UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Proyecto de gestión de la información de una empresa de mensajería según metodologías AGILE

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA INDUSTRIAL

Autor: **Manuel Santos Romero**
Director: **Ana Nieto Morote**

Cartagena, 13 de Febrero de 2019

RESUMEN

El objetivo de este Trabajo de Fin de Máster es explicar las diferentes metodologías Agile para la gestión de proyectos de desarrollo de software, centrándose especialmente en la metodología Scrum, para la cual, además de una explicación teórica, se llevará a cabo una ejemplificación práctica, aplicando su utilización al desarrollo de un proyecto de software para una compañía de comercio electrónico.

Con este trabajo se pretende que el lector llegue a conocer, entender y saber aplicar la Metodología Agile y sus herramientas para el desarrollo de proyectos de software, así como conocer los principios y valores que rigen su utilización.

ABSTRACT

The purpose of this project is to explain the different Agile methodologies for managing software development projects, focusing especially on the Scrum methodology, for which, in addition to a theoretical explanation, a practical example will be carried out, applying its use to the development of a software project for an e-commerce company.

The aim of this project is for the reader to get to understand and know how to use the Agile Methodology and its tools for the development of software projects, as well as the principles and values that govern its use.

Contenido

1. Metodologías Agile

1.1. Introducción

El desarrollo ágil de software es un enfoque intuitivo e incremental para la toma de decisiones en los proyectos de desarrollo de software, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto, incluyendo el compromiso con la entrega oportuna y continua de software, los requisitos cambiantes, la simplicidad en el enfoque y los ciclos de desarrollo sostenible.

1.2. Historia de desarrollo de Software Agile

La gestión ágil de proyectos se remonta a 1957. Debido a la rudimentaria gestión de proyectos de la época, a los desarrolladores de software que en ese momento trabajaban para Motorola e IBM se les ocurrió la idea de un modelo de desarrollo de software, y crearon este modelo para combatir el modelo existente (el modelo de cascada) para el desarrollo de software y de gestión de proyectos. El modelo de cascada era la forma predominante de gestión de proyectos en ese momento y estaba en términos generales relacionado con la idea de los diagramas de Gantt y los diseños de las tareas que se deben realizar en determinados puntos del proyecto para que este pueda avanzar.

En general, debido a su microgestión para asegurar que los procesos y los pasos programados se cumplieran correctamente, el modelo de cascada no dejaba mucho margen de error. Así pues, se identificó la necesidad disponer de más flexibilidad en el proceso de desarrollo de software.

Posteriormente, en los años noventa se introdujeron algunos enfoques más versátiles, incluyendo procedimientos como el método de desarrollo de sistemas dinámicos (DSDM), la programación extrema (XP), la metodología Scrum o el desarrollo guiado por características, que se conoce como FDD. No obstante, aunque estos nuevos enfoques ocurrieron en la década de los 90, es decir, existían antes de Agile, hoy en día se consideran parte del manifiesto de Agile.

El manifiesto Agile se introdujo en 2001, cuando un número de individuos se reunieron para trabajar en la solución de los problemas que estaban teniendo y crear un proceso de desarrollo de software alternativo y algo diferente a lo que había existido en el mundo hasta ese momento. El resultado de esa reunión fue este manifiesto para el desarrollo de software ágil. Ese manifiesto para el desarrollo de software ágil tenía básicamente siete páginas y consistía en cuatro conceptos centrales. Esos conceptos eran:

- Individuos e interacciones por encima de procesos y herramientas.

- Software de trabajo por encima de documentación completa.
- Colaboración del cliente por encima de la negociación de contratos.
- Respuesta al cambio por encima de seguir un plan.

Desde entonces ha habido un crecimiento en la popularidad de la metodología Agile en todo el mundo. Ayudando a realizar proyectos de desarrollo de software con ciclos de lanzamiento más rápido, dentro del presupuesto y tolerantes al cambio, dando paso al crecimiento de lo que llamamos la Alianza Agile y la Alianza Scrum, que son esencialmente organizaciones globales sin fines de lucro que ayudan a promover Agile y Scrum en todo el mundo para hacer la vida de los desarrolladores de software más fácil y para ayudar a que los proyectos sean entregados a tiempo.

1.3. Características Agile

Existen una serie de características que definen la metodología Agile y que, aplicadas a los equipos de desarrollo, ayudan a mejorar la eficiencia en la gestión de proyectos:

- Equipos auto-organizados: las mejores arquitecturas, requisitos y diseños, en general, van a surgir de equipos auto-organizados, es decir, equipos que tienen la capacidad de organizarse a sí mismos.
- Equipos de auto-reflexión: a intervalos regulares en el proyecto, es importante que el equipo reflexione sobre lo que está haciendo. Eso les ayuda a ser más efectivos, ajustando su comportamiento en consecuencia de las conclusiones alcanzadas.
- Estrecha y frecuente colaboración entre cliente y desarrolladores: Permite adaptar al máximo nivel de detalle las especificaciones que el cliente desea para su producto.
- Comunicación cara a cara: uno de los principios más importantes de Agile es que la comunicación sea efectiva y eficiente, y esto se da en mayor medida cuando hay una conversación cara a cara dentro de un equipo de desarrollo, tanto a nivel interno como con el cliente.
- Requerimientos cambiantes: Se debe estar preparado para implementar cualquier cambio que el cliente desee en el desarrollo, con el fin de darle alguna ventaja competitiva.
- Medida del progreso: una pieza de software de trabajo es una buena medida de progreso, y con Agile, somos capaces de entregar software de trabajo con frecuencia desde una escala de tiempo muy corta a una escala de tiempo más larga, pero con preferencia por una más corta.

- Satisfacción del cliente: Una de las más altas prioridades dentro de Agile es satisfacer al cliente, lo que puede hacerse a través de una entrega temprana de un software que está bien probado y que se entrega continuamente.
- Ritmo sostenible: Agile y los procesos que implementa, en esencia, promueven el desarrollo sostenible. Los inversores, los desarrolladores, los clientes y los usuarios deberían poder mantener este ritmo indefinidamente si se hace correctamente.
- Atención al buen diseño: Se centra en la idea de mejora, de requerimientos continuos, de revisión, de desarrollo continuo, de que todo va de la mano y trabaja en conjunto para ayudar a dar buena atención y un buen diseño.
- Simplicidad: cuando tienes procesos que tratan de maximizarlo todo, entonces eso se presta a la simplicidad.

1.4. Ciclo de vida Agile

La principal característica del ciclo de vida Agile es que es un proceso repetitivo, ya que se trata de un ciclo iterativo o incremental con el objetivo de alcanzar el resultado deseado. El objetivo de estas iteraciones es generar soluciones al final de cada incremento (Sprint) para obtener el feedback del cliente y cambiar o añadir lo que este solicite, lo que permitirá una notable reducción del trabajo que habría que volver a realizar.

A pesar de que, como anteriormente se ha visto, existen diferentes metodologías Agile, todas ellas siguen el mismo enfoque básico. Al inicio de cada Sprint o iteración, se definen y analizan los requisitos en la planificación del Sprint, donde se añadirán las tareas a realizar durante ese Sprint a la bolsa de trabajo (backlog). Además, se revisarán estas tareas, para modificarlas, dividirlas o eliminarlas en caso de ser necesario. Una vez están claras las tareas a realizar se procede con el desarrollo del software. Cada vez que un desarrollo es finalizado por un programador, ese desarrollo deberá ser integrado dentro del proyecto completo, comprobar que no existen errores de compilación y que pasa con éxito los test unitarios pertinentes. Este proceso se llevará a cabo para cada desarrollo de cada tarea asignada para ese Sprint. Una vez todos los desarrollos han sido finalizados y probados con éxito se procede a la liberación, es decir, todos los desarrollos entregados para ese Sprint se unen y se integran con el resto del proyecto y se testea en un entorno de prueba. A continuación, el equipo de desarrolladores se reúne con clientes y Product Owners para hacer una demostración de la funcionalidad de los desarrollos realizados, si esta funcionalidad es la deseada por el cliente y es aceptada se procede a integrarla en el sistema de producción. En caso de no ser aceptada por el cliente será necesario un proceso de recopilación de cambios y ajustes a realizar en la siguiente iteración.

1.5. Valores Agile

El Manifiesto Ágil es un documento redactado en 2001 por 17 expertos en programación que supuso un cambio radical en la forma de desarrollar 'software'. Frente a los modelos tradicionales, excesivamente rígidos y alejados de las necesidades de los clientes, estos gurús propusieron cuatro valores que inspiran las diferentes metodologías ágiles que han surgido desde entonces. Aunque nació en el mundo del 'software', la filosofía que promueve este manifiesto es extensible al desarrollo de cualquier otro producto

A continuación, se desarrollarán los cuatro valores del Manifiesto Agile.

El primero son los individuos y las interacciones por encima de los procesos y las herramientas. Crear un enfoque centrado en un equipo con interacción y colaboración definitivamente puede ayudar a que los miembros del equipo sientan que son dueños de lo que están haciendo. Es esencial estar enfocados en la gente, es decir, en el equipo que en este caso es el encargado de desarrollar un software, de manera que se pueda aumentar y mejorar la interacción entre los miembros del equipo, lo que ayudará a obtener unos mejores resultados en el desarrollo del proyecto. Esto se consigue con comunicación clara entre los miembros del equipo, con la posibilidad y el apoyo a la innovación y la posibilidad de disponer de un espacio para ello. Esto propicia el trabajo en equipo, que permite a los miembros del equipo trabajar juntos de manera más eficaz y creará adaptabilidad dentro del equipo, es decir, la gente será capaz de adaptarse rápidamente a los cambios en el proyecto.

La segunda parte del Manifiesto Agile se centra en la idea de trabajar con software por encima de documentación completa. El principal motivo es que con la documentación puede haber una carga de información pesada que va a ralentizar el trabajo, por lo que la metodología Agile intenta alejarse de eso, de manera que los documentos creados apoyen el desarrollo del producto, en lugar de frenarlo. No deben centrarse realmente en lo que se ha hecho hasta ahora o en lo que se hará, sino que están enfocados en la funcionalidad del software que se va a construir, lo que ayuda a obtener unos resultados más favorables, al dedicar más tiempo al propio desarrollo que a leer o a desarrollar documentación. Esto puede conseguirse, por ejemplo, mediante la revisión e iteración regular del código, que puede ayudar a solucionar problemas en lugar de pasar todo el tiempo tratando de descubrirlos; definición de las características del proyecto para que estas características se completen más rápidamente; y requisitos claramente documentados, de manera que cuando empiece a centrarse en piezas discretas de funcionalidad, se podrá escribir una mejor documentación para ellas de forma clara y concisa.

El tercer valor que se verá sobre el Manifiesto Agile es la idea de la colaboración del cliente por encima de la negociación de contratos. Es de suma importancia mantener un contacto constante con el cliente, siempre con la idea principal de la colaboración por encima del conflicto, es decir, involucrar al cliente en el desarrollo del proyecto intentando siempre evitar fricciones y limar asperezas. Esto se consigue con intercambio

de información y de opiniones continuo a través de softwares de comunicación, de manera que el cliente pueda mantenerse al día en la evolución del proyecto y que exista entre cliente y equipo de desarrollo total transparencia. Por otra parte, los clientes también pueden colaborar en esta tarea, modificando los requisitos de comunicación, ayudando así a crear una relación más fuerte con el equipo de desarrollo, de manera que el cliente tenga un aporte de manera constante y consistente. Gracias a estas colaboraciones y el aumento de la confianza que suponen sobre el cliente es posible la firma de contratos flexibles, que permita hacer cambios sobre el proyecto para mejorar los procesos, y además permite la existencia de la llamada responsabilidad mutua, donde cliente y equipo de desarrollo tienen de alguna manera, responsabilidad sobre los resultados del proyecto.

El cuarto concepto del Manifiesto será la idea de responder al cambio en lugar de seguir un plan. Esto se consigue a través de un sistema flexible de gestión de proyectos para ayudar a responder al cambio, buscando siempre la máxima adaptabilidad y la respuesta más rápida al cambio. Es importante tener claro que muchos factores pueden cambiar durante el desarrollo de un proyecto, por lo que es fundamental disponer de las herramientas suficientes para gestionar esos cambios y que ayuden a responder y adaptarse lo mejor posible. Otro enfoque para hacer frente a los cambios que se presenten es convertir los desafíos que esto presenta en beneficios, es decir, ver los cambios como ventajas que pueden ayudar al proyecto a adaptarse y a los miembros el equipo a aprender más rápidamente, en lugar de verlo como un inconveniente que va a ralentizar el desarrollo del proyecto. El cambio es algo que vendrá y que puede ser predecible, de modo que, a través del seguimiento del progreso de manera regular y consistente, y a la transparencia dentro del proyecto será más sencillo identificar y predecir los cambios futuros que serán necesarios.

1.6. Principios Agile

Estos cuatro valores se concretan en 12 principios, que definen el marco de trabajo de cualquier equipo ágil:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

1.7. Buenas prácticas Agile

Dentro de las buenas prácticas Agile hablaremos sobre la lista de objetivos/requisitos (Backlog) y sobre la persona responsable de asegurar que se cumplen las buenas prácticas y valores.

La lista priorizada de objetivos/requisitos representa la visión y expectativas del cliente respecto a los objetivos y entregas del proyecto.

El backlog tiene las siguientes características: los objetivos/requisitos, también conocidos como Historias de Usuario, se expresan detalladamente, asignándoles el valor que suponen y el coste estimado para su ejecución; a la hora de priorizar objetivos, el criterio que se sigue es la relación entre el valor de cada tarea para el conjunto del proyecto y el coste que supone su puesta en marcha, es decir, se analiza su inversión; en la lista se expresan las iteraciones y los plazos para la ejecución de cada objetivo/requisito, teniendo en cuenta para ello la capacidad productiva de los equipos involucrados en la ejecución; debe contener una columna en la que se reflejen los riesgos de cada iteración y algunas actividades para mitigarlos, evitarlos o, si es el caso, eliminarlos del todo.

2. Metodología Scrum

Scrum es reconocida por ser la metodología ágil más prestigiosa internacionalmente en el sector empresarial. De ella sabemos la rápida aceptación que ha tenido desde su creación, en el año 1992, cuando el teórico norteamericano Jeff Sutherland sentó las bases para su posterior desarrollo.

2.1. Características.

La metodología Agile Scrum será con la que se desarrollará la parte práctica de este proyecto. El desarrollo de software Scrum es un framework ágil y se utiliza para completar proyectos complejos. Es importante notar que Scrum no es un proceso o una técnica para construir productos, más bien es un marco de trabajo que emplea varios procesos y técnicas para ayudar a que el proyecto avance

Este tipo de desarrollo Agile se remonta a los años 90 y está basado principalmente en un proceso iterativo formado por periodos cortos de tiempo llamados Sprints. Cada iteración o sprint tiene que proporcionar un resultado completo, un incremento de producto que sea potencialmente entregable, de manera que cuando el cliente (Product Owner) lo solicite sólo sea necesario un esfuerzo mínimo para que el producto esté disponible para ser utilizado.

La metodología Scrum posee tres características clave: agilidad, facilidad de comprender y dificultad de dominar.

