criterios de suspensión y reanudación. Estos criterios tienen que ver con el comienzo y el fin del proceso de ejecución de las pruebas, saber cuándo parar o saber, p.ej. cuándo reanudar un caso de prueba que se hubiese sido pospuesta su ejecución.

### 7.3.4. Meta específica: Hacer las estimaciones de las pruebas

Las estimaciones del testing son realizadas para usarlas a la hora de establecer los métodos de pruebas con las personas interesadas y en la planificación de las actividades del testing. Esta meta no presente en el Plan de puesta en marcha también constituiría una buena práctica a implementar.

Como práctica específica tenemos la de crear:

- La lista de pruebas del producto.
- La EDT para tener una definición clara de lo que se va a probar y su alcance.
- La lista de tareas de pruebas a realizar.

### 7.3.5. Meta específica: Desarrollar el plan de prueba

Tampoco esta meta está presente en el Plan de puesta en marcha propuesto y también constituiría una buena futura práctica a implementar por la empresa objeto de estudio. El desarrollo del plan de prueba se realiza para tener una base en qué apoyarse para gestionar el testing y la comunicación con los interesados. Este documento recoge parte de las actividades vistas hasta el momento, como la estimación del testing, los riesgos del proyecto de pruebas y planes de contingencias.

Como práctica específica está la de establecer la planificación del testing. Para ello:

- Se identificará las dependencias de las tareas del testing
- Se identificará las restricciones en la planificación del testing (duración de las tareas, los recursos y otras entradas necesarias)
- Se definirá la planificación del testing (duración de la actividad del testing, las fases en el ciclo de vida del test y los hitos para el testing)

Otra práctica especifica será la de planificar el personal para la realización del test (los testers)

El objetivo de esta planificación es para tener a disposición al personal necesario con el conocimiento y las competencias necesarias para realizar el testing.

Esta práctica nos genera:

- Los requisitos que tendrá que cumplir el personal que realizará el testing basándonos en la EDT y las estimaciones.
- Un listado de las competencias necesarias para la realización del test.
- Un plan del testing para selección de personal y de nuevas adquisiciones.
- Un plan de entrenamiento, esto es elegir un mecanismo para proporcionar el conocimiento y las competencias necesarias.

### 7.3.6. Meta específica: Obtener la aprobación del plan de prueba

Meta no presente como propuesta de mejora en el Plan de puesta en marcha propuesto y por tanto también sería una buena futura práctica a implementar por la empresa objeto de estudio. En esta meta se establecerán y mantendrán los compromisos relativos al plan de test. Se realizarán revisiones del plan de test para lograr alcanzar y entender mejor los compromisos relativos al testing. En estas revisiones participaran todos los interesados.

Como prácticas específicas tenemos la de:

- Revisará la planificación del test para realizar posibles ajustes.
- Volver a revisar la lista de riesgos del producto
- Obtener los compromisos con los interesados que intervengan en la ejecución del plan de test. Unos de los interesados que deberán participar en nuestro caso de estudio son los desarrolladores, p.ej. su compromiso de actualizar las tablas del apartado “Creación de la “Guía del desarrollador para la automatización del proyecto”” en caso de modificaciones.

Aún quedarían 2 áreas de procesos a implementar para cumplir con este nivel 2 del marco Área de proceso “Monitorización y control del testing” y Área de proceso “Entorno del testing” pero con las áreas ya mencionadas queda demostrado que mi propuesta está lo suficientemente alineada con los marcos estándares y con la empresa objeto de estudio.

# 8. Conclusiones y trabajo futuro

Conclusiones: en este trabajo se ha elaborado un plan destinado a mejorar la forma de trabajar del departamento de calidad del software de una empresa a través del uso de la automatización durante el proceso de ejecución de sus pruebas. Para esto no sólo se ha necesitado del uso de una herramienta de automatización para la ejecución de las pruebas, sino también de la realización de un análisis previo del ciclo de vida del producto software que desarrolla la empresa. Con el resultado de este análisis hemos podido conocer las carencias y necesidades presentes en su actual estrategia, lo que nos ha permitido realizar los ajustes pertinentes exactamente donde fuere requerido.

El diseño de un plan ad-hoc ha sido la estrategia clave para encaminar a la empresa hacia la mejora de sus procesos. Este plan ha necesitado no solo de la elección de las tecnologías adecuadas, también ha necesitado de otras actividades como la reingeniería de los procesos de testing, el diseño de una ventana de comunicación entre los procesos del desarrollo del software y los procesos del testing teniendo como apoyo un manual de buenas prácticas. Por otra parte, ha sido necesaria la creación de nuevos roles (p.ej. responsable del testing) y el diseño de planes de entrenamiento que permitirán adoptar los nuevos procedimientos de una forma no escalada. Y todo esto contando siempre con las principales restricciones de que el ritmo de trabajo de la empresa se vea afectado en lo más mínimo, su presupuesto actual no varíe y la calidad del producto software desarrollado se vea mejorada.

Trabajo futuro: como consecuencia de estas adopciones se ha logrado que la empresa tenga los ingredientes necesarios para una posible adopción de un framework formal de implantación y mejora del testing como es el TMM*i,* lo que en la actualidad es un camino impensable de emprender en la organización. Que mi propuesta sea la fuerza motriz y el impulso necesario para que los procedimientos de la empresa se acerquen a los estándares de la industria válida la alineación de dicha propuesta con las recomendaciones y principios del sector.

