Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

MASTER EN DIRECCIÓN DE PROYECTOS

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

IMPLANTACIÓN DE METODOLOGÍAS ÁGILES EN UN EQUIPO DE DESARROLLO DE SOFTWARE

Autor: D. Jesús Casal Martínez

Tutor: D. David Jesús Poza García

Valladolid, Septiembre, 2019

RESUMEN

Ante la falta de productividad observada en un equipo pequeño de desarrollo Software, se decide implantar algunos principios de las metodologías ágiles más comunes en este sector. A lo largo de 11 meses, se incorporan prácticas de LEAN Development, Extreme Programming y SCRUM, haciendo continuos análisis de situación para comparar la evolución del equipo con los objetivos que se persiguen.

El equipo de desarrollo Software está condicionado por los recursos humanos y materiales de los que dispone, el entorno laboral en el que se encuentra y los objetivos comerciales que fija la compañía. Durante 11 meses, se adoptarán medidas con el objetivo de mejorar la productividad, el ambiente de trabajo y continuar la línea estratégica definida por la dirección.

PALABRAS CLAVE: Metodologías ágiles, Desarrollo de Software, LEAN Development, Extreme Programming, SCRUM.

ÍNDICE

1. INTRODUCCIÓN

1.1. Objeto y motivación

En el marco de un equipo de desarrollo de Software, que da soluciones tecnológicas a personas y empresas, se detectan ciertas improductividades que van aumentando con el paso del tiempo. Las consecuencias más evidentes son la pérdida de clientes, el desperdicio de una gran cantidad de recursos y, por ende, inestabilidad. Antes de que la situación se convierta en insostenible, la dirección de la compañía en la que se encuentra (una consultoría de ingeniería de Valladolid) busca dar solución a estos problemas.

El entorno, los malos hábitos, la mala comunicación y la falta de formación son causas evidentes de estas improductividades, a las que se puede encontrar solución por medio de la implementación de conocimientos en **gestión de proyectos**.

Las metodologías ágiles nacen a finales del Siglo XX para simplificar la ejecución de proyectos de pequeña entidad, y desde sus inicios han estado vinculadas al desarrollo de Software.

El equipo en el que nos encontramos ejecuta proyectos utilizando una metodología tradicional, que imita a la que viene utilizándose en los proyectos de ingeniería de la compañía. Por ello, como decisión conjunta de la dirección y del equipo de trabajo, se decide recibir formación en metodologías ágiles en el desarrollo de Software e implantar sucesivas acciones a lo largo de un periodo de tiempo prolongado.

1.2. Contexto

Nos encontramos en una empresa española que presta servicios integrales de ingeniería y consultoría tecnológica, proporcionando soluciones de alto valor añadido adaptadas a las necesidades de sus clientes.

En 2008 surge la idea de constituir una organización que aporte soluciones a las carencias de tecnificación e innovación existentes en muchas empresas. Aquellos inicios se convirtieron en la realización de proyectos multidisciplinares en sectores muy diversos hasta lograr en la actualidad ser proveedores de servicios globales de ingeniería, consultoría y TICs.

Se proporcionan conocimientos técnicos que permiten diseñar o evolucionar nuevos productos, mejorar productividades e implantar herramientas para gestionar empresas de manera eficiente, incluyendo la investigación tecnológica como base del crecimiento propio y de los clientes.

Los proyectos definen de forma precisa la distribución de usos y espacios, la utilización de materiales y tecnologías, y la justificación técnica del cumplimiento de las especificaciones requeridas por la normativa técnica aplicable.

Se pone a disposición de cada uno de los clientes un equipo de profesionales multidisciplinares altamente cualificados capaces de abordar proyectos de diferentes tipologías:

- Redacción de estudios, anteproyectos, delineación, proyectos básicos, proyectos de ejecución, direcciones de obra, asistencia técnica, coordinaciones de seguridad y salud y legalizaciones (gestión de licencias/permisos y tramitación con la Administración).

- Cobertura integral del desarrollo constructivo de proyectos mediante el servicio "llave en mano" de Gestión integrada de proyectos multidisciplinares de ingeniería y arquitectura.

- Diseño y desarrollo de proyectos de innovación en el marco de la ingeniería y las nuevas tecnologías, herramientas de software personalizado para la gestión de personas y procesos, sistemas de gestión empresarial (calidad, medio ambiente, RSC, eficiencia energética…).

Hoy por hoy nos la compañía se encuentra en continuo crecimiento, siendo capaces de atender la demanda de trabajo a nivel nacional desde las oficinas de Valladolid. La trayectoria actual es la punta del iceberg de nuevos desarrollos que van a permitir al entorno evolucionar hacia valores de sostenibilidad e innovación.

La empresa se conforma de dos departamentos: Ingeniería (que realiza proyectos de instalaciones energéticas, obra civil y arquitectura) y Tecnologías de la Información. Es en este último el marco del caso que se expone a lo largo del presente trabajo.

El **departamento de TICs** está formado por profesionales muy formados y con experiencia en el desarrollo de software. Las áreas de mayor especialización comprenden el desarrollo de aplicaciones de movilidad industrial y empresarial y portales Web, todo ello "a medida", dando como resultado productos personalizados y personalizables, asegurando la satisfacción del cliente.

1.3. El desarrollo de Software

Todo sistema informático está constituido por Hardware y Software. El Hardware es la parte física del sistema, es decir, todo el conjunto que suman el procesador, disco duro, unidades extraíbles, monitor, ratón… Todo lo demás es lo que se conoce como **Software**. El Software es el soporte lógico que hace funcionar a un sistema informático.

Formalmente, según el estándar del Instituto de Ingeniería Eléctrica y Electrónica (IEEE, por sus siglas en inglés), el Software "es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación" [1].

El Software de un programa normalmente está escrito en lenguajes de programación de alto nivel. A través de este, se le desarrollan instrucciones y reglas que permiten ejecutar ciertas tareas en un dispositivo programable.

El **desarrollo de Software** consiste en solucionar problemas específicos de personas o entidades a través de la programación, diseñando y desarrollando productos que faciliten cualquier tipo de actividad a través de la tecnología [2].

El resultado de un desarrollo de Software es un programa capaz de leer diferentes secuencias de instrucciones, guardadas en la memoria de un dispositivo, y ejecutar tareas de forma autónoma. Los ejemplos más intuitivos de Software son cualquier programa informático, las aplicaciones móviles o las páginas Web, pero también forma parte de este grupo la programación de un electrodoméstico o muchas de las funciones de un automóvil.

2. METODOLOGÍAS ÁGILES EN EL DESARROLLO DE SOFTWARE

El desarrollo de Software es una actividad actualmente muy demandada. En la mayoría de situaciones no es necesaria una gran cantidad de recursos (ni humanos ni materiales) para llevar a cabo desarrollos exitosos, pero el uso de metodologías tradicionales de trabajo puede ser excesivamente complejo para este tipo de proyectos [3].

A lo largo del Siglo XXI se ha ido extendiendo el uso de metodologías ágiles para el desarrollo de Software informático. La base de todas ellas es el establecimiento de ciclos de trabajo, al final de cada cual se obtendrá un producto funcional cada vez más completo (procesos incrementales). Se busca aumentar la confianza del cliente implicándolo en el proceso, cooperando con el equipo y obteniendo resultados de manera periódica. Estas metodologías se caracterizan por reducir la gran cantidad de documentación que se genera en los proyectos que siguen metodologías tradicionales, adaptarse continuamente a las circunstancias y las necesidades del cliente, y presentar un desarrollo incremental de un producto.

Manifiesto ágil del desarrollo de Software

En febrero de 2001, se celebró en Snowbird (Utah, EEUU) una reunión con diecisiete críticos para establecer unos principios de mejorar del desarrollo de Software. Como conclusión de esta reunión, se extrajo un documento llamado **Manifiesto Ágil**. El Manifiesto Ágil es un conjunto de reglas, normas y directrices que debe cumplir una metodología para ser considerada como ágil. Aunque en el Manifiesto Ágil se valoran todas las partes implicadas en el desarrollo de un proyecto son importantes, en ocasiones ciertos elementos se encuentran y es necesario establecer prioridades. En concreto, se identifican los siguientes cuatro aspectos (***Fig 1.***):

Fig 1 Valores ágiles [4]

- Los **individuos e interacciones** están por encima de procesos y herramientas

  El trabajo en equipo cobra especial importancia. El equipo debe trabajar de manera sincronizada y en conjunto para cumplir con los objetivos propuestos, sin descuidar la metodología ni los procesos preestablecidos [5].

- El **Software funcional** están por encima de la documentación exhaustiva

  Las metodologías tradicionales tienden a generar una gran cantidad de documentación. Muchos de los artefactos en los que se basan consisten en dejar constancia por escrito de los pasos seguidos durante la realización de un proyecto. No obstante, el desarrollo de Software habitualmente implica proyectos de menor magnitud, y generar toda esa documentación puede generar desgaste en el equipo mientras que no le aporta ningún valor añadido al producto que se genera.