Uno de los componentes principales del Scrum son los Sprints. Como se dijo anteriormente, un sprint es un periodo de tiempo en el que realizamos el trabajo. Los Sprints tienen carácter secuencial, es decir, un nuevo Sprint no empieza hasta que el anterior no ha finalizado. El contenido de cada Sprint se diseña durante el llamado 'Planificación de Sprint'. Esta planificación se realiza al inicio de cada Sprint y debe tener una duración no superior a 8 horas para un Sprint de un mes. Así pues, un Sprint no debe superar un mes de duración. Durante la planificación de Sprint, los miembros del equipo se reúnen para decidir y priorizar el trabajo que será llevado a cabo durante el nuevo Sprint. También, a lo largo de los Sprints, se realizan reuniones diarias, conocidas como Scrum dailys o dstum (daily stand up meeting), donde los miembros del equipo se reúnen, normalmente al inicio de la jornada laboral para hablar sobre el trabajo realizado el día anterior, el trabajo planificado para ese día y los posibles problemas o impedimentos que hayan podido darse. Luego está la revisión del sprint, que generalmente se lleva a cabo al final del sprint para inspeccionar lo que sucedió y adaptar la cantidad de productos pendientes, si es necesario. Y finalmente, está la retrospectiva del sprint, que es una oportunidad para que el equipo se inspeccione a sí mismo y a los procesos.

2.2. Roles Scrum

El primer rol que se expondrá es de Propietario del Producto o Product Owner. El propietario del producto es probablemente uno de los roles más importantes en Scrum. Se interrelacionan con todas las partes interesadas, por lo que se encuentran entre el equipo de desarrollo, los financiadores, los vendedores y todos los demás que participan en el producto, además también son responsables del éxito del producto y de los equipos. Entre sus tareas y responsabilidades se incluyen: desarrollar la visión del producto, es decir, las partes interesadas delegan o informan al propietario de producto lo que quieren que se haga, y este tiene que desarrollar la manera de trasladar esos deseaos o requerimientos al proyecto; recolectar y priorizar requisitos, en función de las necesidades de las partes interesadas y de las capacidades del equipo de desarrollo; controlar el presupuesto y supervisar el rendimiento de la inversión, es decir, se sitúan entre el lado del marketing y de las partes interesados y el lado de los desarrolladores. El rol del Product Owner lo asume una persona con ciertas habilidades y características, incluyendo disponibilidad, la comprensión del negocio y habilidades de comunicación: Primero, el Product Owner necesita estar disponible para su equipo. Los mejores Product Owners muestran su compromiso por hacer lo que sea necesario para construir el mejor producto posible, y eso significa comprometerse activamente con su equipo; tener conocimiento de negocios es importante para el Product Owner, porque es quien toma las decisiones de las características que el producto tendrá. Eso significa que, el PO debe entender el mercado, al cliente y el negocio, a fin de tomar decisiones acertadas; finalmente, la comunicación es una gran responsabilidad del Product Owner. El rol del PO requiere trabajar de forma muy cercana con las principales partes interesadas (en la organización y más allá de ella), así él o ella será capaz de comunicar diferentes mensajes a diferentes personas acerca del proyecto en todo momento.

El segundo rol que va a tratarse, y del que ya se vio algo anteriormente, será el de Scrum Master. El papel del Scrum Master es principalmente el de asesoramiento, alguien dentro del equipo que pueda aconsejar y guiar al resto, de manera que más que como un líder es visto como un “entrenador” del equipo. Ellos se aseguran de que las reglas de Scrum se cumplan y de que los miembros del equipo no se alejen de ellas, proporcionándoles apoyo cuando sea necesario, guiándoles a la hora de proceder y ejecutar correctamente su trabajo. Aunque los Scrum Master no forman parte del equipo de desarrollo, si poseen de conocimiento técnico suficiente para aconsejar sobre la utilización de métodos y tecnologías correctas. Otra responsabilidad del Scrum Master sería eliminar impedimentos, es decir, si hay algo que impide que uno de los miembros del equipo pueda realizar su labor correctamente, entonces el Scrum Master ayudaría a eliminar cualquier impedimento que exista trabajando junto con el equipo para resolverlo. El Scrum Master también se encarga de gestionar y superar los obstáculos organizativos y estratégicos, intentando no involucrar al equipo de desarrollo de manera que estos solo se preocupen por realizar correctamente su trabajo, sin que asuntos que están fuera de su competencia o responsabilidad puedan distraerles o perturbarles. Además, el Scrum Master es bueno para manejar las expectativas de las partes interesadas. Estas expectativas giran generalmente en torno a lo que puede realizarse dentro de un cierto marco de tiempo. Asimismo, también desarrollan y mantienen condiciones óptimas para

el rendimiento, asegurándose de que el equipo tenga lo que necesita y de que todos los sistemas estén configurados de la manera que necesitan. Y en general, se aseguran de que todo esté preparado para que el producto pueda ser entregado.

El Scrum Master es la persona que esencialmente guía al equipo. Tendría una figura similar a la de un coach/mentor que acompañará al equipo durante todo el desarrollo del proyecto y asegurará que se cumplan las buenas prácticas, actuando como un facilitador y solucionador de problemas. Entre sus labores están: Gestionar el trabajo retrasado con las principales partes interesadas, es decir, cliente y equipo de desarrollo, discutiendo las causas que están creando ese retraso, priorizando y configurando correctamente la gestión de esos retrasos, y desarrollando las soluciones para evitar que se repitan los mismos fallos; ayudan a prevenir que se llegue a compromisos excesivos con el cliente; organización de la planificación de los Sprint y de que las fechas estén fijadas, así como la definición de los objetivos del sprint; asesorar y formar a los diferentes miembros para trabajar de forma autoorganizada y con responsabilidad de equipo; moderar las reuniones y gestionar las dinámicas de grupo en el equipo.

El tercer y último rol será el equipo de desarrollo. El equipo de desarrollo es una de las partes más importantes dentro de Scrum, ya que son las personas encargadas de construir los códigos y desarrollar los softwares. Uno de los rasgos más característicos del equipo de desarrollo es que son autoorganizados. El Scrum Master no los organiza ni lidera, sino que esta para ayudarlos en lo que necesiten y facilitarles el trabajo. Además, los objetivos del equipo son autodeterminados, es decir, ellos mismos se marcan los objetivos que se impondrán. También ayudan a determinar los requisitos que se deben cumplir durante un sprint, planificando y estimando las tareas que deben desempeñar mediante el “Planning poker”. El objetivo de Planning Poker es obtener una estimación consensuada y validada por todo el equipo. Para ello, todo el equipo se encierra para estimar, y todos conocen lo que se va a estimar. Si hay gente que no está al tanto de lo que se va a estimar la sesión debe comenzar con una explicación y una sesión de preguntas para despejar cualquier duda sobre las tareas que se van a estimar. Una por una se leen y discuten las tareas. Una vez todos tienen claro en qué consiste cada uno elige una carta de una baraja en la cual hay En cada baraja hay una pseudo-secuencia de Fibonacci modificada. Así cada participante tendrá las siguientes cartas: 0,1/2, 1, 2, 3, 5, 8, 13, infinito y ?. El cero significa que la historia ya está hecha o no requiere ningún esfuerzo, el interrogante significa que nos falta información para estimar esa historia o tarea y el infinito es que es demasiado grande y hay que trocearla, en función del esfuerzo que prevé requerirá esa tarea. No es posible seleccionar un valor no incluido en la baraja. Solo estiman los que después desarrollan (ni el Scrum Master ni el Product Owner estiman, solo resuelven dudas). Si no hay consenso, se abre la discusión. No muy larga, por ejemplo, se puede hacer que explique su elección el que tiene la puntuación más baja y más alta. Se repite la estimación nuevamente en busca de consenso. Si no se consigue a la segunda se vuelve a discutir. A la tercera si no hay consenso se escoge o bien la media o bien el máximo (mejor el máximo). Al final de la sesión el resultado es una estimación consensuada y validada por todo el equipo para cada una de las historias o tareas seleccionadas. Si se trabaja con Sprints lo habitual es

estimar alto nivel cuando los elementos entran en el backlog, y nuevamente cuando se realiza el sprint planning o backlog grooming si se hace por separado para estimar las tareas a bajo nivel.

2.3. Flujo de trabajo Scrum

El flujo de trabajo de Scrum comienza con el Product Backlog (cartera de productos). Se trata de una lista de características o requisitos que el producto tiene que cumplir, a los que se les da prioridad y se ponen en una lista de espera para que el equipo las vaya retirando y desarrollando. El siguiente paso sería la planificación del sprint, que consiste en una reunión en la que todos los miembros del equipo Scrum se reúnen para revisar el backlog y determinar a partir de este lo que se puede hacer dentro de los límites y limitaciones de un sprint. A partir de ahí, se crea sprint backlog, es decir, la lista de tareas o actividades a realizar en ese sprint. Así que las tareas elegidas se trasladan del Product Backlog al Sprint Backlog. Donde pueden ser estimados, pueden ser asignados a individuos y puede haber discusión sobre su desarrollo, por tanto, el sprint backlog permite ver cuál es el trabajo que hay que hacer en este sprint. Una vez que el sprint ha comenzado, normalmente disponemos de un mes o menos para completar todas las tareas de sprint backlog. Además, todos los días hay un Scrum diario o daily, una breve reunión de pie donde todo el mundo habla de lo que ha hecho el día anterior y de lo que va a trabajar ese día, además de cualquier impedimento que pueda tener. Luego, una vez que el sprint ha terminado, tenemos una revisión del sprint, que analizará el trabajo que se ha completado dentro del sprint. Y una retrospectiva del sprint, en la que se analiza lo que ocurrió durante el sprint, qué procesos se utilizaron qué impedimentos se produjeron y cómo se puede mejorar y trabajar en ello para que en el próximo sprint no se repitan los mismos problemas. Finalmente, tenemos el requisito de entrega. Una vez que las tareas han sido revisadas en el sprint y todo funciona correctamente, entonces básicamente se introducen en el producto se comienza otro incremento. Volvemos a la salida para hacer una retrospectiva del sprint y luego volvemos a la planificación del sprint, donde sacamos las cosas del Project Backlog y empezamos de nuevo con nuestro sprint.

2.4. Sprint Planning

El trabajo a realizar durante un sprint se decide en el llamado Sprint Planning, que tiene lugar al comienzo de cada sprint. Este proceso es llevado a cabo por el equipo scrum y debe tener una duración no superior a 8 horas para un sprint de un mes. En la planificación de sprint o sprint planning, lo que se hace básicamente es determinar lo que puede ser entregado para el incremento resultante de ese sprint, así como preguntarse cómo se llevará a cabo el trabajo necesario para lograr ese incremento. En

el sprint planning el equipo de desarrollo trabaja para predecir la funcionalidad de las tareas que se van a desarrollar en ese sprint, el producto owner estudia y discute los objetivos que se han marcado y analiza cual es la capacidad del proyecto para desarrollarla y examina el product backlog para ver cuántos de esos ítems podrían ser alcanzados dentro del sprint, es decir, la meta del sprint.

La meta del sprint es un objetivo establecido para el sprint, que luego podrá ser alcanzado a través de la implementación del backlog, cuyos productos seleccionados ofrecen una función coherente, y ese es generalmente el objetivo del sprint. Además, la meta proporciona orientación al equipo de desarrollo sobre por qué está construyendo ese incremento y le da también cierta flexibilidad en cuanto a la funcionalidad que se implementa dentro del sprint. Además, si el trabajo resulta ser diferente de lo que el equipo de desarrollo esperaba, pueden trabajar con el Product Owner y negociar el alcance del retraso del sprint dentro del sprint actual.

Una vez establecido el objetivo se trabaja en determinar cómo lograr ese resultado. El equipo de desarrollo decide cómo construirá esa funcionalidad como un producto hecho o un incremento entregable. Se seleccionan los ítems del product backlog para el sprint, a los cuales se les asigna una fecha de entrega en ese sprint, pasando a formar parte de lo que se conoce como sprint backlog.

El equipo de desarrollo suele empezar por diseñar la gestión de tareas y el trabajo necesario para convertir el product backlog en el incremento de productos en funcionamiento. No obstante, el trabajo planeado para los primeros días del sprint por el equipo de desarrollo puede ser descompuesto, a menudo en unidades más pequeñas de un día o menos. Además, si el equipo desarrollo considera que tiene demasiado tiempo o poco tiempo para el desarrollo del trabajo planificado, puede negociar los elementos del product backlog con el propietario del producto para cambiar o modificar lo que se va a hacer y el producto que se va a crear por el incremento al final del sprint.

2.5. Daily Scrum

La reunión de scrum diaria es una reunión que se lleva a cabo en el mismo lugar y a la misma hora cada día. Idealmente, una reunión de scrum se lleva a cabo en la mañana y eso ayuda a establecer el contexto para el trabajo de ese día.

Generalmente, las reuniones de scrum están limitadas en el tiempo, estrictamente a 15 minutos. Esto ayuda a mantener la discusión viva y relevante. Se requiere que todos los miembros del equipo asistan a la reunión del scrum. Es importante notar que la reunión de scrum no es usada como una reunión de resolución de problemas, sino que en caso de que estos existan, son tratados por las personas pertinentes inmediatamente después de la reunión. Tampoco es una reunión de actualización de estado en la que un jefe está recopilando información sobre quién está atrasado o lo que sea, sino que es una reunión en la que los miembros del equipo se comprometen entre sí, ayudando al

equipo de desarrollo a determinar su progreso hacia la meta del sprint e inspeccionarla cada día. Además, se utiliza para sincronizar eventos y crear un plan para las próximas 24 horas.

2.6. Revisión y retrospectiva del sprint

La revisión de sprint es una reunión informal, no una reunión de estado, que se celebra al final del sprint y que se utiliza para inspeccionar el incremento del proyecto y también para adaptar el product backlog, si es necesario. También tiene la intención de fomentar la retroalimentación y la colaboración dentro del equipo. Normalmente suelen tener una duración de cuatro horas para un sprint de un mes, pudiendo ser más reducida en caso de un sprint de menor duración. Una revisión de sprint incluye al equipo de Scrum y a las partes interesadas clave que podrán ser invitadas por el product owner. El product owner será quien explicará que productos se han acumulado y que cuales se han finalizado con éxito, abriéndose una discusión sobre el trabajo atrasado y los motivos de este retraso. El equipo de desarrollo discutirá que fue bien durante el sprint, con qué problemas se topó, cómo se resolvieron los problemas y responderá cualquier pregunta sobre el incremento. También hay una revisión de la línea de tiempo, el presupuesto, las capacidades y el mercado para la próxima versión anticipada del producto.

Después de la revisión del sprint, pero antes de la planificación del sprint siguiente tiene lugar la retrospectiva del sprint. El propósito de la retrospectiva de sprint es permitir que el equipo Scrum inspeccione cómo fue el último sprint, en lo concerniente a personas, relaciones, procesos y herramientas. Además, puede ayudar al equipo a crear un plan para implementar mejoras a la hora de realizar el trabajo. Es una reunión de tres horas de duración para un sprint de un mes de duración. El Scrum Master también participa como miembro del equipo en esta reunión. Así que, desde el punto de vista de la rendición de cuentas sobre el proceso de Scrum, se le trata como a un igual dentro de la reunión. Al final de la retrospectiva del sprint, el equipo Scrum deberá haber identificado las mejoras que deben hacerse, que pueden ser implementadas en el próximo sprint, lo que normalmente se hace inspeccionando cómo fue el último sprint. Finalmente, se adaptará el proceso para asegurar que se logren esas mejoras.

2.7. Refinamiento del Backlog

El refinamiento o acondicionamiento del backlog (Backlog grooming) es una actividad que ocurre regularmente, y que puede ser tanto una reunión oficial aislada o una actividad continua. Normalmente, este refinamiento se da cuando el Project owner u otro miembro del equipo scrum revisa los ítems del product backlog y se asegura que contiene los ítems apropiados, añadiendo requisitos, eliminándolos, modificándolos, repriorizándolos (cambiando el contenido de iteraciones, definiendo

un calendario de entregas que se ajuste mejor a sus nuevas necesidades) y detallando los requisitos conforme se acerca el momento de su desarrollo. Para llevar a cabo esta repriorización con toda la información necesaria, el equipo proporciona la estimación de los nuevos ítems y el impacto de los cambios que se necesita realizar. También debe asegurarse de que los primeros ítems de la lista están listos para la entrega. Los motivos de estos cambios en el backlog son diversos: Pueden ser debido a modificaciones que desea el cliente tras llevarse a cabo la demostración de la nueva iteración; a cambios en el contexto del producto; a la creación de nuevas tareas con el fin de prever nuevas amenazas que hayan podido surgir, etc.