Como trabajo futuro he de destacar que hay dos pilares importantes a conseguir, el primero, el de comprobar que en la práctica realmente mi Plan de puesta en marcha proporciona los beneficios que su teoría demuestra y de no ser así, realizar las correspondientes correcciones (mejoras o replanteamientos) necesarias. Una vez que todo este plan haya conseguido acoplarse bien en la organización y rendir lo deseado, será el momento de edificar un segundo pilar, esto es apostar por una estandarización y formalización real del proceso de testing, preferiblemente y recomendable la adopción del marco de trabajo TMM*i*.

# 9. Referencias


1. International Software Testing Qualifications Board, consultado 5 mayo 2020: www.istqb.org
2. Kshirasagar N., Priyadarshi T.,. *Software Testing and Quality Assurance Theory and Practice*. New Jersey: John Wiley & Sons, Inc.
3. Brian Hambling, Peter Morgan, Angelina Samaroo, Geoff Thompson and Peter Williams. *Software Testing - An ISTQB-BCS Certified Tester Foundation guide 4th edition*. UK: BCS Learning and Development Ltd.
4. The Explosion of the Ariane 5, consultado 5 mayo 2020: http://www-users.math.umn.edu/~arnold/disasters/ariane.html
5. The Patriot Missile Failure, consultado 5 mayo 2020: http://www-users.math.umn.edu/~arnold/disasters/patriot.html
6. Death and Denial: The Failure of the THERAC-25, A Medical Linear Accelerator: http://users.csc.calpoly.edu/~jdalbey/SWE/Papers/THERAC25.html
7. The TMMi Foundation. *Test Maturity Model integration (TMMi®) Guidelines for Test Process Improvement Release 1.2*. Disponible en: Disponible en: https://tmmi.org/tm6/wp-content/uploads/2018/11/TMMi-Framework-R1-2.pdf
8. Puppeteer, consultado 14 abril 2020: https://pptr.dev/
9. Katalon Studio, consultado 31 agosto 2020: www.katalon.com/katalon-studio/
10. TestNG, consultado 31 agosto 2020: https://testng.org/
11. Selenium WebDriver, consultado 31 agosto 2020: https://www.selenium.dev/projects/
12. Ian Sommerville. *Software Engineering,* ed. 10. USA: Pearson Education
13. Ilene Burnstein. *Practical software testing*. New York: Springer-Verlag
14. T. Koomen, M. Pol. *Test Process Improvement: A step-by-step guide to structured testing*. Addison-Wesley
15. Sogeti. *TPI Next - Business Driven Test Process Improvement*. The Netherlands: Hertogenbosch
16. El País. *El cambio de neumáticos más rápido de la historia: 1,88s*, consultado 14 agosto 2020: https://elpais.com/deportes/2019/07/30/actualidad/1564504382_279190.html
17. W3C WebDriver, consultado 31 agosto 2020: https://w3c.github.io/webdriver/
18. Squash TM, consultado 31 agosto 2020: https://www.squashtest.com/pricing?lang=en
19. Confluence, consultado 31 agosto 2020: https://www.atlassian.com/software/confluence
20. Git, consultado 31 agosto 2020: https://git-scm.com/

# Bibliografía


1. ea2 Some disasters attributable to bad numerical computing, Douglas N. Arnold, consultado 5 mayo 2020: http://www-users.math.umn.edu/~arnold/index.html
2. Therac-25: https://en.wikipedia.org/wiki/Therac-25
3. Selenium vs. Puppeteer for Test Automation: Is a New Leader Emerging?, consultado 14 abril 2020: https://flood.io/blog/selenium-vs-puppeteer-for-test-automation-is-a-new-leader-emerging
4. Satya Avasarala. *Selenium WebDriver Practical Guide*. Birmingham: Packt Publishing Ltd.
5. Jasmine, consultado 14 abril 2020: https://jasmine.github.io/
6. Selenium Blog, consultado 13 abril 2020: https://www.selenium.dev/blog/2019/seleniumconf-london-2019/
7. ISTQB - Certified Tester Foundation Level Syllabus, version 2018. Disponible en: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html
8. Certified Tester Foundation Level Syllabus Agile Tester, version 2014. Disponible en: https://www.istqb.org/downloads/send/5-foundation-level-agile-tester/41-agile-tester-extension-syllabus.html
9. Jira, consultado 31 agosto 2020: https://www.atlassian.com/software/jira
10. The TMMi Foundation. *Test Maturity Model integration (TMMi®) Guidelines for Test Process Improvement Release 1.2*. Disponible en: https://tmmi.org/tm6/wp-content/uploads/2018/11/TMMi-Framework-R1-2.pdf
11. Libro de Estilos para la redacción de trabajos TFG/TFM de la EPS. Disponible en: https://maktub.eps.ua.es/servicios/gestorContenidos/contenidos/normativaEPS/Pdf/9910.pdf
12. Guía de Estilo Web de la Universidad de Cádiz. Disponible en: http://www.uca.es/recursos/doc/Unidades/Gab_Com_Mark/4652000591942010912 3.pdf