  Las metodologías ágiles favorecen que el tiempo de desarrollo sea invertido en asegurar la correcta funcionalidad del software frente a la redacción de documentación en exceso. Esto no implica que no exista ningún tipo de documentación, sino que es necesario reducir el contenido al contenido imprescindible.

- La **colaboración con el cliente** es más importante que la negociación contractual

  El cliente es prioritario y parte elemental de todos los desarrollos. La relación con el cliente y la comunicación con el equipo de desarrollo son fundamentales para que el proyecto sea exitoso. Uno de los principios de todas las metodologías ágiles es generar confianza haciendo que el cliente se sienta involucrado en todo el proceso.

- La **respuesta ante el cambio** está por encima del seguimiento de un plan

  A pesar de que tener un plan definido que seguir es muy importante en cualquier tipo de proyecto, el desarrollo de Software se caracteriza por ser muy dinámico y variable en el tiempo. Es necesario que el equipo esté capacitado para responder ante los posibles cambios y variaciones durante un desarrollo.

Una de las principales ventajas que encontramos en el uso de metodologías ágiles es su adaptación. Si bien es cierto que nacen de la necesidad de simplificar las metodologías tradicionales para proyectos más sencillos, también pueden atacar proyectos grandes y complejos [6] [7].

La ventaja que obtenemos de la implementación de metodologías ágiles en proyectos complejos es que no hace falta esperar al final del proyecto para obtener el resultado del mismo, sino que también se trabajará por entregas funcionales, iterativas e incrementales, a través de las cuales el cliente podrá conocer cómo crece y cómo se construye su producto (***Fig 2.***).

Fig 2 Etapas del desarrollo de Software con metodologías ágiles [8]

Las metodologías ágiles cumplen con una serie de etapas (o ciclos de trabajo) en las que se le da prioridad al subproyecto (o parte parcial de la solución) que se está atacando en ellas, frente a la solución global.

En la **Tabla 1** podemos ver un resumen de las diferencias más importantes entre las metodologías tradicionales y las metodologías ágiles.

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos
Pocos Roles, más genéricos y flexibles	Más Roles, más específicos
No existe un contrato tradicional, debe ser bastante flexible	Existe un contrato prefijado
Cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos
La arquitectura se va definiendo y mejorando a lo largo del proyecto	Se promueve que la arquitectura se defina tempranamente en el proyecto
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo	Énfasis en la definición del proceso: roles, actividades y artefactos
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

Tabla 1 Diferencias entre metodologías ágiles y metodologías tradicionales

2.1. Programación extrema (Extreme Programming, *XP*)

La metodología XP es una metodología ágil dedicada al desarrollo de software. Es de las más exitosas de la actualidad, y está dedicada al desarrollo de proyectos a corto plazo con un tamaño de equipo pequeño. La metodología consiste en una **programación rápida** (o *extrema*), cuya particularidad es tener como parte del equipo al **usuario final**, lo que se considera uno de los requisitos para llegar al éxito en el proyecto.

La programación extrema cuenta con las siguientes características:

- **Desarrollo iterativo e incremental**: Pequeñas mejoras que van sucediendo unas tras otras.

- **Pruebas unitarias continuas**: Frecuente repetidas y automatizadas, incluyendo pruebas de regresión (para descubrir errores en la funcionalidad - *bug*s -). Es aconsejable escribir el código de la prueba antes de la codificación.

- **Programación en grupo**: se recomienda que las tareas de desarrollo sean realizadas por dos o más personas en un mismo puesto. De esta manera, el código es revisado y discutido mientras se escribe, lo que asegura un incremento de calidad que es más importante que la posible pérdida por improductividad.

- **Corrección de errores**: Antes de añadir nuevas funcionalidades, es importante que todos los errores queden resueltos. Esto se soluciona aumentando el número de entregables.

- **Integración del equipo de programación con el cliente/usuario**: Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

- **Refactorización del código**: Consiste en reescribir ciertas partes del código para mejorar su legibilidad y mantenibilidad, sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

- **Propiedad del código compartida**: En lugar de dividir las responsabilidades del desarrollo de cada módulo en grupos de trabajo distintos, esta metodología promueve que cualquier miembro del equipo pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.

- **Simplicidad en el código**: Es la mejor manera de que las cosas funcionen. La XP afirma que es partir de un desarrollo 'simple' y tener que realizar trabajo extra si se necesita alguna mejora, que utilizar un software y un código complejo, en el cual realizar cambios sea realmente complicado.

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más

simple es el sistema de trabajo, más sencillas serán las comunicaciones en el equipo de trabajo, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

2.1.1. Roles

Aunque la cantidad de roles ha evolucionado desde su concepción inicial, y varía en función del producto o proyecto que se desarrolla, la propuesta original de la metodología XP incluía los siguientes [9]:

- **Cliente / Usuario**:

  Es el encargado de redactar las Historias de Usuario (***User Stories***), que incluye todas las funcionalidades deseadas. También debe participar en las pruebas funcionales del producto para validar su implementación. Categoriza las tareas por dificultad y/o importancia, decidiendo cuáles deben ser desarrolladas en cada ciclo de trabajo. Solo hay un cliente o usuario por proyecto dentro del equipo que, en caso de necesidad, deberá actuar en representación de tantas personas como se puedan ver afectadas por la solución.

- ***Coach***:

  El *Coach* ("entrenador") es el responsable de que la metodología XP se esté implementando correctamente. Necesita conocer los procedimientos XP en profundidad para conducir al equipo durante el desarrollo del proyecto.

- ***Big Boss***:

  El *Big Boss* (Gestor) tiene como principal función ser el enlace entre el cliente y el equipo de desarrollo. Realiza principalmente tareas de coordinación.

- ***Programador***:

  Los programadores son los principales desarrolladores del producto. Escriben e implementan el código del sistema. Deben estar en constante comunicación con los demás miembros del equipo, ya que de su labor depende el desempeño del resto de actividades.

- ***Tester***:

  El *Tester* (encargado de pruebas), realiza las pruebas funcionales en conjunto con el cliente de manera regular. Suele ser una única persona, que debe estar en continua comunicación con el equipo de programación para solucionar posibles errores de código.

- ***Tracker***:

El *Tracker* (encargado de seguimiento) debe evaluar de manera continua el estado del proyecto, comparándolo con las estimaciones realizadas tanto al inicio como durante el transcurso del mismo. Reconsidera objetivos, plazos y recursos en cada ciclo de trabajo para mejorar las estimaciones.

Todos estos roles están presentes en prácticamente cualquier proyecto en el que se emplea esta metodología, aunque es frecuente que una misma persona asuma varios de ellos de forma simultánea. Es habitual que varias de las funciones de *Tracker*, *Coach* y *Big Boss* recaigan sobre un mismo integrante del equipo.

2.1.2. Prácticas XP

Al contrario que otras metodologías ágiles, la metodología XP no impone un ciclo de desarrollo del proyecto. En su concepción, se prefijaron seis fases para un ciclo ideal: Exploración, Planificación, Iteraciones, Producción, Mantenimiento y Muerte del proyecto.

No obstante, la metodología ha evolucionado de tal manera que los proyectos no tienen por qué seguir siempre estas directrices, sino que se desarrollan siguiendo **doce principios** básicos agrupados en tres categorías (***Fig 3.***) [10]:

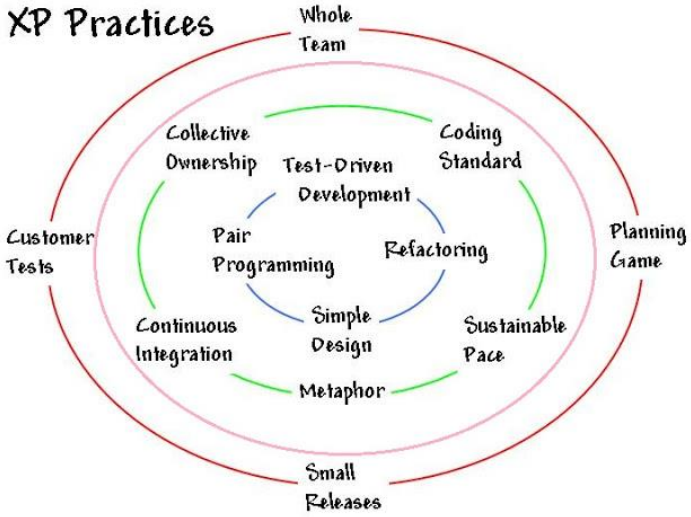


Fig 3 Prácticas XP [11]

1. Retroalimentación a escala fina

El principal objetivo es evaluar unidades de código continuamente, es decir, el fragmento de código más pequeño que puede aislarse de forma lógica en un sistema, para obtener respuesta fiable de su correcto funcionamiento. Para conseguir realizar esto de forma sistemática, se hace necesario encontrar las condiciones de trabajo óptimas.