Una de los principales beneficios del refinamiento es que permite que todos los ítems que contiene el backlog son relevantes para el producto así como asegurarse de que todos los ítems están correctamente detallados y estimados. Siempre y cuando haya un número suficiente de tareas o requisitos en el backlog podrán ser utilizadas para programar las próximas iteraciones. Además, el refinamiento puede ayudar a evitar la que el alcance se desvíe a requisitos que tal vez no han sido eliminados y que ya no tienen relevancia, evitando también problemas como sobrecostes y excesos de programación.

2.8. Limitaciones de Scrum

A pesar de las muchas ventajas que tiene la implantación de la metodología Scrum para el desarrollo de un proyecto. En algunos casos, Scrum no satisface la necesidad de la gerencia de controlar el presupuesto, el alcance y el calendario, ya que, al trabajar con estimaciones aproximadas de costo y calendario, puede generar incertidumbre en torno al producto final. Por tanto, para evitar esto se requiere una definición exhaustiva de las tareas y los plazos, sino se definen adecuadamente, Scrum se desvanece.

Otro aspecto que puede generar conflictos es que Scrum se basa en equipos reducidos, autoorganizados y una intensa colaboración, que puede ser incompatible con la estructura organizativa tradicional, que se basa en una estructura jerárquica.

Existe el riesgo de que el proyecto fracase si cada individuo que participa en él no se muestra cooperativo y comprometido. Además, exige una alta cualificación. No es una modalidad de gestión apropiada para individuos en formación o con perfil junior, sino que su éxito radica en la colaboración de un grupo de individuos con alta experiencia y cualificación.

También pueden plantearse dificultades para el Scrum Master a la hora de planificar y organizar un proyecto que carece de una definición clara, ya que con Scrum solamente el próximo backlog de la iteración es definido con claridad. Otra dificultad o inconveniente que puede tener que lidiar el Scrum Master es que las daily scrum y las revisiones frecuentes puedan llegar a resultar frustrantes y onerosas para el equipo, además de suponer en ocasiones un requerimiento substancial de recursos.

2.9. Valores Scrum

Los valoren que definen la metodología Scrum son: Compromiso, foco, franqueza, respeto y coraje.

- Compromiso: Estar comprometido a colaborar y aprender con el equipo, con la transparencia, con autoorganizarse cada vez mejor, con ser proactivos, con la meta del sprint, y con el producto desarrollado y con su calidad. El compromiso se trata de una dedicación aplicada a acciones y esfuerzo, y nunca a resultados finales, ya que un compromiso sobre los resultados finales no es posible, pues el desarrollo de software es un mundo altamente complejo e impredecible, por lo que compromiso cerrado e inviolable en alcance sencillamente es imposible y solo conduce a la frustración.
- El enemigo numero uno de la productividad es la multitarea, la falta de detalles o especificación de las tareas y los objetivos, y la dispersión. Para evitar esto, Scrum dicta que el equipo debe centrarse en las necesidades momentáneas del proyecto y del sprint actual, ya que el futuro es incierto y vagamente definido, por lo que centrarse en tareas futuras supondrá una perdida de tiempo y dinero. YAGNI (You Ain't Gonna Need It) o KISS (Keep It Simple Stupid). Foco es concentrarse en cumplir la meta del sprint y finalizar el backlog de ese sprint, trabajando en entregar el máximo valor posible en cada momento. Esto se consigue si la meta del sprint y backlog han sido bien definidos y priorizados.
- Scrum defiende la transparencia como un pilar básico del empirismo sobre el que se sustenta. Sin transparencia es imposible llevar a cabo la Inspección y la Adaptación. El equipo debe ser abierto sobre su trabajo, su progreso y sus problemas. Los miembros del equipo deben tener una actitud abierta y proactiva para mejorar sus capacidades y competencias profesionales, para la colaboración entre disciplinas y habilidades, para compartir opiniones y aprender unos de otros. Además, debe estar abierto al cambio, ya que la organización en la gestión de proyectos de software cambia de manera impredecible, inesperada, frecuente y constante.
- Cada miembro del equipo respeta y valora la experiencia y los conocimientos no solo del resto del equipo, sino también del resto de personas que intervienen de alguna manera en el proceso de desarrollo del proyecto. Se respeta la diversidad y se aceptan los diferentes puntos de vista. Respetamos el marco Scrum, compartiendo información, fomentando un espíritu colaborador, aprendiendo y tomando decisiones juntos. Los miembros de un equipo Scrum se respetan entre ellos comprendiendo sus roles Scrum y la responsabilidad de cada uno dentro del equipo. Respeto por el entorno en el que se desarrolla el proyecto, participando para transferir conocimiento y experiencia, y no aislándose.
- El equipo debe mostrar coraje para hacer lo correcto, construyendo lo que la gente quiere y necesita. Admitir que los requisitos nunca serán perfectos y que no todo puede planificarse debido a la complejidad y la realidad. Además, debe

reconocerse el valor de considerar que el cambio es una fuente de bien, inspiración e innovación. El valor de no entregar software incompleto o insatisfactorio. La valentía de compartir toda la información. El coraje para cambiar de dirección cuando sea necesario y compartir los riesgos y los beneficios de esos riesgos. Y finalmente, mostrar el valor de apoyar los valores de Scrum.

2.10. Scrumban

Scrumban es una evolución de Scrum complementado con prácticas básicas Kanban. Scrum es el más adecuado para el desarrollo de productos y proyectos de desarrollo web. El Kanban es el mejor para el soporte de producción. Así que Scrumban combina las mejores características de Scrum y Kanban, y se utiliza mucho en proyectos de mantenimiento. Como se verá más adelante con mayor detalle, Kanban consiste en la visualización del flujo de trabajo. Divide el trabajo en elementos más pequeños. Por lo general, las tareas se escriben en una tarjeta y luego se colocan en la pared en diferentes columnas con nombre para ilustrar dónde está cada elemento en el flujo de trabajo. La tabla de estilo Kanban incluye una tabla que contiene tres columnas: TO-DO, DOING, y DONE]. Además, limita el trabajo en curso, asignando límites explícitos sobre cuántos elementos pueden estar en progreso en cada estado, en cada estado de flujo de trabajo. A continuación, se mide el tiempo de entrega, que es básicamente el tiempo promedio para completar una tarea, y luego se optimiza el proceso para que el tiempo de entrega sea lo más pequeño y predecible posible. Scrumban básicamente utiliza el tipo de naturaleza prescriptiva de Scrum para ayudar a que sea ágil, para crear agilidad, utilizando la mejora de procesos de Kanban para permitir que el equipo mejore continuamente su proceso. También utiliza el sistema pull de Kanban, por lo que los flujos de trabajo se vuelven más fluidos a medida que la capacidad mejora. El tiempo medio de entrega y el tiempo de ciclo se convertirán en el foco principal del rendimiento, y en lugar de pasar por el problema de estimar el alcance del trabajo para cada una de las iteraciones, en Scrumban, simplemente fijamos el tamaño del trabajo atrasado. Además de eso, esta es la idea de Kaizen o mejora continua. Mejoramos la calidad y también minimizamos y reducimos los desperdicios, ya que todo lo que no agrega valor al cliente que contratamos

Será especialmente adecuado y útil el uso de Scrumban en aquellos equipos de sprint que se centran en el desarrollo de nuevos productos; en proyectos con cambios frecuentes o historias de usuarios inesperadas; en proyectos con errores de programación frecuentes; para proyectos que gestionan comunidades de mejora después del despliegue, así como para trabajar en proyectos de mantenimiento. Si Scrum tiene problemas de flujo de trabajo o de recursos, puede implementar Scrumban en su lugar. Por lo tanto, algunas de las mejores prácticas para el atraso de Scrumban serían la priorización bajo demanda. Siempre debe proporcionar al equipo lo mejor para trabajar a continuación. Ni más ni menos, así que todo es priorizar según la demanda.

El tablero Scrumban proporciona una excelente visión del flujo de trabajo en un proyecto. Informa sobre las tareas pendientes de hacer, las tareas en desarrollo, las tareas finalizadas y las tareas revisadas y aceptadas.

Tablero Scrumban

2.11. Scrum de Scrums

El Scrum de Scrums es una técnica que se usa para escalar el enfoque Scrum para grandes equipos. Como se mencionó antes, les equipos Scrum deben de ser de tamaño reducido, sin superar los 9 miembros. En el caso de que se requieran más miembros para el desarrollo del producto, en lugar de hacer crecer el equipo, el enfoque Agile es crear varios equipos para distribuir el trabajo entre ellos. Para ello es necesaria una forma de coordinación eficaz, que facilite la comunicación entre los equipos y la fusión de las entregas al final del Sprint.

El Scrum de Scrums es la solución para este problema. Tiene el formato de una Scrum daily en la que participan un representante de cada equipo. Estas reuniones pueden acontecer todos los días de la semana, hasta un día a la semana como mínimo. Debe hacerse después de las Scrum daily, por lo que es conveniente que todos los equipos las realicen en paralelo. Una vez finalizada la Scrum daily, los representantes de cada equipo se reúnen en la Scrum de Scrums. Este representante este debe de ser el miembro del equipo que más involucrado esté con el desarrollo del objetivo del sprint, por tanto a sucesivos Scrum de Scrums pueden acudir diferentes representantes. La reunión no debe superar la media hora de duración, y aunque es el Scrum Master el que la facilita, ningún Scrum Master de ningún equipo debe participar en la Scrum de Scrums, ya que el representante debe tener poder de decisión como equipo de desarrollo y que adquiera los compromisos acordados en nombre de su equipo.

3. Otras metodologías Agile

Sin embargo, dentro del panorama ágil existen, además de Scrum, otras opciones y cada una de ellas, cuenta con una forma diferente de entender la flexibilidad. A continuación, se explicarán cada una de estas **metodologías ágiles de gestión de proyectos**:

3.1. Agile Modeling

La siguiente metodología Agile que veremos es el llamado Agile Modeling. El Agile Modeling (AM) es una metodología para modelar y documentar sistemas de software basados en las mejores prácticas. Es una colección de valores y principios, que pueden ser aplicados en un proyecto de desarrollo de software (ágil). Esta metodología es más flexible que los métodos tradicionales de modelado, lo que hace que se adapte mejor a un entorno de cambios rápidos. Forma parte del kit de herramientas de desarrollo de software ágil, es decir, es un complemento para otras metodologías de desarrollo Agile; validación de productos y la verificación de la calidad, asegurándose de que lo que se está construyendo es realmente lo que el cliente necesita, y también se aseguran de que la calidad que se está generando coincida con las expectativas del cliente.

Uno de los pilares de esta metodología es la gestión de la documentación. La documentación se realiza a lo largo de todo el ciclo de vida, de manera constante, en paralelo a la creación del resto de la solución; la documentación se realiza lo más tarde posible, evitando ideas especulativas que puedan cambiar a favor de una información estable; los requisitos se especifican en forma de "pruebas de cliente" ejecutables, en lugar de documentación "estática" no ejecutable; por último la información (modelos, documentación, software), se almacena en un solo lugar y en un solo lugar, para evitar preguntas sobre cuál es la versión / información "correcta".

Agile Modeling requiere una participación de los interesados en el proyecto, involucrándose con el desarrollo del software. Esta práctica es una extensión de la práctica clientes insitu de Extreme Programming, que se verá más adelante. Además, a la hora de comenzar el proyecto, el equipo realiza una visualización de la arquitectura, en la que se realiza un modelo ligero y de alto nivel al principio del proyecto para explorar la estrategia de arquitectura que el equipo de desarrollo va a seguir. Para hacer esto es preferible la utilización de herramientas de modelado como pizarras o papel, con los que resulta fácil trabajar (herramientas inclusivas).

Un equipo ágil revisará su backlog una o más iteraciones por adelantado para asegurarse de que los ítems requeridos estén listo para ser desarrollado. Esto se conoce como modelado look-ahead, y también es conocido “backlog grooming" o "refinamiento de backlog" en Scrum. Cuando un ítem no ha sido suficientemente explorado en detalle a través este procedimiento, el equipo puede elegir hacer esa exploración durante su sesión de planificación de iteración/impresión. La necesidad de hacer esto es generalmente vista como un síntoma de que el equipo no está haciendo suficiente modelado de look-ahead.

Uno de los principales inconvenientes del Agile Modeling es que existe una gran dependencia de la comunicación personal y de la colaboración con el cliente, por lo que puede ser difícil

aplicar las disciplinas que conforman el Agile Modeling, especialmente en equipos grandes, cuando las habilidades de modelado son débiles o inexistentes, y cuando los miembros del equipo no colaboran lo suficiente.

3.2. Programación Extrema

Otro tipo de metodología Agile es lo que se conoce como Programación Extrema. La Programación Extrema permite una mejora de los proyectos de software de cuatro maneras esenciales: comunicación, simplicidad, retroalimentación, coraje o valentía, y respeto. Es decir, los desarrolladores se comunican constantemente con el cliente y con su equipo; mantienen sus diseños y códigos simples y limpios. La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente; al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real. Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante; muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario; el respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de producción.

3.3. Kanban

El método Kanban es uno de los más populares dentro del desarrollo Agile de proyectos. Kanban es una palabra de origen japonés que significa "tarjetas visuales". Esta técnica está dentro de la estrategia de mejora continua (estrategia Kaizen), y es utilizada para controlar la evolución y avance del trabajo dentro de una línea de desarrollo, gestionando como van completándose las tareas. Hay tres reglas principales para llevar a cabo esta tarea:

- Visualizar el trabajo y las fases del ciclo de formación o flujo de trabajo: El trabajo se divide en partes, normalmente cada una de esas partes se escribe en un post-it y se pega en una pizarra. Los post-it suelen tener información variada, si bien, aparte de la descripción, debieran tener la estimación de la duración de la tarea. La pizarra tiene tantas columnas como estados por los que puede pasar la tarea (ejemplo, en espera de ser desarrollada, en análisis, en diseño, etc.). El objetivo de esta visualización es que quede claro el trabajo a realizar, en qué está trabajando cada persona, que todo el mundo tenga algo que hacer y el tener clara la prioridad de las tareas.
- Determinar el límite de Trabajo en Curso: Es importante que el número de tareas máximo que se puede realizar en cada fase sea conocido. Ese número de tareas se es conocido como límite del “work in progress”. A esto se añade otra idea tan razonable como que para empezar con una nueva tarea alguna otra tarea previa debe haber finalizado. Es decir, la idea es centrarse en cerrar tareas y no en comenzarlas.
- Medición del tiempo en completar una tarea: El tiempo que se tarda en finalizar una tarea se conoce como "lead time". Este periodo cuenta desde que se hace la petición hasta que se hace la entrega.

3.4. Desarrollo de Software Adaptativo

Después tenemos el llamado Desarrollo de Software Adaptativo, el modelo ASD (Adaptative Software Development), que fue desarrollada por Jim Highsmith y Sam Bayer a comienzos de 1990. Se basa en el principio de adaptación continua al proceso para desarrollar software y sistemas muy complejos, centrándose en la colaboración humana y en la organización del equipo. Se divide en tres fases:

La fase de especulación es donde se llevan a cabo los documentos de especificación del proyecto, se inicia el proyecto y se planifica el ciclo adaptativo utilizando información del inicio de proyecto, como la meta del cliente, las restricciones y los requisitos básicos para definir el conjunto de ciclos de lanzamiento. Se realizan estimaciones teniendo en cuenta los posibles desvíos y se decide el número de iteraciones necesarias para completar el proyecto, centrándose especialmente en las características que pueden ser utilizadas por el cliente al final de la iteración, por lo que es necesario marcar objetivos prioritarios dentro de las mismas iteraciones.