1.1. El principio de pruebas

Se tiene que establecer un periodo de pruebas de aceptación del programa, también conocido como periodo *de caja negra*, donde se definirán las entradas al

sistema y los resultados esperados de estas. Estas pruebas deben ser, cada vez, más automatizadas, con el fin de poder realizar varias simulaciones del sistema en funcionamiento [12].

Para hacer estas simulaciones automatizadas, se pueden utilizar ambientes de prueba (*Unit Testing Frameworks*), a través de los cuales obtenemos resultados de fragmentos unitarios de código.

1.2. Juego de planificación

En esta fase, el usuario final / cliente tendrá que describir sus necesidades, definiendo las actividades que realizará el sistema. Se creará un documento de referencia denominado *User Stories* (Historias del Usuario), que albergará un número de 'historias' en función de la complejidad del problema que se desea resolver. A raíz de estas, se podrán definir los tiempos de entrega de la aplicación para recibir retroalimentación por parte del usuario.

Son muy importantes y tienen que ser una constante las reuniones periódicas durante esta fase, que pueden llegar a ser, incluso, a diario, y necesitarán contar con todo el equipo de desarrollo para identificar problemas, proponer soluciones, y señalar aquellos puntos a los que se les debe dar más importancia por su dificultad o por su criticidad.

1.3. El cliente, en el sitio

Al cliente se le dará poder para determinar los requerimientos, definir funcionalidades y señalar las prioridades, así como será necesarias su ayuda para resolver las dudas del equipo de programación. Esta fuerte interacción –cara a cara– entre cliente y programador, disminuye el tiempo de comunicación y la cantidad de documentación, así como los elevados costes que conllevan su creación y mantenimiento.

El cliente (o representante del cliente) estará con el equipo de trabajo durante la realización completa del proyecto.

1.4. Programación en parejas (*Pair Programming*)

Este es, probablemente, el principio más radical de la metodología XP, y en el que la mayoría de gerentes de desarrollo pone sus dudas. Requiere que todos los programadores escriban su código en parejas o, incluso, en grupos más numerosos, compartiendo una sola máquina.

De acuerdo con algunos experimentos con metodología XP, este principio puede producir mejores aplicaciones de manera consistente, a iguales o menores costes. Aunque el *Pair Programming* puede no ser útil en todas las situaciones, los resultados en proyectos que han seguido la metodología XP han demostrado un gran éxito.

2. Proceso continuo, en lugar de proceso por lotes

En muchas ocasiones, nos encontramos que la metodología consiste en validar paquetes de trabajo independientes, que quedan inamovibles una vez terminados. La metodología XP pretende evitar el fraccionamiento radical de las tareas de un proyecto, integrando todo el proceso en un sistema dinámico, que está sujeto a continuos cambios.

2.1. Integración continua

Permite al equipo hacer un rápido progreso, implementando las nuevas características del software. En lugar de crear versiones estables en función de un cronograma preestablecido, los equipos de programadores pueden reunir su código y reconstruir el sistema varias veces, incluso, al día. Esto reduce los problemas de integración comunes en proyectos largos.

2.2. Refactorización

Permite a los equipos de programación mejorar el diseño del sistema a través de todo el proceso de desarrollo. Los programadores evalúan continuamente el diseño y recodifican lo necesario, a fin de mantener un sistema enfocado a incrementar el valor del programa mediante la minimización del código duplicado o ineficiente [13].

2.3. Entregas pequeñas

A través de un sistema sencillo en producción, que se actualiza de forma rápida y constante, es posible que el verdadero valor del producto sea evaluado de forma constante en un ambiente realista. Estas entregas no deben superar las 2-3 semanas de plazo, como máximo.

3. Entendimiento compartido

En muchas ocasiones, nos encontramos que la metodología consiste en validar paquetes de trabajo independientes, que quedan inamovibles una vez terminados. La metodología XP pretende evitar el fraccionamiento radical de las tareas de un proyecto, integrando todo el proceso en un sistema dinámico, que está sujeto a continuos cambios.

3.1. Diseño simple (*Simple Design*)

Se basa en la filosofía de que el Software más valioso es entregado por el programa más sencillo que cumpla todos los requisitos funcionales. *Simple Design* se enfoca en proporcionar un sistema que cubra tan sólo las necesidades inmediatas del cliente, sin abarcar ni más ni menos de lo necesario.

Este proceso permite eliminar redundancias y rejuvenecer los diseños obsoletos de forma sencilla.

3.2. Estándares de programación

Uno de los principios de la metodología XP es la comunicación fluida entre los miembros del equipo. Para que esto ocurra, es necesario que todos los programadores sean capaces de interpretar todo el código, por lo que es indispensable que se utilicen estándares de programación.

No obstante, la metodología XP no establece cuál o cuáles son los estándares a seguir, ya que muchas veces depende del lenguaje de programación utilizado, o la formación particular de los miembros del equipo.

3.3. Metáfora

Desarrollada por el equipo de programación al inicio del proyecto, se ha de definir una 'historia' de cómo funciona el diseño completo. La metodología XP se basa en historias, que son breves descripciones de las funcionalidades que ha de poder ejecutar un Software, que sustituyen a los tradicionales diagramas.

La metáfora describe la visión evolutiva del proyecto, y define el alcance y propósito del programa final.

3.4. Propiedad colectiva del código

El Software está desarrollado a través de un código con propiedad compartida. Nadie es el propietario de ninguna de sus partes, y todos los miembros del equipo pueden participar en todos los desarrollos. Esto difiere ampliamente de las metodologías tradicionales, en las cuales un grupo de programación o, incluso, un único programador, es desarrollador y propietario único de un conjunto de código.

Los defensores de la metodología XP argumentan que mientras haya más programadores que participen del mismo bloque de código, menor es el riesgo al que se expone.

3.5. 40 horas semanales

Uno de los objetivos de esta metodología es sacar el máximo rendimiento a cada uno de los miembros del equipo, ajustándose a su jornada laboral. Admite que, en momentos puntuales, esta pueda verse excedida, pero si es algo que ocurre de manera recurrente, es un síntoma inequívoco de que algo no se está haciendo correctamente. Además, el necesitar continuamente de horas extra puede desmotivar al grupo.

Las prácticas señaladas no son independientes unas de otras, si no que su mayor rendimiento lo obtenemos cuando se alimentan y refuerzan entre sí (***Fig 4.***). El logro más importante de esta metodología no es el diseñar estas prácticas, ya que prácticamente todas ya existían con anterioridad, sino agruparlas e integrarlas de forma efectiva.

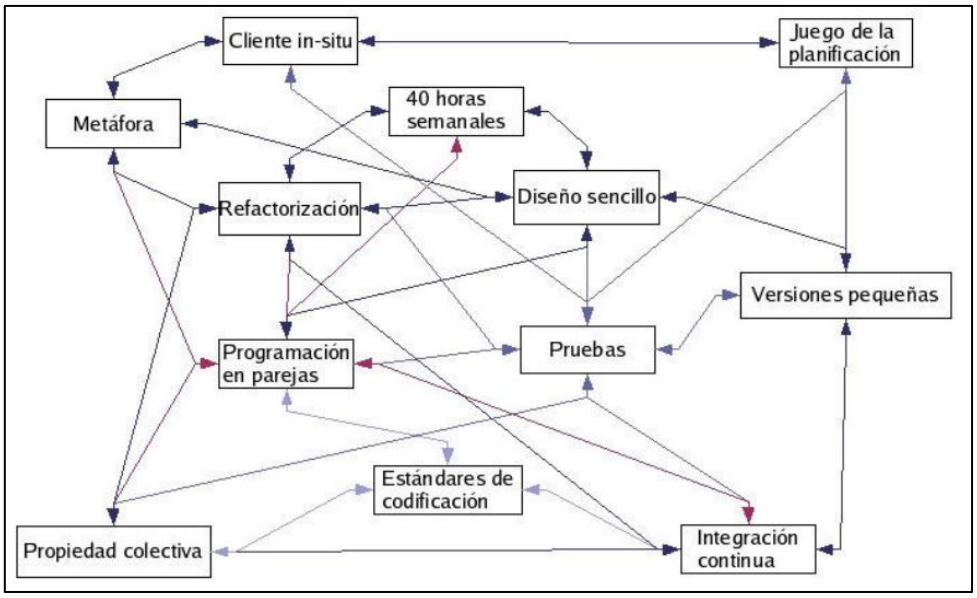


Fig 4 Las prácticas XP se refuerzan entre sí [9]

2.2. SCRUM

SCRUM es una metodología de trabajo para el desarrollo y mantenimiento de productos de cualquier tipo de complejidad. Es una de las metodologías ágiles más representativas y utilizadas en la actualidad. Su enfoque inicial fue para utilizarlo en proyectos de desarrollo de Software, aunque sus principios son fácilmente adaptables en cualquier contexto o área de conocimiento [14].