La fase de colaboración es donde se lleva a cabo la mayor parte del desarrollo, manteniendo una estructura de trabajo cíclica. Es importante que exista una coordinación que permita a los equipos transmitir los conocimientos adquiridos al resto, de manera que no tenga que ser aprendido por los otros equipos.

Por último, la fase de aprendizaje consiste en una serie de ciclos colaborativos entre los equipos, poniendo sobre la mesa lo que se ha aprendido, tanto positivo como negativo. Es un elemento fundamental para que un equipo resulte eficaz.

Jim Highsmith identifica cuatro tipos de aprendizaje en esta etapa:

- Calidad del producto desde un punto de vista del cliente. Es la única medida legítima de éxito.
- Calidad del producto desde un punto de vista de los desarrolladores. Es una evaluación de la calidad de los productos desde un punto de vista técnico.
- La gestión del rendimiento. Evaluación de lo aprendido mediante el empleo de los procesos utilizados por el equipo.
- Situación del proyecto. Punto de partida para la construcción de la siguiente serie de características.

3.5. FDD

Por último, tenemos la metodología FDD (Feature-Driven Development). Esta metodología está orientada a equipos de mayor número de efectivos que el resto de las metodologías Agile. FDD es una metodología dirigida por modelos, y de iteraciones cortas. Además, al contrario que otras metodologías como Scrum, en FDD existe la figura de Jefe de Proyecto y existe una fase de arquitectura. Esta metodología está definida por 5 procesos: Desarrollo del modelo global; Construir una lista de características; Planificar; Diseñar; y Construir. Las tres primeras fases constituyen los llamados "procesos iniciales" que serían lo equivalente en otras metodologías a la iteración 0.

Desarrollo del modelo global: Es la actividad inicial del proyecto, que incluye a los miembros de gestión y a los miembros de desarrollo del equipo bajo la guía de un experto modelador de objetos en el papel de Arquitecto Jefe, en la que se estudia el alcance y requisitos del proyecto. Después, se forman pequeños equipos con una mezcla de personal de gestión y desarrollo que luego componen sus propios modelos y los presentan al resto del equipo para su posterior revisión y discusión. Una de las propuestas o una fusión de los modelos, se selecciona por consenso, convirtiéndose así en el modelo que se empleará.

Construir una lista de funcionalidades: Es la actividad inicial a nivel de todo el proyecto para identificar todas las características que se necesitan para cumplir con los requisitos. Un equipo formado por los programadores jefes de la fase 1 es creado para descomponer el dominio del proyecto en áreas especializadas, en las actividades de negocio dentro de ellas y en los pasos a seguir para cada actividad, dando forma de esta manera a la lista categorizada de requisitos del proyecto.

Planificar por funcionalidad: El jefe de proyecto, el jefe de desarrollo y los programadores jefe planifican el orden en el que se van a realizar las características,

dependiendo de la complejidad de las funcionalidades que deben implementarse, de la dependencia entre funcionalidades y de su prioridad para implantarse.

Diseñar por funcionalidad: Se fijan una serie de funcionalidades para su desarrollo asignándolas a un programador jefe. El Programador Jefe forma entonces un equipo de funcionalidades identificando a los desarrolladores que pueden ser más adecuados para participar en el desarrollo de las funcionalidades seleccionadas. Este equipo produce entonces el Diagrama de secuencia para las funcionalidades asignadas.

Construir por funcionalidad: Los desarrolladores implementan los elementos necesarios para el diseño de esta característica. El código desarrollado es entonces probado por unidad y el código inspeccionado (el orden de que es determinado por el Programador Jefe). Después de una inspección de código exitosa, el código es elevado a la categoría de producción.

4. Metodología Agile vs Metodología en Cascada

En este apartado se va a comparar la metodología Agile con la metodología de desarrollo de proyectos tradicional o en cascada

4.1. Metodología en cascada

Antes de iniciar la comparativa entre las metodologías Agile y en Cascada, se hará una breve explicación sobre qué es y en que consiste la metodología en cascada.

El modelo clásico de cascada divide el ciclo de vida en un conjunto de fases. Este modelo considera que una fase puede iniciarse una vez finalizada la fase anterior. Es decir, la salida de una fase será la entrada a la siguiente fase. Por lo tanto, el proceso de desarrollo puede considerarse como un flujo secuencial en la cascada, de manera que las fases no se solapan entre sí.

Si bien su sencillez de aplicación es uno de sus puntos fuertes, tras numerosos estudios ha resultado evidente que resulta menos preciso que las metodologías Agile. Estos son algunos de sus atributos:

- Es un modelo estructurado, en el cuál, su rigidez supone en ocasiones una limitación para llevarlo a la práctica.
- En lugar de dividir el proyecto en unidades más sencillas de controlar para dinamizar su gestión, desarrolla el proyecto como un todo.
- Consiste en un planteamiento secuencial, donde la simultaneidad no tiene cabida, lo que se pierde el valor que algunas colaboraciones puedan aportar.

- No está preparada para el cambio, lo que supone un grave impedimento teniendo en cuenta que un proyecto es un ente cambiante a todos los niveles: fases, objetivos y condiciones.

El modelo de desarrollo en cascada sigue una serie de etapas o fases de forma sucesiva. Las fases que componen el modelo son las siguientes:

Estudio de viabilidad: En esta fase se determina si resulta factible, tanto a nivel económico como técnico, desarrollar el software. El estudio de viabilidad implica el conocimiento del problema y el planteamiento de las soluciones a ese problema. A continuación, las soluciones halladas se analizan en función de sus beneficios e inconvenientes. Finalmente se toma una decisión y se elije la solución más adecuada, para la cual se desarrollarán el resto de las fases.

Análisis y especificación de requisitos: El objetivo de esta fase es entender exactamente las necesidades del usuario y documentar estos deseos apropiadamente. Esta fase puede ser dividida en dos subfases:

- Recopilación y análisis de requisitos: Se recogen todos los requisitos relativos al software del cliente para su posterior análisis. El objetivo de este análisis es eliminar los requisitos incompletos, es decir, aquellos en los que se han omitido partes de los requisitos existentes, y los requisitos incoherentes o contradictorios.
- Especificación de requisitos: estos requisitos analizados se documentan en una memoria llamada SRD (documento de especificación de requisitos), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos. Además, sirve como contrato entre el equipo de desarrollo y los clientes, de manera que en caso de futuras disputas entre cliente y desarrolladores podrá ser resuelta examinando este documento.

Diseño: El objetivo de la fase de diseño es transformar los requisitos especificados en el documento SRD en una estructura que sea adecuada para su implementación en algún lenguaje de programación. Como resultado surge el SDD (Documento de Diseño del Software), que es una descripción escrita de un producto de software, que un diseñador de software escribe con el fin de dar un equipo de desarrollo de software orientación general para la arquitectura de un proyecto de software. Es conveniente distinguir entre diseño arquitectónico y diseño detallado. El diseño arquitectónico define como se va a dar forma al proyecto y la estructura que este tendrá, mientras que el diseño detallado define los algoritmos y organización del código.

Codificación y Test Unitarios: En la fase de codificación, el diseño del software se traduce al código fuente utilizando cualquier lenguaje de programación adecuado (java, .net, C++, etc.). Por lo tanto, cada módulo diseñado está codificado. El objetivo de la fase de pruebas unitarias es comprobar si cada módulo funciona correctamente o no.

Integración y pruebas del sistema: La integración de los diferentes módulos es llevada a cabo después de ser codificado y de pasar los test unitarios. La integración de los

diferentes módulos se lleva a cabo de forma gradual a lo largo de una serie de pasos. Durante cada paso se añaden al sistema módulos previamente diseñados y se prueba el sistema resultante. Finalmente, una vez que todos los módulos han sido integrados y testeados con éxito, se tiene el sistema completo y se llevan a cabo las pruebas del sistema:

- α-testing: es la prueba del sistema realizada por el equipo de desarrollo.
- β-testing: es la prueba del sistema realizada por un grupo de clientes amistosos.
- Pruebas de aceptación: Después de la entrega del software, el cliente realizará la prueba de aceptación para determinar si acepta el software entregado o lo rechaza.

Mantenimiento: El mantenimiento es la fase más importante del ciclo de vida de un software. El esfuerzo invertido en mantenimiento es el 60% del esfuerzo total invertido en el desarrollo de un software. Existen tres tipos de mantenimiento:

- Correctivo: Este tipo de mantenimiento se centra en corregir errores que se hayan podo dar durante la fase de desarrollo.
- Perfeccionamiento: Se utiliza para mejorar la funcionalidad del sistema de acuerdo a los deseos del cliente.
- Adaptativo: Este tipo de mantenimiento es necesario para que el software funcione en un nuevo entorno. La tecnología está en constante evolución, por lo que el software diseñado tiene que adaptarse a nuevas versiones y tecnologías para evitar que se quede obsoleto.

Las principales ventajas de utilizar la metodología en cascada son: es fácil de entender e implementar debido a su fuerte estructuración; la documentación es muy exhaustiva; el tiempo empleado durante el diseño del producto puede evitar problemas que a la larga serían más costosos; es ideal para proyectos estables y con pocos cambios.

Por otra parte, el modelo clásico de cascada tiene varios defectos. A continuación se presentan algunos de los principales inconvenientes de este modelo: si se trata de un proyecto susceptible a cambios, muchos requisitos cambiaran, lo que supondrá volver a realizar fases previamente hechas; no permite mostrar al cliente los avances del producto a medida que se va desarrollando; si no se han tenido en cuenta posibles dificultades a la hora de desarrollar el software, conllevara un rediseño del proyecto; los proyectos a largo plazo pueden suponer graves problemas, ya que las necesidades pueden cambiar; no hay superposición de fases: Este modelo recomienda que la nueva fase sólo pueda comenzar después de la finalización de la fase anterior. Pero en los proyectos reales, esto no se puede mantener. Para aumentar la eficiencia y reducir el coste, las fases pueden solaparse.

Teniendo en cuenta los pros y contras de utilizar la metodología en cascada anteriormente citados, se llega a la conclusión de que su aplicación debería aplicarse exclusivamente a proyectos en los que: los requisitos estén muy claros y sean fijos; la definición del producto sea estable; no existan dudas sobre la tecnología utilizada; no

existan ambigüedades; y se cuente con un equipo cualificado y con experiencia en este tipo de proyectos.

4.2. Razones para escoger Agile

El hecho de introducir Agile en una organización en la gestión de desarrollo de proyectos viene dado por un cambio de prioridades. Usando Agile se priorizan las características de mayor importancia primero, gracias a su carácter iterativo. Es decir, cuando se itera y se prioriza el trabajo que hay que hacer en cada iteración significa que todo el trabajo es de alto valor y el trabajo de mayor prioridad será realizado primero.

También es fundamental la interacción con el cliente, de manera que los equipos Agile pongan el software desarrollado en manos de sus clientes con bastante rapidez y construyendo lo suficiente para satisfacer sus necesidades.

Además, el uso de Agile ayuda a mantener la calidad del producto, asegurando que los equipos sean responsables de mantener un alto nivel de calidad y también verificando que todo funciona y se integra correctamente.

Ayuda a mantener la competitividad, ya que permite encontrar los fallos y problemas en el proceso con más facilidad y ayudar a los equipos a solucionarlos permitiéndoles concentrarse en problemas que tal vez los estén retrasando o haciendo que la entrega sea poco confiable, lo que supone una mejora en la competitividad tanto del equipo como del proceso.

Ayuda a sobreponerse de la pérdida de un empleado estrella, un desarrollador de crack, un gran gerente de proyectos, etc. La forma en que Agile lucha contra estos inconvenientes es a través de equipos cambiantes, nuevas incorporaciones, movimiento entre proyectos de los empleados. Otra forma de lidiar con estos problemas es la programación de pares, donde los desarrolladores se emparejan entre sí y trabajan en una computadora similar o trabajan de forma remota a través de terminales, lo que ayuda a distribuir el conocimiento y el “know-how” a través de la organización, de modo que en el caso de que uno de sus desarrolladores estrella se vaya, el conocimiento se haya transferido entre los otros miembros.

Ayuda en el control de costes. La idea de un sistema iterativo con escalas de tiempo fijas pero con necesidades cambiantes ayudan a poder adaptar a un presupuesto fijo un alcance del producto y de sus característica variable. Esto ayuda a mantener los costes bajo control en lugar de que sufra grandes incrementos.

Ayuda a la gestión de riesgos. Dado que se están realizando pequeñas versiones incrementales y que el cliente y el equipo de desarrollo trabajan juntos para poder identificar cualquier problema en una fase temprana, es más fácil responder a los cambios y amenazas, por lo que se reduce el riesgo de que se desarrolle cualquier cosa que no se ajustase a las necesidades o propósitos de la aplicación u organización.

4.3. Ventajas Agile frente a Cascada

Si bien es cierto que para desarrollar un proyecto según una metodología u otra influyen numerosos factores, en general, la metodología Agile resulta en la mayoría de los casos más ventajosa que la metodología en cascada. Esto es debido en primer lugar a que es difícil introducir cambios usando el modelo de cascada. La documentación se hace desde el principio y el proyecto avanza a partir de ahí, por lo que hacer cambio resulta problemático, al haberse establecido todo al principio. Otro inconveniente es que el software se produce al final del ciclo. Al ser un proceso por etapas o "paso a paso", pasa tiempo hasta que cualquier trabajo de software real se produce. También hay un gran riesgo e incertidumbre porque es imposible predecir el riesgo en una línea de tiempo larga, de modo que se genera incertidumbre. Tampoco es ideal para proyectos de larga duración, porque a medida que el proyecto crece con el tiempo, ese riesgo e incertidumbre crece y los requisitos pueden cambiar. Y tampoco es adecuado para proyectos complejos orientados a objetos donde hay mucha interacción entre los elementos que pueden necesitar ser revisados a medida que el proyecto avanza.

En lo referente al trato con el cliente, con Agile se prefiere que el cliente esté disponible durante todo el proyecto para discusiones, consultas y supervisión del trabajo realizado, mientras que en las cascadas se involucran en los objetivos del producto, pero no intervienen durante el desarrollo.

En cuanto a cambios en el alcance del proyecto, Agile acepta bien los cambios que puedan presentarse, ya que al desarrollar el proyecto por incrementos cada cierto periodo de tiempo y definir las tareas para ese periodo, resulta mucho más sencillo realizar cualquier modificación, mientras que la cascada funciona bien cuando el alcance se conoce de antemano o cuando los términos del contrato limitan los cambios, es decir, el alcance está muy fijado en la cascada.

Otro aspecto a destacar sería la priorización de características. En Agile, la prioridad está generalmente organizada según cuáles son las características más valiosas, de forma que estas se implementan primero, reduciendo el riesgo de que el producto que se produce sea inutilizable. O si, por ejemplo, los fondos se acaban y no queda dinero, al menos se han priorizado las características para que las más importantes se hagan primero. En cascada, básicamente se hace todo según lo establecido al inicio del proyecto, obteniéndose todo o nada.

En términos de equipos, Agile prefiere equipos más pequeños y dedicados que estén altamente coordinados y sincronizados. Mientras que, en cascada, hay coordinación de equipos, pero la sincronización entre ellos se limita a ciertos puntos de entrega en lugar de una sincronización constante.

En lo relativo a la financiación Agile funciona muy bien con contratos por tiempo y materiales u otro tipo de financiación no fija, mientras que la metodología en cascada reduce el riesgo en los contratos de precio fijo al obtener un acuerdo por adelantado en términos de lo que se debe entregar.

4.4. Problemas de transición a Agile.

A la hora de hacer la transición de la metodología en cascada a Agile, los equipos de desarrollo se van a enfrentar a una serie de problemas y desafíos para una implementación satisfactoria.

En primer lugar, hay que tener en cuenta los retos de gestión y organización, ya que, en una transición a Agile, todos los aspectos de la organización van a cambiar. Los desafíos que supondrá esto no serán solo a nivel técnico o de problemas de aprendizaje o de formación, sino que también interviene la mentalidad de los trabajadores, ya que un cambio en la metodología de trabajo supone un giro radical a la forma habitual de trabajar, lo que puede suponer para algunos trabajadores una barrera complicada de superar y olvidar su papel anterior. El método tradicional o en cascada requería mucha documentación e informes rígidos, mientras que Agile es más flexible y dinámico, lo que puede resultar problemático a la hora de que trabajadores habituados a manejar documentación pesada y proyectos poco flexibles se adapten. Otro aspecto que puede resultar confuso es el ciclo de vida iterativo de los proyectos Agile, ya que este puede causar conflictos para las personas que tienen dificultades para entender cómo pasar del ciclo de vida tradicional a uno iterativo e incremental. Por ejemplo, un Project Manager debe ser consciente de que su rol de "comandante" ya no va a existir. No hay Project Managers que actúen como jefes o gestores del trabajo y el equipo. La gente está acostumbrada a una forma jerárquica de trabajar, pero dentro de la metodología Agile se trabaja con una estructura plana donde todos trabajan juntos un mismo objetivo, sin hay posiciones intermedias, ni ascensos verticales, ni nadie indicando a nadie cómo debe trabajar, porque todo el mundo es responsable de su trabajo y conoce su función en el equipo. La responsabilidad de la gestión del proyecto se distribuye entre los distintos roles, es decir, no hay una gestión centralizada en una figura de Project Manager, lo que puede suponer en ocasiones un inconveniente, ya que el equipo de desarrollo no está acostumbrado a ser responsable.

Por otra parte, está el dilema de las estimaciones frente a los plazos. Es común que existan malentendidos en este aspecto. Las estimaciones y los plazos no son la misma cosa, y eso puede ser difícil de asimilar para la gente que vienen de un escenario de plazos en el que se exige un determinado trabajo para una fecha límite, en lugar de crear estimaciones. En Agile, es el propio equipo el que asigna una estimación del tiempo que considera que va a suponer llevar a cabo una determinada tarea, lo que puede resultar complejo para alguien habituado a que le exijan un plazo fijo de entrega.

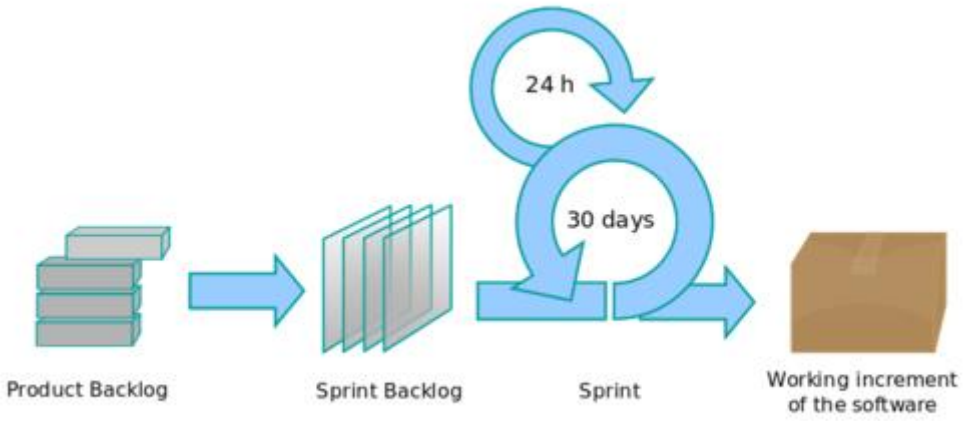
Otro aspecto en el que pueden surgir problemas es a la hora de crear un equipo multifuncional. La gente está acostumbrada a trabajar con su equipo funcional dentro de un departamento específico, con Agile se busca a un grupo de individuos cuyo conjunto de conocimientos integrados engloba todas las habilidades para desarrollar con éxito el proyecto.

5. Ejemplo Scrum: Desarrollo de software para una empresa de mensajería.

Una vez han sido examinados las características de la metodología Agile Scrum, se procederá a desarrollar un ejemplo práctico de su desarrollo y funcionamiento. Este ejemplo consistirá en el desarrollo de un proyecto de desarrollo de software aplicado a una empresa de mensajería y en el se describirán las diferentes fases y entregas del proyecto.

5.1. Contexto

Una nueva empresa de mensajería desea desarrollar un sistema de software para gestionar todos los aspectos relacionados con el negocio (clientes, finanzas, ventas, etc.). Para ello decide contratar los servicios de una consultora tecnológica que se encargue de diseñar el software adecuado para ellos. Se ha decido desarrollar el proyecto con la metodología Agile Scrum, ya que es la metodología más en auge en los últimos años a la hora de gestionar proyectos de desarrollo de software, debido a su flexibilidad al cambio, a la interacción continua con el cliente, a la implicación de todos los interesados y a las entregas iterativas o incrementales, que permiten un seguimiento continuo del avance del proyecto.



Ciclo Scrum

La implementación de Scrum se dividirá en dos fases: La fase de análisis, donde se definirán los objetivos que deben ser alcanzados al final del proyecto, se definen las responsabilidades del equipo Scrum, se escoge la tecnología más adecuada para el

desarrollo del proyecto y se define la funcionalidad del sistema que va a ser desarrollado, mediante la definición de las historias de usuario.

5.2. Análisis del proyecto

5.2.1. Estructura y definicón del proyecto

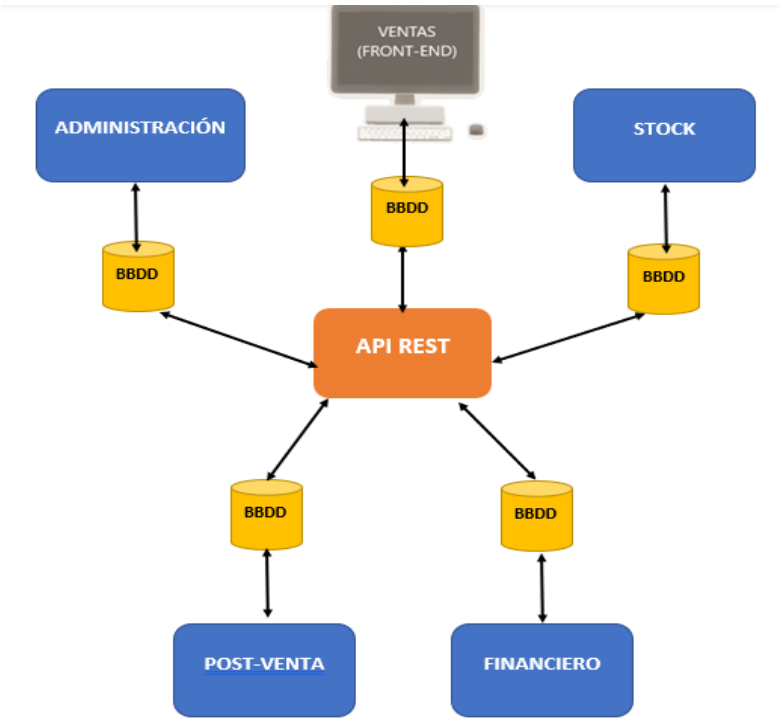
El proyecto se iniciará con una reunión entre el Project Manager que se responsabilizará de este proyecto y un arquitecto de software, que será el que diseñará la estructura del proyecto, por parte de la consultora tecnológica; y los representantes de los clientes, es decir, la empresa de mensajería.

Tras esta reunión, se ha establecido la estructura que tendrá el proyecto. El proyecto constará de varios módulos, con el fin de simplificar el trabajo y de trabajar en equipos reducidos (uno de los requisitos Scrum). Así pues, el proyecto estará dividido en los siguientes módulos.

- Ventas: Será la parte "front-end" de la aplicación, es decir, es la parte visible para los usuarios. Consistirá en una página web a la que los usuarios acceden para ver el catálogo de productos y efectuar cualquiera de las diferentes operaciones que se ofrecen: compra, reclamaciones, devoluciones, seguimiento de paquetes, etc. Además, podrán tener acceso a su historial de datos, buscar artículos según sus preferencias o función, recibir ofertas y sugerencias de artículos que concuerden con su perfil e historial de compras, guardar artículos deseados en una lista para futuras compras, almacenar puntos de compra para futuros descuentos y regalos. La información de la página web se irá actualizando continuamente a través de la conexión con Api REST, que se explicará a continuación.
- API REST: Será el centro neurálgico de la aplicación. Se trata de un protocolo de intercambio y manipulación de datos en cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar). Twitter, YouTube, los sistemas de identificación con Facebook… hay cientos de empresas que generan negocio gracias a REST y las API's REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de API's para

servicios de Internet. La API será la encargada de conectar toda la información entre las distintas aplicaciones del sistema, a través de bases de datos que se actualizan de forma bidireccional cada vez que se produce cualquier cambio.

- Stock: Una buena gestión de stock es fundamental para cualquier empresa, ya que regula el flujo entre las entradas y las salidas de existencias de los productos de una empresa. La forma de regular el flujo de entrada es variando la frecuencia y el tamaño de los pedidos que se realicen a los proveedores. El control sobre el flujo de salida es mucho menor pues las condiciones son impuestas por los consumidores. El módulo de stock será el que permitirá conocer que cantidad de productos tiene la empresa a su disposición. Lo ideal sería que el flujo de entrada fuese igual al de salida, pero esto no es materialmente posible, pues es necesario un tiempo para responder adecuadamente. Por lo tanto se ha de intentar que el nivel de existencias sea mínimo, sin que se produzcan rupturas en la salida. Esta información se irá actualizando a medida que los productos se vayan vendiendo y a medida que la empresa adquiera nueva mercancía, gracias a la conexión de esta aplicación con la API que, al estar también conectada con el módulo de ventas, permitirá que el flujo de información de adquisiciones y ventas este siempre al día.
- Postventa: El módulo de postventa será el encargado de continuar ofreciendo un servicio a los clientes después de hacer una compra. Tener un sistema de servicio de postventa es fundamental para toda compañía de ventas online ya que una correcta gestión de la postventa ayuda a fidelizar clientes y sirve como reclamo para conseguir nuevos, es decir, es una fuente de ingresos más. Aquí se incluirán todas las actividades que se hagan después de la venta. Algunas de estas actividades que es importante atender después de la compra son: promociones y descuentos para futuras compras; acumulación de puntos de compra, seguimiento de pedidos, para saber en qué proceso del envío se encuentra el paquete comprado; gestión de cambios y devoluciones; garantías y soporte para el mantenimiento del producto. El módulo de postventa actualizará su información sobre los productos vendidos y los usuarios a los que se ha vendido gracias a su conexión a la API, que a su vez está unida al resto de módulos y se encargará de actualizar continuamente el flujo de información.
- Financiero: El modulo financiero será el encargado de gestionar todos los aspectos económicos de la empresa, conectando a través de la API y de las bases de datos las transacciones económicas que se lleven a cabo, los recursos financieros de la empresa, correspondientes tanto al activo como el pasivo de la organización, los datos bancarios y la contabilidad.
- Administración: El módulo de administración conectará a través de la API y de las bases de datos con el resto de los módulos para gestionar y organizar el funcionamiento de todas las secciones de la empresa, por ello este modo deberá recoger datos de todos los demás módulos y almacenarlos adecuada y eficientemente. A través de este modulo también se gestionará toda la parte del transporte de la mercancía a sus destinatarios, que se realizará a través de compañías de reparto externas.



Módulos del proyecto

Cada uno de estos módulos serán mini proyectos desarrollados por diferentes equipos Scrum. Cada equipo Scrum estará formado por los siguientes individuos:

- Product Owner: Su misión será comunicar la visión del producto al equipo de desarrollo, representando los intereses del cliente. Es el miembro del equipo con más autoridad y con más responsabilidad, ya que será el que afrontará las consecuencias si el proyecto no sale como estaba previsto.
- Scrum Master: Es el encargado de interactuar entre Product Owner y equipo de desarrollo. Es el que se encargará de gestionar el trabajo del equipo de manera que este alcance el máximo nivel de productividad y ayudará a eliminar progresiva y constantemente impedimentos que van surgiendo en la organización y que pueden afectar a las entregas de producto. El Scrum Master debe ser el responsable de velar porque Scrum se lleve adelante.
- Equipo de desarrollo: es el grupo de individuos encargado de construir el proyecto gracias a sus conocimientos técnicos, autoorganizándose y autogestionándose para llevar a cabo las tareas establecidas en cada Sprint.

En este trabajo se va a desarrollar el proceso Scrum de uno de los módulos, ya que realizar el proceso de desarrollo de cada uno de los módulos resultaría repetitivo y extenso en demasía, por ello se estudiará el caso del módulo de ventas, puesto que este módulo es el más sencillo desde un punto de vista técnico, y este trabajo pretende

mostrar la parte del proceso de gestión de proyectos con la metodología Agile Scrum y no la parte de desarrollo de software.

Para el desarrollo del módulo de ventas el equipo Scrum estará formado por el Scrum Master, el Product Owner y un equipo de desarrollo de 2 personas. En este caso, al tratarse de un equipo pequeño, ya que la aplicación no es de gran envergadura el Scrum Master y el Product Owner no solo se encargarán de este equipo, sino que, como es común en estos casos, tendrán varios equipos de diferentes aplicaciones a su cargo. Una vez que el equipo Scrum está consolidado se procederá a identificar los objetivos que deben ser cumplidos para alcanzar el éxito del proyecto. Estos objetivos se definirán a partir de una reunión entre equipo Scrum y cliente. En esta reunión el cliente explicará al equipo Scrum que es exactamente lo que quiere a nivel funcional y las necesidades que debe cumplir, y el equipo de desarrollo le hace sugerencias y aporta un diferente punto de vista al cliente. Una vez que los objetivos están definidos, el Product Owner escribirá las Historias de Usuario, donde se describe la funcionalidad que incorporará el sistema y se representa formalmente su definición. Las historias de usuario se descomponen en tareas, quedando formado el Product Backlog, que será la base del proyecto a desarrollar. Estas tareas son unidades de trabajo que tienen asignado un esfuerzo estimado y un estado. Por lo que es en las tareas en los que se basa la estimación de esfuerzos general del proyecto.

5.2.2. Historias de Usuario

Las historias de usuario se definirán utilizando el siguiente modelo:

Historia de Usuario	
Id	
Nombre	
Prioridad	
Riesgo	
Descripción	
Validación	

- Id: Identificador único asignado a este elemento del proyecto, con formato US XX (siendo XX numeración progresiva de 01 a 99).
- Nombre de Historia: Nombre corto utilizado para describir muy brevemente la historia de usuario.
- Prioridad en Negocio: Preferencia de cara al desarrollo de la historia de usuario respecto a las demás. Toma valores: Alta, media y baja.

- Riesgo en Desarrollo: Se trata de la importancia de la historia de usuario con relación al conjunto del proyecto. Cuantificando de este modo el daño provocado en caso de fallo. Toma valores: Alto, medio y bajo
- Descripción: Breve explicación de las intenciones de la historia de usuario. Debe dejar clara la idea de la propia historia.
- Validación: Los elementos funcionales que se pretenden alcanzar llevando a cabo esta Historia de Usuario.