2.2.1. Roles

El ciclo de vida del SCRUM se compone de tres roles principales [15]:

- **Dueño del producto (*Product Owner*)**:

  Es la representación del cliente dentro del equipo de trabajo. Su principal responsabilidad es expresar claramente la necesidad del cliente dentro del *Product Backlog*.

  No tiene que ser necesariamente el propio cliente, ni alguien externo que defienda sus intereses. Lo más habitual es que sea un miembro del equipo, que actúe como ingeniero de requisitos, cuya misión principal es acceder a toda la información y conocer de primera mano todas las necesidades del cliente, para después transmitirlo al resto de miembros del equipo [16].

- **Equipo de desarrollo (*Development Team*)**:

  El equipo de desarrollo se compone del personal responsable de cumplir tareas a través de *Sprints*. Debe estar compuesto por profesionales cualificados, que sean capaces de dar solución a las necesidades del *Product Owner*. Es muy importante que sea un equipo autogestionado y organizado.

  Aunque dentro del *Development Team* las funciones son muy variadas, a los miembros del equipo se los conoce como desarrolladores. Esto se realiza para simplificar los roles y evitar jerarquías dentro del equipo, por lo que programadores, *testers*, analistas… tendrán la misma importancia.

- **SCRUM Master**:

  Es el responsable de asegurar que la metodología SCRUM se está aplicando correctamente durante el proyecto, asegurando que el equipo trabaja ajustándose a la teoría, prácticas y reglas correspondientes. Toma la función de moderador, por lo que no es el responsable de dar órdenes ni de conocer el funcionamiento del software. Su función principal es ayudar al *Development Team* a comprender las necesidades del cliente, expresadas a través del *Product Owner*, y a organizar el proyecto siguiendo los principios de la metodología SCRUM.

2.2.2. Fases y procesos

Se puede dividir el ciclo de vida de un desarrollo de proyecto mediante metodología SCRUM en cinco fases [17]:

1. INICIACIÓN

En primer lugar, el *Product Owner* va a definir un documento con todas las necesidades que necesita cubrir el cliente. Ese documento se denomina ***Product Backlog***. Debe contener todas las ideas, necesidades y requisitos que darán cumplimento a solicitud formal del cliente.

Una vez se ha conoce correctamente qué se debe hacer, es necesario identificar al *Scrum Master* y a los interesados del proyecto, así como formar el equipo con los desarrolladores que participarán en él, formando el *Development Team.*

2. PLANIFICACIÓN Y ESTIMACIÓN

El *Product Owner* tiene que manifestar las necesidades del cliente al equipo de desarrollo y al *Scrum Master*. Esto se realiza en una reunión llamada ***Sprint Planning Meeting***. En ella, se va a llegar a una primera aproximación a la solución del producto final deseado.

El resultado de esta reunión es una lista detallada de funcionalidades concretas, denominada ***Sprint Backlog*** (que se extrae del *Product Backlog*). Consiste en un conjunto tareas que dan cumplimiento a los requisitos del cliente, cada una de las cuales ha de completarse en un ciclo cuya duración es establecida por el *Scrum Master*.

3. IMPLEMENTACIÓN

El proyecto se desarrolla mediante la realización de tareas que han de completarse en periodos de tiempo constantes, denominados ***Sprints*** [14]. Los *Sprints* son el centro de la metodología SCRUM. Corresponden al proceso de desarrollo de las necesidades del cliente divididas en módulos funcionales, de tal manera que al final de cada *Sprint* se obtiene un producto incremental. El tiempo correspondiente a cada *Sprint* es establecido por el *Scrum Master*, pero normalmente toma entre 1 y 4 semanas, en función de la dificultad del desarrollo.

En esta fase intervienen tanto el *Scrum Master* como el *Development Team*. Este último toma un papel protagonista, ya que es el encargado de desarrollar las funcionalidades descritas en el *Product Backlog*, mientras el *Scrum Master* adopta una posición de apoyo. No obstante, en constantes ocasiones, el *Scrum Master* puede formar parte del equipo de desarrollo.

Una de las actividades más representativas de la metodología SCRUM son los *Daily SCRUM*. Consisten en reuniones periódicas (a ser posible, diarias) del equipo de

desarrollo con el *Scrum Master* para realizar un seguimiento constante del avance del proyecto. En ella se tratan cuatro cuestiones, que son siempre una constante:

- ¿Qué se ha hecho desde la anterior reunión?
- ¿Qué se va a hacer hasta la siguiente reunión?
- ¿Qué se tiene previsto hacer después de la siguiente reunión?
- ¿Qué problemas se han encontrado?

En estas reuniones debe intervenir cada uno de los miembros del equipo de desarrollo. Están pensadas para ser de muy corta duración (inferior a 15min), y que diariamente se pueda tener un contexto global del estado actualizado del *Sprint*. Esto facilita el seguimiento del proyecto y, por tanto, la toma de decisiones.

La principal herramienta que se utiliza para el seguimiento del proyecto consiste en un **tablero *Kanban***. Este tablero consta de un número variable de columnas (habitualmente entre 3 y 5), a través del cual se observa el avance de las tareas planificadas durante un *Sprint*. Cada tarea se separa en un trozo de papel indenpendiente que pasa, como mínimo, por tres estados (correspondientes a las columnas del tablero, ***Fig 5.***): pendiente, en proceso y finalizado. Esto agiliza bastante el proceso de entendimiento del *Sprint* en el que se está trabajando y así, además, se favorece la transparencia, de tal manera que cada miembro del equipo puede conocer el estado de avance de las tareas del resto.

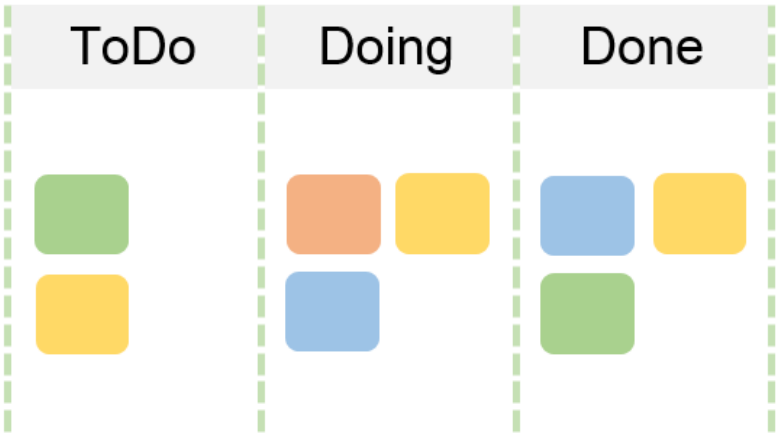


Fig 5 Ejemplo de tablero Kanban simplificado [18]

El éxito de un *Sprint* depende de todo el equipo de desarrollo. Se busca que todas las personas tengan tareas asignadas, de las cuales serán responsables. El objetivo es garantizar el cumplimento de todos los requerimientos del *Sprint* en los tiempos definidos.

4. REVISIÓN Y RETROSPECTIVA

Al finalizar un *Sprint*, se realiza una nueva reunión, llamada ***Sprint Review***. En ella se verán involucrados todos los miembros del proyecto (incluso el *Product Owner*), y se deberán verificar el cumplimiento de los objetivos del *Sprint* en cuestión, para así garantizar la entrega del producto al cliente final.

Cada vez que finaliza un *Sprint*, es necesario tener un producto funcional. Este producto funcional debe ser entregado al cliente para que pueda interactuar con él, de tal manera que pueda comprobar el avance del proyecto de manera incremental.

Si todos los objetivos han sido alcanzados y el producto es entregado, es cuando se realiza la reunión de Retrospectiva al *Sprint*. En ella se analizan los resultados del *Sprint* anterior para reflejar incidencias, problemas o mejoras que puedan afectar al siguiente. De este modo, se inicia un nuevo *Sprint*, lo que requiere volver al *Product Backlog* para extraer necesidades del cliente que el desarrollo actual del proyecto no cubre. Esto se repite hasta que acabe el último *Sprint* y, por tanto, el proyecto finalice y el producto quede terminado.

5. LANZAMIENTO

Para que el proyecto se pueda dar por concluido, el cliente debe recibir tanto su producto completo, como el resto de entregables que se hayan establecido en el inicio del proyecto. Superada esta fase, se vuelve a realizar una reunión de Retrospectiva con todos los miembros del proyecto, de tal manera que queden reflejados todos aquellos aspectos relevantes del desarrollo que puedan ser de utilidad para proyectos futuros.

2.3. LEAN DEVELOPMENT

LEAN Manufacturing es un sistema de producción basado en la eliminación de todos los recursos innecesarios en un proceso, minimizando la cantidad de desperdicios. Es una estrategia operativa de excelencia creada en la multinacional automovilística *Toyota*, y sus ideas se han extendido por todo el sector industrial a nivel mundial.

Los principios del *LEAN Manufacturing* son tan claros que pueden reformularse para ser aplicados en el desarrollo de Software, dando lugar a una metodología Software conocida como *LEAN Development* [19]. Con su implementación, se persigue optimizar el ciclo de desarrollo desde la primera solicitud del cliente hasta la entrega del producto y lograr mejoras sustanciales en la ejecución del proyecto. El *LEAN Development* representa un proceso de desarrollo de Software altamente efectivo que, a pesar de partir de una base ideal, es totalmente práctico.

Las ideas de la metodología *LEAN*, aplicadas al desarrollo de Software, hacen perseguir los siguientes objetivos:

- Enfoque hacia la entrega de **valor** y mejora en la **calidad**. El punto fuerte de la creación de valor consiste en la reducción de la cantidad de recursos innecesarios.

- Mejora en la toma de **decisiones** en base a principios *LEAN*. Este objetivo va de la mano de un término esencial en esta metodología: ***Just in Time*** (“en el momento preciso”). Las mejores decisiones han de ser tomadas en los momentos más importantes, optimizando el desarrollo y mitigando riesgos.

- Incremento de la **productividad**, a través de la eliminación de los desperdicios ligados al desarrollo de Software. Con ello, se mejorarán los tiempos de entrega y se desarrollarán productos de calidad desde el primer momento.

La base de la metodología *LEAN* es identificar el desperdicio que se genera durante el desarrollo de un proyecto, centrándose en actitudes y creencias consideradas aceptables, que en realidad consumen recursos innecesarios [20].

2.3.1. Principios LEAN

Para utilizar metodologías ágiles en el desarrollo de Software, es necesario tener un conocimiento avanzado de las disciplinas más técnicas. Sin estos conocimientos previos, los principios de metodologías como el *LEAN Development* son mucho menos efectivos para el desarrollo óptimo de proyectos [21]. El *LEAN Development* establece los siguientes principios (***Fig. 6***):

1. *Eliminate Waste*

Waste (Basura) representa todo aquello que no aporta valor para el usuario final. Son todas aquellas acciones que implican un mal consumo de los recursos del equipo de desarrollo (incluído el tiempo), como desarrollar soluciones innecesariamente complejas,

rehacer varias veces el mismo trabajo, añadir funcionalidades que el cliente no ha requerido…

Todas estas acciones no aportan valor al usuario, por lo tanto son desperdicios y es necesario eliminarlas.

2. *Amplify Learning*

El propio proceso de desarrollo ya es, de por sí, un aprendizaje continuo. Las ideas adquiridas pueden ser plasmadas en el código o en archivos de texto simples, haciendo prescindible la elaboración de documentos que, a medio – largo plazo, pueden hacerse inútiles.

El proceso de desarrollo debe realizarse en ciclos cortos con reuniones frecuentes, que también favorecerán el aprendizaje que se persigue y servirán para identificar las necesidades del cliente y del equipo.

3. *Decide as late as posible*

Durante el desarrollo de un proyecto hay muchas decisiones críticas que, aunque parezca incoherente, es necesario retrasar hasta tan tarde como sea posible. Principalmente, estas decisiones corresponden al entorno de implementación del desarrollo. La solución se debe ir probando y evaluando en ciclos cortos, adquiriendo versatilidad suficiente como para que algunas decisiones puedan retrasarse en el transcurso del proyecto sin afectar al coste, calidad y plazos del mismo.

4. *Deliver as fast as posible*

"El primero en llegar, triunfa" es una máxima muy representativa del paradigma actual. Actualmente, el *feedback* que se recibe a lo largo de un desarrollo por parte del usuario es continuo y ágil, por lo que es necesario ser activo y entregar el producto tan rápido como sea posible para satisfacer al cliente/usuario, así como para mantener distancia con la competencia (si la hubiera).

5. *Empower the team*

Un equipo no debe ser visto como un "conjunto de recursos" que se asocian a un servicio. No debe ser valorado en número de horas ni en coste económico. Idealmente, el equipo de desarrollo debería asumir su propia gestión, tener unas tareas asumibles y realistas, y un contacto directo con el cliente.

6. *Build integrity in*

El proyecto, tanto de cara al cliente como de cara al usuario, debe ofrecer una imagen robusta. La integridad se alcanza uniendo flexibilidad con un producto que sea mantenible a largo plazo, realizando continuas iteraciones en ciclos cortos, para mejorar funcionalidades a partir del *feedback* del usuario.

7. *See the whole*

La mayoría de proyectos actuales son de gran complejidad. Para gestionarlos correctamente es necesario dividirlos y descomponerlos en tareas de menores. Se hace necesario reducir el tamaño del equipo para poder implicar a todas las partes. Los proyectos pueden estar, incluso, en la mano de varios equipos de forma simultánea.

Fig 6 Principios LEAN Development

3. CASO PRÁCTICO

El desarrollo de Software se caracteriza por ser un proceso dinámico y complejo. Al contrario que ocurre con proyectos de otros sectores comerciales, los requisitos técnicos del producto final, así como las necesidades del cliente, varían continuamente a lo largo de la ejecución del proyecto. En esta situación, conocer exactamente los recursos (materiales y humanos) necesarios y estimar correctamente los plazos son tareas difíciles de planificar al inicio del proyecto, y de gestionar durante el transcurso del mismo.

Antes de implementar en fases sucesivas los conocimientos adquiridos en metodologías ágiles en el desarrollo de Software, es necesario conocer el paradigma en el cual se encuentra el equipo de desarrollo cuyo rendimiento se pretende mejorar.

3.1. Paradigma inicial

Al inicio del caso práctico que se expone, el equipo de proyecto se encuentra en un entorno de desarrollo de proyectos de ingeniería ajenos a la ingeniería de Software, principalmente de instalaciones, obra civil, edificación. En ellos, se utilizan metodologías tradicionales para la ejecución de los proyectos.

El equipo de desarrollo de Software, formado 4 años antes de la fase inicial del caso, es posterior a la formación del resto de las divisiones de la compañía, por lo que ha heredado la metodología de trabajo, y la ha adaptado con el paso del tiempo para enfocarla a la actividad específica que realiza.

3.1.1. Desarrollo software en cascada

Como consecuencia de lo anterior, los proyectos se ejecutan siguiendo un modelo secuencial, también conocido como **desarrollo en cascada**. De forma genérica, este proceso se puede dividir en cinco fases (***Fig 7.***):

1. **Definición de requisitos**: Identificación de las necesidades del cliente y de los requisitos que debe cumplir el producto.

2. **Análisis y diseño del Software**: Se estudia la viabilidad técnica del desarrollo y se establece una planificación de actividades a ejecutar.

3. **Codificación**: Programación del software en sí. Corresponde a la fase más extensa del proyecto, en la cual también se realizan pruebas de funcionamiento parciales.

4. **Integración y pruebas del sistema**: Se integran en un único software los desarrollos independientes y se realizan las pruebas funcionales del producto completo.

5. **Implantación y mantenimiento**: Se entrega el producto al cliente. No obstante, el software habitualmente necesita cambios después de que esto ocurra, por lo que se vuelven a aplicar cada una de las cinco fases con cada modificación que se realice.

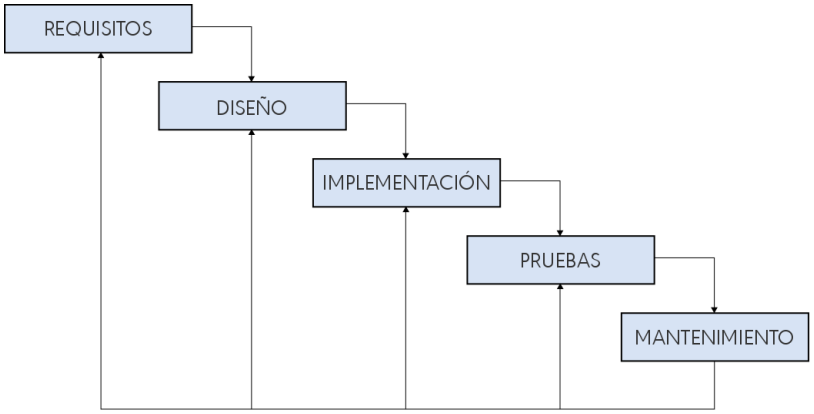


Fig 7 Esquema de metodología en cascada (Fuente propia)

3.1.2. Equipo de trabajo

Otro factor importante para comprender la situación inicial es conocer los perfiles profesionales de los integrantes del equipo de trabajo, y los recursos materiales de los que se dispone. Se identifican los siguientes roles (***Fig 8.***):

- **Gerente de Negocio**: Ingeniero Técnico Informático. Realiza la coordinación técnica, la gestión administrativa y las labores comerciales del equipo.
- **Programador**: Técnico superior en programación en múltiples lenguajes. Realiza los desarrollos de codificación y las pruebas del sistema.
- **Diseñador 3D**: Técnico superior en diseño 3D. Realiza modelados 3D en los proyectos que lo requieren, programa animaciones y diseña la interfaz gráfica del producto final.