Las historias de usuario del desarrollo del módulo de ventas serán las que siguen:

Historia de Usuario	
Id	US-001
Nombre	Creación de la aplicación web
Prioridad	Alta
Riesgo	Alto
Descripción	Desarrollo de la interfaz en la que se desarrollará el front-end, con un aspecto simple y de manejo sencillo
Validación	• La página web base ha sido creada. • Entorno de ejecución E/S, asíncrona y dinámica. • La aplicación funciona en diferentes navegadores (Chrome, Explorer, Firefox…)

Historia de Usuario	
Id	US-002
Nombre	Gestión de publicidad
Prioridad	Media
Riesgo	Bajo
Descripción	Incorporación a la interfaz de banners y anuncios publicitarios
Validación	• Aparición de banner al realizar una compra • Anuncios laterales • Anuncios relacionados con criterios de búsqueda

Historia de Usuario	
Id	US-003
Nombre	Creación de cuentas de usuario
Prioridad	Alta
Riesgo	Medio
Descripción	Formulario para darse de alta, definición y gestión de los campos a rellenar
Validación	• El formulario tiene todos los campos establecidos (distinguiendo entre obligatorios y opcionales) • Si no se rellenan todos aparece un mensaje de error solicitando rellenar los campos restantes. • Enviar mensaje de confirmación y de activación de la cuenta. • Cada usuario tenga acceso único y registrado

Historia de Usuario	
Id	US-004
Nombre	Resumen de la compra
Prioridad	Baja
Riesgo	Bajo
Descripción	El comprador recibe la información concerniente a la compra que ha realizado.
Validación	• Tras hacer una compra aparece un pop-up con la información de la compra. • Se recibe un correo con la información de la compra.

Historia de Usuario	
Id	US-005
Nombre	Gestión de pedidos
Prioridad	Alta
Riesgo	Alto
Descripción	El procedimiento para realizar un pedido.
Validación	Una vez el ítem o los ítems deseados han sido seleccionados, el sistema: • “Cesta de la compra” • Confirma que el usuario es el correcto. • Pide la dirección en la que se desea que se entregue el pedido. • Pide los datos de pago. • Lista de deseos.

Historia de Usuario	
Id	US-006
Nombre	Editar pedidos
Prioridad	Alta
Riesgo	Alto
Descripción	Modificar diferentes aspectos de un pedido.
Validación	Aspectos que pueden ser modificados: • Fecha de entrega. • Lugar de entrega. • Características del producto. • Cancelar pedido.

Historia de Usuario	
Id	US-007
Nombre	Seguridad
Prioridad	Alta
Riesgo	Alto
Descripción	Diseñar la seguridad de la aplicación de acuerdo a lo establecido en el artículo 9 de la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos y Garantía de los Derechos Digitales
Validación	• Encriptación de información. • Uso de sesiones de usuario.

Historia de Usuario	
Id	US-008
Nombre	Gestionar cuentas de usuario
Prioridad	Media
Riesgo	Medio
Descripción	Cambiar los diferentes campos de la información de un usuario.
Validación	• Cualquiera de los datos de un usuario podrá ser modificado. • El usuario podrá ser dado de baja.

Historia de Usuario	
Id	US-009
Nombre	Seguimiento de paquetes.
Prioridad	Baja
Riesgo	Bajo
Descripción	El usuario podrá localizar donde está su paquete en cada momento.
Validación	• Facilitar la ubicación del paquete. • Facilitar la información sobre la empresa de transporte encargada de realizar el reparto. • Estimación de la hora y el día de entrega.

Historia de Usuario	
Id	US-010
Nombre	Gestión de base de datos.
Prioridad	Alta
Riesgo	Alto
Descripción	Utilización de una base de datos para almacenar y actualizar la diferente información necesaria para el funcionamiento adecuado de la aplicación.

Validación	• Introducción, modificación y eliminación de datos de:
	  ○ Clientes.
	  ○ Pedidos.
	  ○ Productos.
	• Actualizado con la API.

Historia de Usuario	
Id	US-011
Nombre	Gestión de productos.
Prioridad	Alta
Riesgo	Medio
Descripción	Los productos a la venta podrán variar, eliminando o modificando algunos e introduciendo otros.
Validación	• Introducción de nuevos productos a la lista. • Eliminación de productos que ya no se venden. • Modificación de características de venta de un producto. • Organizar por secciones. • Actualización con la base de datos.

Historia de Usuario	
Id	US-012
Nombre	Preferencias y sugerencias.
Prioridad	Baja
Riesgo	Bajo
Descripción	La aplicación sugerirá productos que se ajusten a búsquedas o pedidos realizados con anterioridad por el usuario.
Validación	El usuario encontrará una sección de preferencias en la cual le aparecen productos relacionados con su historial de búsquedas.

5.2.3. Tareas

Una vez han sido definidas las Historias de Usuario, el equipo Scrum elige cuales de ellas va a desarrollar en cada Sprint. El equipo no solo debe planificar lo que va a hacer cada Sprint, sino que debe desarrollar un plan detallado sobre como va a conseguirlo. Para ello, normalmente el equipo divide cada Historia de Usuario en tareas. Sin embargo, este procedimiento es complejo y requiere de experiencia, conocimiento y decisión.

La división de una Historia de Usuario va a depender de muchos factores: el tamaño de la Historia de Usuario, de la complejidad del producto, de la experiencia del equipo Scrum, etc. Tampoco hay un procedimiento específico para realizar la partición en tareas, sino que es algo que se aprende con la experiencia.

Lo ideal es que es que las tareas sean pequeñas (entre 0.5 y 2 días), facilitando así el seguimiento y la estimación por parte del equipo. Además, siempre que sea posible, las tareas deberían poder probarse de forma individual, es decir, no depender de que otra tarea esté finalizada para poder probarlo.

No todas las tareas son técnicas, sino que existen muchas tareas adicionales que deben cumplirse para que una Historia de Usuario se dé por finalizada. Para identificar este tipo de tareas se utilizará una check-list, que contiene tareas típicas dentro de todas o casi todas las historias de usuario de todos los equipos. Esta check-list se compone de las siguientes tareas complementarias:

- Interfaces y métodos necesarios.
- Unit testing, de integración, pruebas de aceptación (automáticas o no), pruebas exploratorias, tests de regresión.
- Requisitos no funcionales: Rendimiento, escalabilidad, mantenibilidad, seguridad…
- Revisiones de código.
- Refactors/Rebuilds necesarios + pruebas asociadas necesarias.
- Corrección de errores.
- Reparación de incidencias del análisis de código estático.
- Preparación de la demo (datos de prueba, scripts…).

- Actualizar la documentación (técnica, javadoc, de usuario…).

Con todo lo anterior, las Historias de Usuario establecidas se van a dividir en las siguientes tareas:

Tarea	
Id	**T-001**
US	US-001
Puntos estimados	4
Descripción	Creación y configuración de la aplicación web.

Tarea	
Id	**T-002**
US	US-001
Puntos estimados	4
Descripción	Configuración de llamada a la API a través de la base de datos.

Tarea	
Id	**T-003**
US	US-001
Puntos estimados	2
Descripción	Configuración de renderización de vistas.

Tarea	
Id	**T-004**
US	US-001
Puntos estimados	3
Descripción	Configuración de la estructura front end.

Tarea	
Id	**T-005**
US	US-002
Puntos estimados	0.5
Descripción	Configuración de banners HTML 5.

Tarea	
Id	**T-006**
US	US-002
Puntos estimados	0.5
Descripción	Configuración de publicidad lateral.

Tarea	
Id	**T-007**
US	US-002
Puntos estimados	1.5
Descripción	Configuración de publicidad selectiva.

Tarea	
Id	**T-008**
US	US-003
Puntos estimados	0.5
Descripción	Configuración de formulario para darse de alta.

Tarea	
Id	**T-009**
US	US-003
Puntos estimados	0.5
Descripción	Configuración de respuesta vía e-Mail con la información de usuario.

Tarea	
Id	**T-010**
US	US-003
Puntos estimados	2.5
Descripción	Sincronización con la base de datos.

Tarea	
Id	**T-011**
US	US-004
Puntos estimados	0.5
Descripción	Configuración de respuesta vía e-Mail con la información de la compra.

Tarea	
Id	**T-012**
US	US-004
Puntos estimados	0.5
Descripción	Configuración del pop-up con la información de la compra.

Tarea	
Id	**T-013**
US	US-005
Puntos estimados	1
Descripción	Configuración de la sección “Cesta de la compra”, donde se almacenan los ítems que desean adquirirse, para después proceder a la compra.

Tarea	
Id	**T-014**
US	US-005
Puntos estimados	1.5
Descripción	Configuración de confirmaciones de compra (Autentificación de usuario, dirección de entrega y método de pago)

Tarea	
Id	**T-015**
US	US-005
Puntos estimados	1
Descripción	Configuración de la sección “Lista de deseos”, donde se van a guardar los objetos que gusten al usuario o que deseé comprar en un futuro.

Tarea	
Id	**T-016**
US	US-005
Puntos estimados	2.5
Descripción	Configuración de la conexión a la base de datos

Tarea	
Id	**T-017**
US	US-006
Puntos estimados	1
Descripción	Configuración de la sección “Editar pedido”

Tarea	
Id	**T-018**
US	US-006
Puntos estimados	1.5
Descripción	Configuración de la conexión a la base de datos

Tarea	
Id	**T-019**
US	US-007
Puntos estimados	4
Descripción	Implementación del módulo principal Frontend.

Tarea	
Id	**T-020**
US	US-007
Puntos estimados	2
Descripción	Configuración e implementación de registro e identificación.

Tarea	
Id	**T-021**
US	US-008
Puntos estimados	2
Descripción	Configuración del uso de sesiones.

Tarea	
Id	**T-022**
US	US-008
Puntos estimados	1.5
Descripción	Configuración de cuenta de usuario.

Tarea	
Id	**T-023**
US	US-008
Puntos estimados	2
Descripción	Conexión con la base de datos.

Tarea	
Id	**T-024**
US	US-008
Puntos estimados	1
Descripción	Eliminación de un usuario.

Tarea	
Id	**T-025**
US	US-009
Puntos estimados	2.5
Descripción	Conexión con base de datos.

Tarea	
Id	**T-026**
US	US-009
Puntos estimados	2
Descripción	Configuración de geolocalización.

Tarea	
Id	**T-027**
US	US-010
Puntos estimados	8
Descripción	Configuración de la base de datos.

Tarea	
Id	**T-028**
US	US-010
Puntos estimados	4
Descripción	Conexión de la API con la base de datos

Tarea	
Id	**T-029**
US	US-011
Puntos estimados	2.5
Descripción	Conexión con base de datos.

Tarea	
Id	**T-030**
US	US-011
Puntos estimados	5
Descripción	Configuración de las secciones de productos.

Tarea	
Id	**T-031**
US	US-012
Puntos estimados	2
Descripción	Conexión con base de datos.

Tarea	
Id	**T-032**
US	US-012
Puntos estimados	**2**
Descripción	Configuración de preferencias

Tarea	
Id	**T-033**
US	Preparación demo
Puntos estimados	
Descripción	Preparar las demostraciones funcionales del código ejecutado en ese Sprint para mostrárselo al cliente.

Tarea	
Id	**T-034**
US	Revisión de código
Puntos estimados	
Descripción	Revisar código.

Tarea	
Id	**T-035**
US	Testing BDD
Puntos estimados	
Descripción	Testeo funcional de la aplicación.

5.2.4. Estimación de tareas

La estimación de tareas se llevará a cabo al comienzo de cada Sprint, en el Sprint Planning. Para la estimación de tareas en Scrum se utilizará un procedimiento conocido como Planning Poker o Scrum Poker. Para llevar a cabo e Planning Poker, cada uno de los miembros del equipo de desarrollo tiene una baraja que contiene una pseudo-secuencia de Fibonacci modificada. Así cada participante tendrá las siguientes cartas: 0, 1/2, 1, 2, 3, 5, 8, 13, infinito y ?. El cero significa que la historia ya está hecha o no requiere ningún esfuerzo, el interrogante significa que nos falta información para estimar esa historia o tarea y el infinito es que es demasiado grande y hay que trocearla, en función del esfuerzo que prevé requerirá esa tarea. No es posible seleccionar un valor no incluido en la baraja. Una por una se leen y discuten las tareas. Una vez todos tienen claro en qué consiste cada uno elige una carta de una baraja en la cual hay, Solo estiman los que después desarrollan (ni el Scrum Master ni el Product Owner estiman, solo resuelven dudas).

5.3. Desarrollo iterativo

Una vez se ha realizado el análisis del proyecto se procederá al desarrollo iterativo del mismo. Estas iteraciones se ejecutan en bloques temporales cortos y fijos llamados Sprints. Un Sprint normal tendrá los siguientes eventos o ceremonias:

1. El Sprint Planning al comienzo del Sprint
2. Daily Scrums a diario
3. Un Sprint Review al final del Sprint para inspeccionar el incremento realizado.
4. Y, finalmente, una Retrospectiva para inspeccionar el equipo y levantar mejoras que se apliquen en el siguiente Sprint.
5. Adicionalmente se ha incorporado también una reunión que sirve para afinar y aclarar ciertas historias de usuario que pudieron quedar pendientes durante el Sprint Planning y que se conoce como Grooming o Refinement.

Antes de comenzar con las iteraciones se desarrollará el Sprint 0.

5.3.1. Sprint 0

El Sprint 0 es un paso previo de preparación que permitirá asentar las bases sobre las que se va a trabajar. En esta fase se establecen las responsabilidades y funciones tanto a nivel tecnológico como metodológico.

También se define la configuración de los Sprints, es decir su duración y la capacidad de trabajo del equipo en cada iteración:

- Cada Sprint tendrá una duración de 2 semanas. Esta duración no es de las más comunes, pero al tratarse de una aplicación de tamaño pequeño y de carácter visual (front-end) se ha llegado a la conclusión de que es la duración más adecuada.
- La capacidad de trabajo del equipo será de 50 horas (25 horas por desarrollador) de trabajo productivo, es decir, restando el tiempo estimado para reuniones, tereas de mejora, retrasos, corrección de errores, etc. Siendo los Sprint de dos semanas de duración, el tiempo de trabajo productivo, es decir, para las tareas asignadas a un US, de cada Sprint será de 100 horas. Sin embargo, este tiempo productivo es orientativo, ya que el tiempo total de trabajo por Sprint será de 140, es decir, quedarían 40 horas de trabajo no productivo o de ampliación del trabajo productivo para afrontar imprevistos. Por ello el trabajo productivo asignado a tareas podrá superar estas 100 horas establecidas por una necesidad de ajuste de tiempo, siempre y cuando no sea superada en exceso.
- La estimación que se realiza mediante el Planning Poker tendrá una equivalencia entre los puntos de estimación asignados (0, 1/2, 1, 2, 3, 5, 8, 13) y el tiempo en horas, siendo una unidad de estimación equivalente a 8 horas.

5.3.2. Sprint 1

El primer Sprint comenzará con el Sprint Planning, donde se establecerán las tareas del Product Backlog que van a desarrollarse ese Sprint y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas, se estimarán las que sean asignadas a este Sprint, teniendo en cuenta que 100 horas es el tiempo establecido para dar a las tareas que forman parte del desarrollo de un US, y se mostrará su estado de desarrollo (al tratarse del primer Sprint).

Tarea	US	Estado
T-001	US-001	Sin Comenzar
T-002	US-001	Sin Comenzar
T-003	US-001	Sin Comenzar
T-004	US-001	Sin Comenzar
T-005	US-002	Sin Comenzar
T-006	US-002	Sin Comenzar
T-007	US-002	Sin Comenzar
T-008	US-003	Sin Comenzar
T-009	US-003	Sin Comenzar
T-010	US-003	Sin Comenzar
T-011	US-004	Sin Comenzar
T-012	US-004	Sin Comenzar
T-013	US-005	Sin Comenzar
T-014	US-005	Sin Comenzar
T-015	US-005	Sin Comenzar
T-016	US-005	Sin Comenzar
T-017	US-006	Sin Comenzar
T-018	US-006	Sin Comenzar
T-019	US-007	Sin Comenzar
T-020	US-007	Sin Comenzar
T-021	US-008	Sin Comenzar
T-022	US-008	Sin Comenzar
T-023	US-008	Sin Comenzar
T-024	US-008	Sin Comenzar
T-025	US-009	Sin Comenzar
T-026	US-009	Sin Comenzar
T-027	US-010	Sin Comenzar
T-028	US-010	Sin Comenzar
T-029	US-011	Sin Comenzar
T-030	US-011	Sin Comenzar
T-031	US-012	Sin Comenzar
T-032	US-012	Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-001	5	3	4	**32**
T-002	5	3	4	**32**
T-003	2	2	2	**16**
T-004	3	3	3	**24**
				104

En total el trabajo estimado para realizar las tareas asignadas al Sprint Backlog es de 104 horas.

Una vez las anteriores tareas han sido introducidas en el Sprint Backlog, cada desarrollador procederá a ir cogiendo tareas para su desarrollo. A lo largo del desarrollo, cada día se llevará a cabo la Scrum Daily o DSTUM, de una duración aproximada de 15 min, donde se hablará de los avances realizados el día anterior, de los objetivos de este día y de los posibles impedimentos, dudas o bloqueaos que hayan podido encontrarse.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

		Tiempo por tarea		
	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-001	30	–	
	T-002	–	34	
	T-003	20	–	
	T-004	–	24	
	TOTAL	**50**	**58**	**108**
NO PROD	T-033	6	4	
	T-034	6	4	
	T-035	4	4	
	Otras	4	-	
	TOTAL	**20**	**12**	**32**
TOTAL		**70**	**70**	**140**

Atendiendo a los datos obtenidos en la tabla, el trabajo productivo total empleado en el primer Sprint ha sido de 108 horas, que está por encima de las 104 horas planificadas. No obstante, las tareas planificadas para el Sprint se han llevado a cabo con éxito y validadas por el Product Owner.

Una vez el Sprint ha finalizado se llevan a cabo dos ceremonias: el Sprint Review o Demo; y el Sprint Retrospective o Retro.

Durante la Demo, el equipo Scrum se reúne con el cliente y le enseña los progresos llevados a cabo en ese Sprint, y mantienen un diálogo con aclaraciones, sugerencias y análisis de las tareas desarrolladas. En este Sprint las funcionalidades que el Product Owner va a mostrar al cliente son las concernientes al US 001, la creación de la aplicación Web. El Product Owner junto con el resto del equipo de desarrollo, mostrará al cliente, quién confirmará si es lo que desea, o bien requiere de modificaciones, las siguientes funcionalidades:

- Diseño de la interfaz de la página web (T-001)
- Estructura del front end (T-004)
- Renderización de vistas (T-003)

Tras la demo, el cliente no está del todo satisfecho con el diseño de la interfaz de la página web, por lo que explica al Product Owner y a los desarrolladores los cambio que desea introducir al diseño ya realizado. Esta tarea pasará a finalizarse en el siguiente Sprint.

Finalmente, como último paso del Sprint, se llevará a cabo la Retro (Sprint Retrospective) para analizar qué ha ido bien durante el Sprint, qué ha fallado y qué se puede mejorar, pidiendo a los miembros del equipo Scrum que escriban notas –en post-it –para luego agruparlas y votar aquellos ítems más relevantes, dando la oportunidad a todos de hablar y expresar sus inquietudes. En este Sprint no ha habido grandes problemas o necesidades de cambio dignos de mención.

5.3.3. Sprint 2

El segundo Sprint comenzará con el Sprint Planning, donde se establecerán las nuevas tareas del Product Backlog que van a desarrollarse ese Sprint y las tareas del Sprint anterior sin finalizar, y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas actualizadas con el trabajo desarrollado en el primer Sprint.

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32	30	En proceso
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-005	US-002			Sin Comenzar
T-006	US-002			Sin Comenzar
T-007	US-002			Sin Comenzar
T-008	US-003			Sin Comenzar
T-009	US-003			Sin Comenzar
T-010	US-003			Sin Comenzar
T-011	US-004			Sin Comenzar
T-012	US-004			Sin Comenzar
T-013	US-005			Sin Comenzar
T-014	US-005			Sin Comenzar
T-015	US-005			Sin Comenzar
T-016	US-005			Sin Comenzar
T-017	US-006			Sin Comenzar
T-018	US-006			Sin Comenzar
T-019	US-007			Sin Comenzar
T-020	US-007			Sin Comenzar
T-021	US-008			Sin Comenzar
T-022	US-008			Sin Comenzar
T-023	US-008			Sin Comenzar
T-024	US-008			Sin Comenzar
T-025	US-009			Sin Comenzar
T-026	US-009			Sin Comenzar
T-027	US-010			Sin Comenzar
T-028	US-010			Sin Comenzar
T-029	US-011			Sin Comenzar
T-030	US-011			Sin Comenzar
T-031	US-012			Sin Comenzar
T-032	US-012			Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-001	2	2	2	**16**
T-027	8	8	8	**64**
T-028	4	4	4	**32**
				112

En este Sprint el tiempo estimado para desarrollar las tareas del Sprint Backlog es de 112, que esta por encima de las 100 horas previamente establecidas, pero el equipo ha hablado y ha decidido que pueden sacar las tareas dentro del Sprint. Como en el anterior Sprint, cada desarrollador cogerá una tarea según finalice la anterior. En este caso, el desarrollador que comenzó la tarea T-001 en el primer Sprint, continuará con su desarrollo.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-001	24	–	
	T-027	–	66	
	T-028	31	–	
	TOTAL	**55**	**66**	**121**
NO PROD	T-033	3	0	
	T-034	4	0	
	T-035	5	–	
	Otras	3	4	
	TOTAL	**15**	**4**	**19**
TOTAL		**70**	**70**	**140**

En este Sprint se han empleado 121 horas de trabajo productivo y 19 horas de trabajos de mejora de código, testing, etc. En este Spring los resultados obtenidos no han sido satisfactorios por varias causas:

- La tarea T-001, que venía del Sprint anterior no se ha estimado correctamente y se ha ido 8 horas del tiempo planificado para su finalización.

- La tarea T-027 ha resultado ser más compleja de lo esperado y ha acarreado una serie de problemas al desarrollador encargado de su ejecución que no ha podido finalizarla.

Una vez el Sprint ha finalizado se llevan a cabo dos ceremonias: el Sprint Review o Demo; y el Sprint Retrospective o Retro.

En la Demo se muestra al cliente la tarea T-001, con la cual no estuvo satisfecho en el primer Sprint y pidió una serie de modificaciones que se han llevado a cabo, y en esta ocasión el cliente si da el visto bueno.
En cuanto a la tarea T-028, aunque está finalizada y aprobada por el Product Owner, no es posible llevar a cabo la demostración, ya que la configuración de la base de datos aún no ha sido finalizada (T-027).

En la Retro, el equipo analiza los problemas que han acontecido en este Sprint y las enseñanzas para que estos errores no se repitan. Las conclusiones sacadas se citan a continuación:

- De la tarea T-001 se ha llegado a la conclusión que deben establecerse mas detalladamente los deseos del cliente y la comunicación debe ser más fluida.
- De la tarea T-027 se ha llegado a la conclusión de que las tareas no deben ser tan largas, ya que los Sprints son solamente de dos semanas, por lo que, ante cualquier contratiempo, la entrega de desarrollo puede fallar, como ha sido el caso, y a no poder realizar la demostración de otras tareas que puedan estar relacionadas con esta. Por tanto, para futuras tareas de longitud similar o mayor se llevará a cabo una división de la tarea en varias más pequeñas.

5.3.4. Sprint 3

El tercer Sprint comenzará con el Sprint Planning, donde se establecerán las nuevas tareas del Product Backlog que van a desarrollarse ese Sprint y las tareas del Sprint anterior sin finalizar, y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas actualizadas con el trabajo desarrollado en el segundo Sprint.

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32+16	30+22	Finalizada
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-027	US-010	64	66	En proceso
T-028	US-010	32	31	Finalizada
T-005	US-002			Sin Comenzar
T-006	US-002			Sin Comenzar
T-007	US-002			Sin Comenzar
T-008	US-003			Sin Comenzar
T-009	US-003			Sin Comenzar
T-010	US-003			Sin Comenzar
T-011	US-004			Sin Comenzar
T-012	US-004			Sin Comenzar
T-013	US-005			Sin Comenzar
T-014	US-005			Sin Comenzar
T-015	US-005			Sin Comenzar
T-016	US-005			Sin Comenzar
T-017	US-006			Sin Comenzar
T-018	US-006			Sin Comenzar
T-019	US-007			Sin Comenzar
T-020	US-007			Sin Comenzar
T-021	US-008			Sin Comenzar
T-022	US-008			Sin Comenzar
T-023	US-008			Sin Comenzar
T-024	US-008			Sin Comenzar
T-025	US-009			Sin Comenzar
T-026	US-009			Sin Comenzar
T-029	US-011			Sin Comenzar
T-030	US-011			Sin Comenzar
T-031	US-012			Sin Comenzar
T-032	US-012			Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-027	1	1	1	**8**
T-008	0,5	0,5	0,5	**4**
T-009	0,5	0,5	0,5	**4**
T-010	2	3	2,5	**20**
T-019	3	5	4	**32**
T-020	2	2	2	**16**
T-005	0,5	0,5	0,5	**4**
T-006	0,5	0,5	0,5	**4**
T-007	1	2	1,5	**12**
				104

En este Sprint, el tiempo estimado para tareas relacionadas con una Historia de Usuario es de 104. Entre estas tareas se encuentra la T-028, proveniente del Sprint anterior.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-027	6	–	
	T-008	–	4	
	T-009	–	4	
	T-010	–	18	
	T-019	33	–	
	T-020	16	–	
	T-005	–	4	
	T-006	–	4	
	T-007	–	13	
	TOTAL	**55**	**47**	**102**
NO PROD	T-033	3	8	
	T-034	4	4	
	T-035	6	6	
	Otras	2	5	
	TOTAL	**15**	**23**	**38**
TOTAL		**70**	**70**	**140**

Viendo los resultados de este Sprint, se observa que el tiempo de ejecución de las tareas productivas ha sido de 102 horas, es decir, se ha hecho antes de las 104 horas estimadas, por lo que se ha contado con más tiempo para probar test, para mejora del código y cualimetría, y para preparar la Demo para el cliente. Esta mejora es debida en parte a los puntos de mejora establecidos en la Retro del Sprint anterior y a la motivación extra del equipo por salir de un Sprint que no fue tan bien como se esperaba.

Como siempre, al final del Sprint vienen la Retro y la Demo. En la Demo, el Product Owner le hace al cliente una demostración al cliente de las siguientes funcionalidades:

- Utilización de la base de datos (T-027 y T-028, ambas del Sprint anterior).
- Alta de un usuario (T-008 y T-009).
- Seguridad de las cuentas (T-019 y T-020).
- Publicidad (T-005, T-006 y T-007).

En la Retro se analizan las mejoras en la metodología de trabajo de este Sprint:

- Mejora de comunicación con el cliente.
- Tareas más reducidas.
- Mayor motivación.

Y se marca el objetivo de continuar con esta dinámica para el próximo Sprint.

5.3.5. Sprint 4

El cuarto Sprint comenzará con el Sprint Planning, donde se establecerán las nuevas tareas del Product Backlog que van a desarrollarse ese Sprint y las tareas del Sprint anterior sin finalizar, y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas actualizadas con el trabajo desarrollado en el tercer Sprint.

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32+16	30+22	Finalizada
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-027	US-010	64+8	66+6	Finalizada
T-028	US-010	32	31	Finalizada
T-008	US-003	4	4	Finalizada
T-009	US-003	4	4	Finalizada
T-010	US-003	20	18	Finalizada
T-019	US-007	32	33	Finalizada
T-020	US-007	16	16	Finalizada
T-005	US-002	4	4	Finalizada
T-006	US-002	4	4	Finalizada
T-007	US-002	12	13	Finalizada
T-011	US-004			Sin Comenzar
T-012	US-004			Sin Comenzar
T-013	US-005			Sin Comenzar
T-014	US-005			Sin Comenzar
T-015	US-005			Sin Comenzar
T-016	US-005			Sin Comenzar
T-017	US-006			Sin Comenzar
T-018	US-006			Sin Comenzar
T-021	US-008			Sin Comenzar
T-022	US-008			Sin Comenzar
T-023	US-008			Sin Comenzar
T-024	US-008			Sin Comenzar
T-025	US-009			Sin Comenzar
T-026	US-009			Sin Comenzar
T-029	US-011			Sin Comenzar
T-030	US-011			Sin Comenzar
T-031	US-012			Sin Comenzar
T-032	US-012			Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-013	1	1	1	**8**
T-014	1	2	1,5	**12**
T-015	1	1	1	**8**
T-016	2	3	2,5	**20**
T-017	1	1	1	**8**
T-018	1	2	1,5	**12**
T-025	2	3	2,5	**20**
T-026	2	2	2	**16**
				104

En este Sprint, el tiempo estimado para tareas relacionadas con una Historia de Usuario es de 104.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-013	8	–	
	T-014	14	–	
	T-015	8	–	
	T-016	20	–	
	T-017	–	7	
	T-018	–	12	
	T-025	–	18	
	T-026	–	16	
	TOTAL	**50**	**53**	**103**
NO PROD	T-033	5	4	
	T-034	4	4	
	T-035	6	4	
	Otras	5	5	
	TOTAL	**20**	**17**	**37**
TOTAL		**70**	**70**	**140**

Atendiendo a los resultados de este Sprint recogidos en la tabla anterior, se observa que el tiempo de ejecución de las tareas productivas ha sido de 103 horas, es decir, se ha hecho antes de las 104 horas estimadas, por lo que se ha contado con tiempo suficiente para probar test, para mejora del código y cualimetría, y para preparar la Demo para el cliente.

En la Demo, el Product Owner le hace al cliente una demostración al cliente de las siguientes funcionalidades:

- Las diferentes opciones de compra (T-013 y T-015, ambas del Sprint anterior).
- Confirmación de compras (T-014).
- Editar pedido realizado (T-017).
- Seguimiento del pedido (T-026).

En la Retro En este Sprint no ha habido grandes problemas o necesidades de cambio dignos de mención. Se ha continuado con la dinámica positiva del Sprint anterior.

5.3.6. Sprint 5

El quinto Sprint comenzará con el Sprint Planning, donde se establecerán las nuevas tareas del Product Backlog que van a desarrollarse ese Sprint y las tareas del Sprint anterior sin finalizar, y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas actualizadas con el trabajo desarrollado en el cuatro Sprint.