Cada integrante cuenta con un ordenador personal de altas prestaciones, y se han adquirido todas las licencias necesarias para los desarrollos, por lo que en este aspecto no se encuentra ninguna limitación. El equipo cuenta con varios programadores (habitualmente, entre 5 y 7) y varios diseñadores 3D (entre 2 y 3). Ninguno de los miembros tiene conocimientos específicos de Dirección de Proyectos, ni experiencia en el uso de metodologías ágiles.

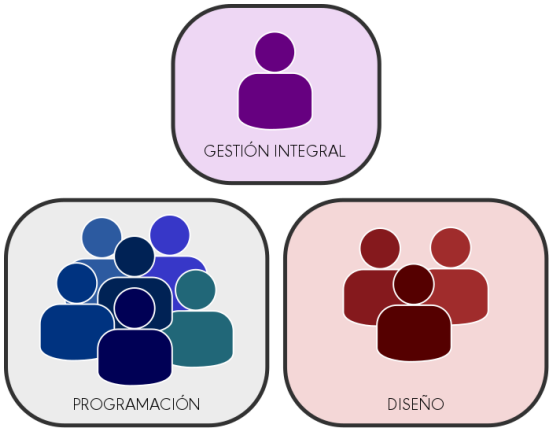


Fig 8 Organigrama del equipo de desarrollo Software

3.1.3. Análisis de situación

Se realiza un análisis de la situación inicial, del cual se extrae la siguiente información:

1. GESTIÓN DE LOS PROYECTOS

- Alcance: Por norma general, existe una gran diferencia entre la idea inicial del cliente y el producto final. Habitualmente, por limitaciones de la tecnología, no se pueden cumplir todos los requisitos iniciales tal y como se solicitan. Durante el transcurso del proyecto, se tiende a modificar las especificaciones del producto final desarrollando funcionalidades alternativas o sustitutivas sin la aprobación del cliente.

- Tiempo: Al no existir entregas parciales durante el transcurso del proyecto, la productividad del equipo se reduce en las etapas iniciales del desarrollo. Además, durante el transcurso del mismo se encuentran funcionalidades a desarrollar que suponen cuellos de botella no esperados. Todo ello conduce que en la fase final del proyecto sea necesario un sobreesfuerzo del equipo que, habitualmente, implica retrasos significativos en la entrega del producto final.

- Coste: Se realizan desarrollos poco complejos, aunque extensos en duración. El mercado hace que las aplicaciones informáticas básicas tengan un coste muy bajo, por lo que parte del equipo de desarrollo es personal en prácticas (es decir, sin experiencia). La falta de experiencia equivale a falta de formación para realizar desarrollos avanzados, lo que, nuevamente, conlleva a que los productos sean poco complejos. Todo esto conduce a que los proyectos obtengan una rentabilidad muy baja.

2. EQUIPO DE TRABAJO

A excepción del Gerente de Negocio, todos los miembros del equipo de trabajo tienen un perfil muy técnico, por lo que es una sola persona la que realiza toda la labor de gestión (a todos los niveles), además de realizar pruebas de calidad y ser el único contacto del equipo con el cliente.

El código se comparte en un servidor común, pero no es frecuente que dos programadores colaboren en un mismo desarrollo, sino que se tiende a diferenciar módulos y funcionalidades de un software, que se codifican de manera independiente. Al final del proyecto, esto implica muchos problemas de integración, y de frecuente re-trabajo de desarrollo.

3. CONTACTO CON EL CLIENTE

El contacto con el cliente se realiza mayoritariamente por correo electrónico con el Gerente del equipo. Es constante durante la definición de requisitos y en la fase

final del proyecto, pero durante el transcurso del mismo se da de forma esporádica y lejana, y no se le notifica apropiadamente del avance real del desarrollo.

En las etapas de cierre de proyecto, cuando el contacto es más directo, el cliente se encuentra con su producto casi terminado. Al no haber seguido el transcurso del desarrollo, pueden existir diferencias importantes entre las expectativas y la realidad, lo que acaba supone modificaciones inesperadas, cambios en la planificación, extensión en los plazos y, en definitiva, sobrecostes de desarrollo.

4. ENTORNO

El ambiente de trabajo es muy importante en cualquier equipo. Hay una única persona que realiza todas las labores de gestión, lo que conlleva a una sobrecarga de trabajo y el posible descontento del trabajador. Esta sobreasignación de tareas implica que algunas labores administrativas se deriven al equipo de desarrollo, que no tiene la formación necesaria para ejecutarlas correctamente. Además, solo 4 de los programadores están contratados, mientras suele haber entre 2 y 3 programadores en prácticas.

Todo ello, unido a la falta de planificación, y a los sobreesfuezos que se realizan en las etapas finales de los proyectos, genera inestabilidad en el equipo de desarrollo, que puede verse reflejada en el rendimiento individual de cada trabajador.

3.2. Implementación de metodologías ágiles en el desarrollo software

Como consecuencia de todo lo expuesto anteriormente, se buscan soluciones que, de manera simultánea, aumenten la productividad del equipo y mejoren el ambiente general de trabajo. Como medida inicial, se incorpora una persona con conocimientos en **Dirección de Proyectos** que asume la coordinación técnica del equipo, y en decisión conjunta con la dirección de la compañía y el resto de miembros, se decide implementar prácticas de metodologías ágiles (***Fig 9.***).

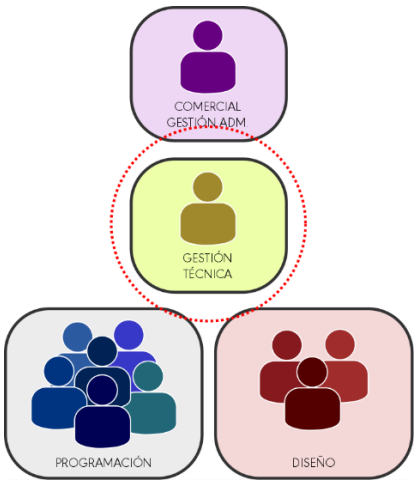
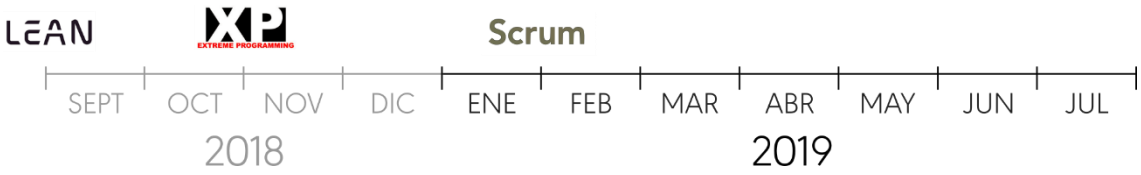


Fig 9 Organigrama modificado del equipo de desarrollo Software

A lo largo de 11 meses se han implementado diferentes metodologías ágiles en el desarrollo de proyectos de Software informático. Se ha seguido la siguiente línea temporal:



3.2.1. Prácticas LEAN

Alterar por completo el ciclo de vida de los proyectos podría implicar un cambio demasiado radical, por lo que la primera medida tomada es aplicar los principios LEAN en el desarrollo de los productos.

En primer lugar, el equipo de desarrollo recibe formación en prácticas LEAN asociadas al desarrollo de Software. No obstante, por simples y evidentes que parezcan, no es posible implantar todos los conocimientos adquiridos desde un inicio, sino que se realiza de manera progresiva a lo largo de dos meses.

Durante este periodo, se implantan las siguientes medidas:

- **Reuniones del equipo completo dos veces a la semana**

  Una primera reunión (habitualmente, en lunes) para extraer los resultados de la semana anterior, analizar los problemas encontrados y programar la semana entrante. Cada desarrollador debe exponer su actividad de la semana anterior y se le da mucha importancia a que el resto de compañeros conozcan en qué consiste su desarrollo, especialmente los implicados en el mismo proyecto. Duración estimada: 25-30min.

  Se realiza otra una reunión (habitualmente, en jueves) para el seguimiento de la actividad. El principal objetivo es solucionar en común los problemas que hayan podido surgir en desarrollos individuales, pero también es útil para hacer un balance de expectativas – realidad. Duración estimada: 10-15min

- **Designación de un responsable de proyecto**

  Uno de los miembros del equipo de cada proyecto asume la responsabilidad del desarrollo del mismo, lo que no implica que deba realizar tareas de gestión ni coordinación.