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32+16	30+22	Finalizada
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-027	US-010	64+8	66+6	Finalizada
T-028	US-010	32	31	Finalizada
T-008	US-003	4	4	Finalizada
T-009	US-003	4	4	Finalizada
T-010	US-003	20	18	Finalizada
T-019	US-007	32	33	Finalizada
T-020	US-007	16	16	Finalizada
T-005	US-002	4	4	Finalizada
T-006	US-002	4	4	Finalizada
T-007	US-002	12	13	Finalizada
T-013	US-005	8	8	Finalizada
T-014	US-005	12	14	Finalizada
T-015	US-005	8	8	Finalizada
T-016	US-005	20	20	Finalizada
T-017	US-006	8	7	Finalizada
T-018	US-006	12	12	Finalizada
T-025	US-009	20	18	Finalizada
T-026	US-009	16	16	Finalizada
T-011	US-004			Sin Comenzar
T-012	US-004			Sin Comenzar
T-021	US-008			Sin Comenzar
T-022	US-008			Sin Comenzar
T-023	US-008			Sin Comenzar
T-024	US-008			Sin Comenzar
T-029	US-011			Sin Comenzar
T-030	US-011			Sin Comenzar
T-031	US-012			Sin Comenzar
T-032	US-012			Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-029	2	3	2,5	**20**
T-030	3	5	4	**32**
T-031	2	2	2	**16**
T-032	2	1	1,5	**12**
T-011	0,5	0,5	0,5	**4**
T-012	0,5	0,5	0,5	**4**
				88

En este Sprint puede verse que el tiempo estimado es más reducido (88 horas) debido a que el proyecto esta finalizando y para el último Sprint va a quedar únicamente una Historia de Usuario por desarrollar, pero era demasiado grande como para acoplarla en este Sprint. No obstante, estando en los últimos pasos del proyecto será conveniente aprovechar el tiempo restante, es decir, el tiempo fuera de la estimación de tareas pertenecientes a US's, para tareas de mejora de calidad.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-029	23	–	
	T-030	32	–	
	T-031	–	16	
	T-032	–	14	
	T-011	–	5	
	T-012	–	6	
	TOTAL	**55**	**41**	**96**
NO PROD	T-033	3	8	
	T-034	4	6	
	T-035	5	9	
	Otras	3	6	
	TOTAL	**15**	**29**	**44**
TOTAL		**70**	**70**	**140**

Analizando los resultados anteriores una vez finalizado el Sprint, se observa que el tiempo real de trabajo productivo (96 horas) ha sido superior al estimado (88 horas). Esto puede ser debido a una falta de concentración por parte del equipo provocada por la menor cantidad de tareas de US a realizar, lo que ha supuesto una relajación que se ha traducido en una caída de la productividad.

En la Demo, el Product Owner le hace al cliente una demostración al cliente de las siguientes funcionalidades:

- La configuración de las secciones de productos (T-030).
- La vista de preferencias (T-032).
- Mail de confirmación de pedido (T-011).
- Pop-up con la información de la compra (T-012).

El la Retro se analiza principalmente la caída de productividad, llegando el propio equipo a la conclusión de su relajamiento y el motivo que lo causó, y estableciendo un objetivo de mantener el rendimiento al máximo para el siguiente y último Sprint.

5.3.7. Sprint 6

El sexto Sprint comenzará con el Sprint Planning, donde se establecerán las nuevas tareas del Product Backlog que van a desarrollarse ese Sprint y las tareas del Sprint anterior sin finalizar, y que pasarán a formar parte del Sprint Backlog. A continuación, se enumerarán las tareas actualizadas con el trabajo desarrollado en el quinto Sprint.

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32+16	30+22	Finalizada
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-027	US-010	64+8	66+6	Finalizada
T-028	US-010	32	31	Finalizada
T-008	US-003	4	4	Finalizada
T-009	US-003	4	4	Finalizada
T-010	US-003	20	18	Finalizada
T-019	US-007	32	33	Finalizada
T-020	US-007	16	16	Finalizada
T-005	US-002	4	4	Finalizada
T-006	US-002	4	4	Finalizada
T-007	US-002	12	13	Finalizada
T-013	US-005	8	8	Finalizada
T-014	US-005	12	14	Finalizada
T-015	US-005	8	8	Finalizada
T-016	US-005	20	20	Finalizada
T-017	US-006	8	7	Finalizada
T-018	US-006	12	12	Finalizada
T-025	US-009	20	18	Finalizada
T-026	US-009	16	16	Finalizada
T-029	US-011	20	23	Finalizada
T-030	US-011	32	32	Finalizada
T-031	US-012	16	16	Finalizada
T-032	US-012	12	14	Finalizada
T-011	US-004	4	5	Finalizada
T-012	US-004	4	6	Finalizada
T-021	US-008			Sin Comenzar
T-022	US-008			Sin Comenzar
T-023	US-008			Sin Comenzar
T-024	US-008			Sin Comenzar

En el Sprint Planning, los miembros del equipo de desarrollo estiman las siguientes tareas para desarrollar en este Sprint. En la siguiente tabla se puede apreciar la estimación realizada por cada desarrollador en Planning Poker (P.P.), la estimación total y la estimación en horas.

Tarea	P. P. 1	P. P. 2	Estimación	Estimación Horas
T-021	2	2	2	**16**
T-022	2	1	1,5	**12**
T-023	2	2	2	**16**
T-024	1	1	1	**8**
				52

En este último Sprint hay 52 horas previstas para tareas productivas y terminar de desarrollar el último US. El resto del tiempo se empleará para pruebas funcionales de la aplicación y cualimetría de código.

Al llegar al final del Sprint, cada desarrollador ha llevado a cabo las siguientes tareas en los siguientes intervalos de tiempo:

	Tarea	Desarrollador 1	Desarrollador 2	TOTAL
PROD	T-021	14	–	
	T-022	12	–	
	T-023	–	15	
	T-024	–	6	
	TOTAL	**26**	**21**	**47**
NO PROD	T-033	10	10	
	T-034	8	8	
	T-035	20	20	
	Otras	6	11	
	TOTAL	**44**	**49**	**93**
TOTAL		**70**	**70**	**140**

En este último Sprint se ha llevado a cabo el trabajo productivo en 47 horas, que está por debajo del tiempo estimado, de 52 horas. El resto del tiempo se ha empleado en depurar y mejorar el rendimiento del código y en realizar pruebas funcionales.

En la Demo, el Product Owner le hace al cliente una demostración al cliente de las siguientes funcionalidades:

- La configuración de las cuentas de los usuarios (T-022).
- Configuración del uso de sesiones (T-021).
- Eliminación de usuario (T-011).
- Demostración global, probando diferentes combinaciones y opciones.

En la Retro se felicita al equipo por el buen trabajo realizado y por haber seguido las pautas de motivación a las que se comprometieron en el anterior Sprint, lo que ha permitido que el trabajo se realice en un tiempo inferior al estimado.

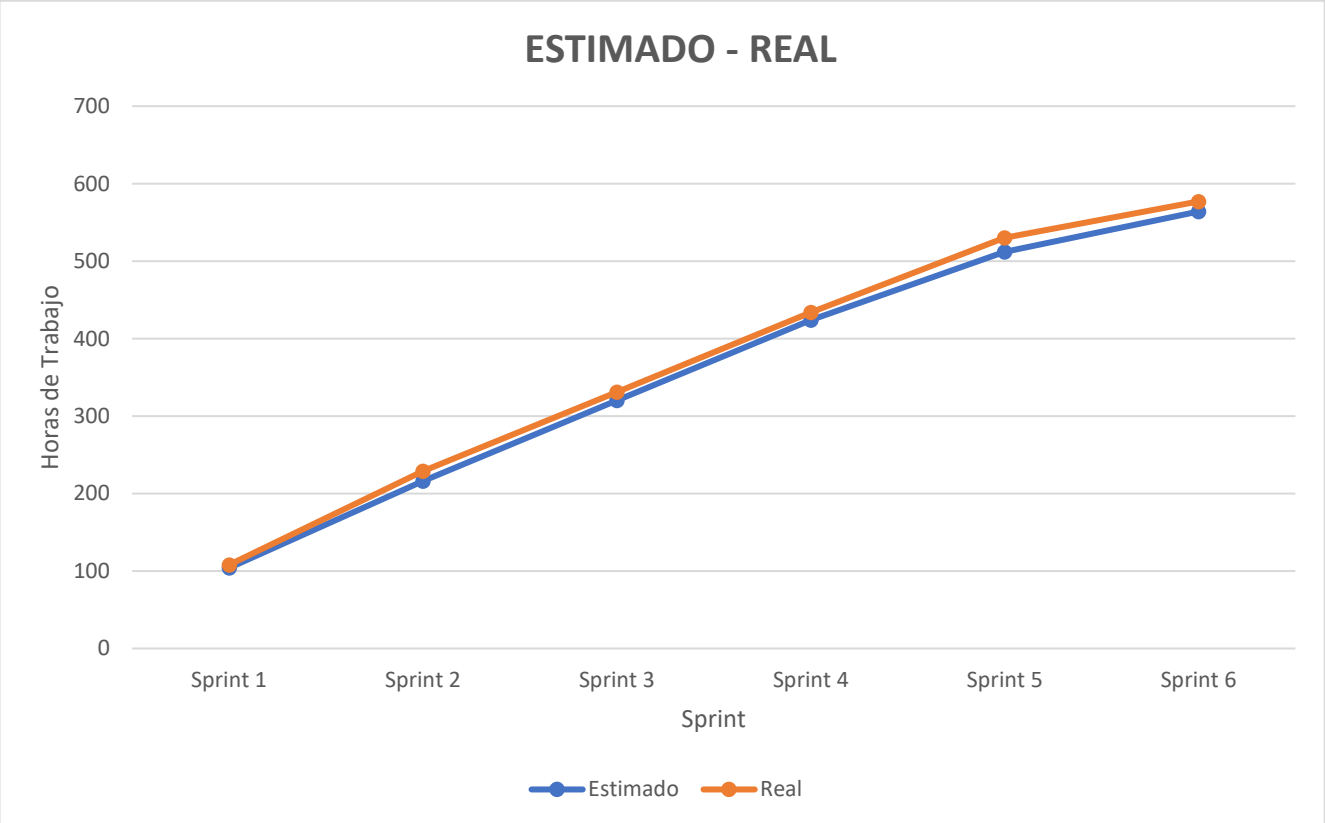
Así pues, el resumen de tareas realizadas quedaría:

Tarea	US	Estimación	Tiempo Real	Estado
T-001	US-001	32+16	30+22	Finalizada
T-002	US-001	32	34	Finalizada
T-003	US-001	16	20	Finalizada
T-004	US-001	24	24	Finalizada
T-027	US-010	64+8	66+6	Finalizada
T-028	US-010	32	31	Finalizada
T-008	US-003	4	4	Finalizada
T-009	US-003	4	4	Finalizada
T-010	US-003	20	18	Finalizada
T-019	US-007	32	33	Finalizada
T-020	US-007	16	16	Finalizada
T-005	US-002	4	4	Finalizada
T-006	US-002	4	4	Finalizada
T-007	US-002	12	13	Finalizada
T-013	US-005	8	8	Finalizada
T-014	US-005	12	14	Finalizada
T-015	US-005	8	8	Finalizada
T-016	US-005	20	20	Finalizada
T-017	US-006	8	7	Finalizada
T-018	US-006	12	12	Finalizada
T-025	US-009	20	18	Finalizada
T-026	US-009	16	16	Finalizada
T-029	US-011	20	23	Finalizada
T-030	US-011	32	32	Finalizada
T-031	US-012	16	16	Finalizada
T-032	US-012	12	14	Finalizada
T-011	US-004	4	5	Finalizada
T-012	US-004	4	6	Finalizada
T-021	US-008	16	14	Finalizada
T-022	US-008	12	12	Finalizada
T-023	US-008	16	15	Finalizada
T-024	US-008	8	6	Finalizada

Donde las horas trabajadas productivas estimadas y reales han sido:

	Estimado	Real
Sprint 1	**104**	**108**
Sprint 2	**216**	**229**
Sprint 3	**320**	**331**
Sprint 4	**424**	**434**
Sprint 5	**512**	**530**
Sprint 6	**564**	**577**

En la gráfica siguiente se aprecia la desviación que ha habido entre el tiempo estimado y el tiempo que finalmente se ha empleado para realizar las tareas que formaban parte de las Historias de Usuario.



5.4. Presupuesto

En este caso, siendo el proyecto desarrollado por una empresa de consultoría tecnológica especializada en aplicaciones web, el presupuesto irá en función de las horas trabajadas por los miembros del equipo Scrum. En la siguiente tabla se observa el procedimiento:

	Cantidad	Horas/trabajador	Horas Totales	Euros/hora	Euros Totales
Desarrollador	2	420	840	7	5880
Scrum Master	1	210	210	10	2100
Product Owner	1	210	210	12	2520
					10500 €

Como se aprecia en la tabla, las horas totales empleadas por los desarrolladores han sido 840 horas, es decir 70 horas por trabajador y por Sprint. En cuanto a las horas empleadas por el Scrum Master y el Product Owner, serán de 210 horas cada uno, es decir, 35 horas por Sprint. El tiempo empleado por Scrum Master y Product Owner es inferior al de los desarrolladores debido a que al tratarse de un proyecto de pequeña envergadura, estos dos trabajaban a tiempo parcial en este proyecto, quedándoles tiempo para poder llevar otros proyectos diferentes. Por tanto, el precio final que ha costado desarrollar el proyecto ha sido de 10500 €.

6. CONCLUSIÓN

Una vez finalizado el proyecto, se analizarán los objetivos previamente establecidos para comprobar si, efectivamente estos se han cumplido.

En primer lugar, el trabajo desarrollado ayudará a quien lo lea a comprender mejor en que consisten las metodologías Agile de proyectos de desarrollo de software, así como las herramientas de las que se sirve para gestionar con éxito un proyecto. Además, el ejemplo desarrollado de la metodología Scrum ayuda a comprender mejor como llevar a la práctica el uso de esta metodología y como ha de usarse correctamente.

También han quedado claros cuales son los principios y valores que rigen las metodologías Agile y la forma de trabajo que se espera que desarrollen aquellos que la van a emplear.

Así mismo, han quedado esclarecidas las ventajas de Agile frente a la metodología de gestión tradicional, así como los problemas que puede presentar Agile, ya que, aunque mejor que la tradicional, no es perfecta, como ha podido verse.

Por último, en cuanto a lo personal, el desarrollo de este trabajo me ha ayudado a comprender mejor el trabajo que desarrollo día a día en mi empresa y a crecer como profesional.

BIBLIOGRAFÍA

https://agilemanifesto.org/iso/es/manifesto.html

https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog/

https://www.cesarpiqueras.com/implicar-a-las-personas/

https://www.obs-edu.com/es/blog-project-management/metodologias-agiles/

http://www.ceolevel.com

https://www.viewnext.com/el-ciclo-de-vida-de-las-metodologias-agiles-de-desarrollo/

https://proyectosagiles.org/ejecucion-iteracion-sprint/

https://www.agilealliance.org/glossary/xp/

https://programacionymas.com/blog/scrum-product-owner

https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html

https://retos-directivos.eae.es/usos-y-limitaciones-de-la-metodologia-scrum/

https://www.paradigmadigital.com/techbiz/los-51-valores-de-equipos-scrum-altamente-efectivos/

http://ingenieriadesoftware.mex.tl/61154_ASD.html

https://www.workfront.com/blog/rolling-out-scrumban-with-andrea-fryrear

http://desarrolloadaptativodesoftware.blogspot.com/2011/06/desarrollo-adaptativo-de-software-das.html

https://kanbantool.com/es/scrumban-scrum-y-kanban

http://scrum.menzinsky.com/2017/12/como-funciona-scrum-de-scrums.html

https://medium.com/forecast-en-espa%C3%B1ol/agile-vs-cascada-parte-3-de-5-costo-funcionalidad-y-tiempo-8d9bcb254f36

https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

https://www.obs-edu.com/es/blog-project-management/actualidad-project-management

https://managementplaza.es/blog/ciclo-de-vida-agil/

https://www.northware.mx/wp-content/uploads/2011/12/ciclo-de-entrega-agil.png

https://www.javiergarzas.com/2012/09/metodologia-gil-fdd-1.html

https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

https://www.deustoformacion.com/blog/empresa/que-es-gestion-stocks-concepto-que-hay-que-conocer-bien

http://blog.metodoconsultores.com/scrum-gestion-proyectos/

https://www.shopify.es/blog/17011080-lo-que-debes-saber-sobre-el-servicio-post-venta

https://samuelcasanova.com/2016/11/como-dividir-historias-de-usuario-en-tareas/

https://blog.opensistemas.com/estimacion-de-tareas-en-scrum/