  Conlleva que debe conocer en todo momento el estado de los posibles desarrollos independientes que puedan estar realizándose de forma simultánea, y asegurar que todos pueden integrarse de manera directa. Además, pasa a ser el responsable de las pruebas unitarias y finales del producto, convirtiéndose en el principal responsable de calidad del proyecto.

- **Contacto directo entre cliente y equipo de desarrollo**

  El contacto entre cliente y equipo de desarrollo cambia de forma radical. Desde el inicio de cada proyecto, se programan reuniones periódicas presenciales / semipresenciales (vía videollamada) en las que siempre está presente, al menos, el responsable de cada proyecto.

  Además, las consultas técnicas rutinarias, que se dan vía correo electrónico, las realiza también el responsable del proyecto. Se asegura, de esta manera, un vínculo directo entre el equipo de desarrollo y el cliente.

- **Reducción de la documentación**

  Se establecen nuevas directrices sobre lo que debe quedar documentado en cada proyecto, que variará en función de la complejidad de cada desarrollo, la extensión en el tiempo y el volumen de facturación que implican

  Para los proyectos que necesitan un desarrollo sencillo o en un plazo pequeño, no se genera más que un plan de proyecto reducido, en el que se detallan los requisitos del cliente y las funcionalidades a desarrollas; las actas de cada reunión presencial / semipresencial con el cliente; y los certificados de buena ejecución, que dan por concluido el proyecto.

  Conforme aumenta la entidad del desarrollo, se hace necesario documentar algunos aspectos del transcurso del proyecto. No obstante, solo se documentan cambios en los requisitos y funcionalidades, y los certificados de validación de hitos intermedios.

  El responsable del proyecto, así como el resto de desarrolladores implicados, no deben generar documentación alguna, más que el propio código. No obstante, este debe quedar correctamente comentado, por lo que se alcanza un acuerdo sobre qué debe comentarse y cómo, con el fin de que todos los programadores puedan entender todos los desarrollos.

- **Comunicación en el equipo de desarrollo**

  La designación de un miembro del equipo de desarrollo como responsable del proyecto implica que exista un hilo conductor entre todos los programadores implicados, que es realmente consciente de los posibles desarrollos independientes que se estén produciendo.

  Es frecuente que, cuando esto existe, dos desarrollos diferentes compartan ciertas funcionalidades. La falta de comunicación puede hacer que estas se desarrollen de manera independiente en repetidas ocasiones. Si no se hace un uso correcto del código ya escrito, y no se reutiliza cuando es posible, la productividad del equipo baja radicalmente.

El responsable de cada proyecto es el que asume a tarea de facilitar la comunicación entre desarrolladores para minimizar estos re-trabajos. Cuando esto se consigue, no sólo aumenta la productividad, sino que el clima de trabajo se ve favorecido.

Uno de los principales problemas que supone, en un principio, la implantación de los principios LEAN, es la adaptación de los proyectos en curso a las nuevas prácticas establecidas. En un primer momento, implica invertir una gran cantidad de tiempo en adaptar la documentación, el código y la gestión de recursos humanos.

A pesar de que la adaptación sea progresiva, no deja de ser muy rápida. En un periodo de tiempo muy corto, ha variado el funcionamiento de gran parte de los mecanismos del equipo. Además, cada integrante tiene un nuevo rol que, incluso, puede ser diferente en cada proyecto en el que se ve implicado.

En este periodo de tiempo, el equipo se ve envuelto en varios proyectos de forma simultánea. Algunas de las actividades que se desarrollan son las siguientes:

1. PÁGINAS WEB

Representa la actividad más habitual durante este periodo. Los proyectos son sencillos y cortos, aunque no aportan gran rentabilidad debido a su bajo coste (***Fig 10***).

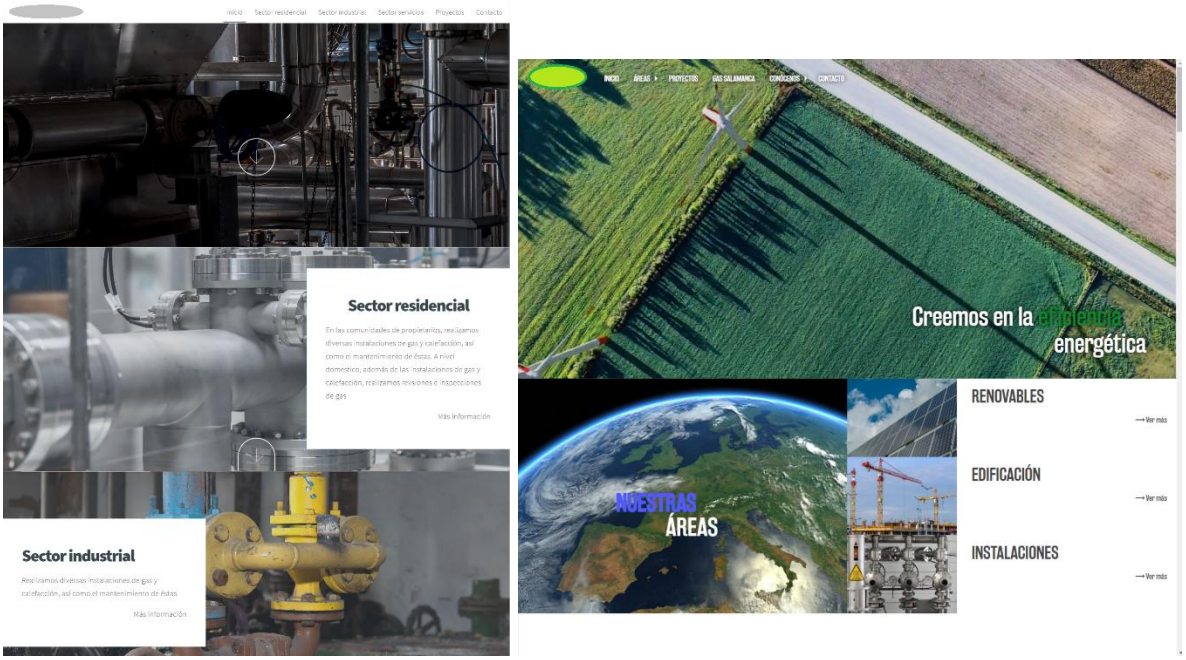


Fig 10. Ejemplos de Páginas Web (octubre 2018)

2. APLICACIONES MÓVILES

Se desarrollan aplicaciones móviles corporativas (y, por tanto, privadas). No se busca obtener una rentabilidad a través de los mercados de aplicaciones, sino que son parte del Software a medida que solicitan algunas compañías (***Fig 11***).

Los proyectos no son complejos, pero los desarrollos son más largos que en el caso de las páginas WEB. Necesitan de continuas iteraciones con el cliente para aprobar el diseño, las funcionalidades y la conexión segura con su servidor empresarial.

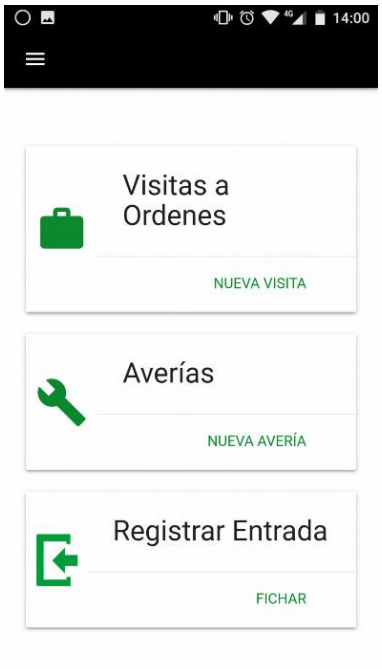


Fig 11 Ejemplo de aplicación móvil (septiembre 2018)

3. SOPORTE Y MANTENIMIENTO

Como fase final de algunos proyectos cuyo desarrollo se ha dado por concluido, en casi todos los casos existe un periodo de mantenimiento como garantía para el cliente. Por ello, invertir recursos en mantenimientos es consecuencia de no haber conseguido un desarrollo lo suficientemente sólido y, por tanto, acarrea improductividades.

4. CONSULTORÍA TECNOLÓGICA

Se colabora tanto con empresa privada como con administración pública (***Fig 12***) en labores de consultoría tecnológica. Se realizan pequeños proyectos que no implican programación, sino análisis de mejora de las competencias tecnológicas en diversos sectores.

También resultan proyectos cortos y sencillos, pero requieren de un gasto elevado en desplazamientos para realizar estudios 'in situ'.

Fig 12 Ejemplo de proyecto de consultoría tecnológica (Octubre 2018)

5. I+D+i

Esta etapa coincide con el inicio del primer proyecto de I+D+i en el que se ve envuelto el equipo. Implica tanto desarrollo de código como diseño 3D (***Fig 13***). Estos proyectos son a muy largo plazo, por lo que es difícil medir la rentabilidad del mismo.

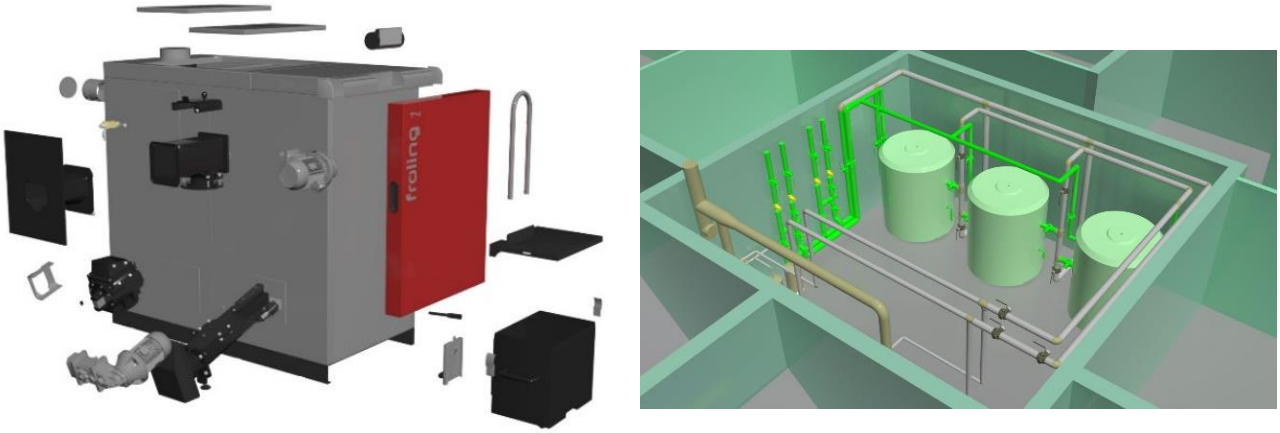


Fig 13 Proyecto de diseño 3D de I+D+i (septiembre 2018)

Una vez se da por concluida la adaptación de estas prácticas a la metodología de trabajo, y transcurridos dos meses del inicio de la implantación de las citadas medidas, se hace un nuevo estudio de situación (Tabla 2):

		NOVIEMBRE 2018
GESTIÓN DE PROYECTOS	ALCANCE	Los requisitos quedan definidos antes del inicio del proyecto. El equipo de desarrollo participa directamente en ello con el cliente, por lo que conoce mejor sus necesidades, aunque siguen siendo frecuentes los cambios en las especificaciones del producto a lo largo del desarrollo.
	TIEMPO	Por norma general, la gestión del plazo sigue siendo un problema, en gran parte, debido a que no existen entregas parciales durante los desarrollos. Las etapas finales de los proyectos implican un sobreesfuerzo del equipo.
	COSTE	La mayoría de proyectos son de pequeña entidad, lo que obliga a reducir al mínimo los recursos empleados en ellos. Durante este periodo, la rentabilidad obtenida continúa siendo baja
EQUIPO DE TRABAJO		Se ha conseguido descentralizar la toma de decisiones y repartir la responsabilidad de manera equitativa. Por medio de las continuas reuniones, los desarrollos se enfocan más al producto final que a la funcionalidad particular, y se utiliza un código más estandarizado y que comprende todo el equipo.
CONTACTO CON EL CLIENTE		Representa uno de los cambios más importantes que ha visto el equipo en esta etapa. No obstante, incuso en el corto plazo, se ha obtenido una mejora en la comunicación que ha favorecido tanto al cliente como al equipo de desarrollo.
ENTORNO		Las reuniones frecuentes y el hecho de que todos los desarrolladores sean responsables de alguno de los proyectos, han mejorado la comunicación dentro del equipo. Esto ha favorecido la compartición de código, la resolución de problemas en desarrollos individuales e, incluso, las relaciones interpersonales.

Tabla 2 Resumen de la situación a noviembre de 2018

3.2.2. Metodología XP

El ambiente general del equipo ha mejorado con la implantación de las primeras prácticas en metodologías ágiles, aunque no tanto el rendimiento y la productividad. Es necesario avanzar y dar otro paso más.

A pesar de los cambios experimentados, hay una constante que se ha mantenido durante los meses previos: el desarrollo en cascada. El principio más importante de las metodologías ágiles en el desarrollo de Software consiste en que a lo largo de la ejecución de los proyectos deben existir entregas funcionales, iterativas e incrementales.

Tras valorar los principios de algunas de las metodologías ágiles que más se utilizan en la actualidad, a principios de noviembre de 2018 se decide adoptar algunas de las prácticas del *Extreme Programming*, que está orientada a equipos pequeños y a proyectos cortos.

Durante este periodo, se toman las siguientes medidas:

- **Asignación de nuevos roles**

  La metodología XP define seis roles que debe tener todo equipo de programación. Con esta medida, no se pretende aumentar la carga de trabajo del equipo, sino repartir las responsabilidades en función de las habilidades, experiencia y cualidades de cada miembro del equipo en un proyecto.

  En nuestra situación, se conserva la asignación de un responsable de proyecto, que será uno de los miembros del equipo de programación. Además de realizar tareas de programación, asume el rol de *Tester* (responsable de pruebas).

  El responsable de la coordinación técnica del equipo es el responsable de que los procedimientos XP se están aplicando correctamente (*Coach*), además de ser el encargado de seguimiento del proyecto (*Tracker*). Para esta labor, debe existir una buena coordinación con el responsable del proyecto.

  Las labores de gestión (*Big Boss*) quedan para el Gerente de Negocio del equipo, y el rol de programador lo asumen el resto de desarrolladores que forman parte del equipo.

- **Definición de requisitos**

  En línea con lo propuesto por la metodología XP, aunque no exista un documento de *User Stories* como tal, se define en conjunto con el cliente cómo se a realizar el desarrollo de cada una de las funcionalidades. En lugar de diseñar dentro del equipo cuáles son los desarrollos que deben realizarse para cumplir los requisitos y especificaciones del cliente, se incorpora a este último a dicha planificación. Con ello, se pretende generar seguridad en él y, además, asegurar que aprueba los procedimientos del equipo.

- **Proceso continuo con entregas pequeñas**

  Los desarrollos comienzan a realizarse en ciclos de trabajo, que se establecen semanales. En las reuniones programadas (cuya frecuencia se conserva), se realiza una revisión progresiva de los desarrollos.

  Al final de cada ciclo, el producto debe contener todas las funcionalidades establecidas en la programación semanal y ser totalmente funcional, es decir, constituir un software independiente sin errores en su ejecución.

- **Integración de desarrollos independientes**

  Dentro de un mismo proyecto, todos los desarrollos independientes se integran semanalmente. En línea con la anterior medida, el objetivo es alcanzar al final de cada iteración un único Software que contenga el progreso del desarrollo de cada ciclo de trabajo, y no diferentes unidades funcionales independientes.

  Será el responsable de cada proyecto el encargado de realizar integraciones continuas, para lo cual debe tener una muy buena concepción global del proyecto, conocer el estado del trabajo de cada desarrollador y asegurar que se siguen los estándares de programación establecidos.

- **Cercanía al cliente**

  La metodología XP propone que el cliente forme parte del equipo de desarrollo durante la ejecución del proyecto. En este caso, eso representa una situación utópica, ya que en proyectos pequeños los clientes no consideran necesario participar en su desarrollo.

  No obstante, se lleva a cabo un cambio en la relación con el cliente. Al final de cada iteración, el cliente recibe el producto parcial que se ha obtenido al concluir el ciclo de trabajo, de tal manera que puede observar directamente el progreso en la ejecución de su proyecto.

- **Programación en parejas**

  Se decide implantar el principio más disruptivo de la metodología XP, la programación en parejas. Existen múltiples opiniones sobre el éxito de adoptar esta medida, por lo que se implementa en el grupo de trabajo.

  La medida adoptada, no obstante, no es tan radical como que una pareja de programadores utiliza un único ordenador para realizar un desarrollo, sino que la responsabilidad de cada tarea individual es siempre compartida por, al menos, dos personas del equipo.

Cuando esto ocurre, aunque en la práctica sea un único programador quien realiza la codificación, siempre hay otra persona del equipo opinando y supervisando el trabajo, además de colaborando en *tests* unitarios.

Nuevamente, una de las mayores barreras que se encuentran al inicio de la implantación de estas medidas es la adaptación de los proyectos en curso a la metodología XP. Abandonar el desarrollo en cascada y realizar los desarrollos en ciclos iterativos e incrementales, supone un cambio muy importante en la filosofía de trabajo para todos los integrantes del equipo, principalmente para aquellos con más experiencia.

Además, no tanto como adopción de medida XP, sino como decisión corporativa, se pretende comenzar con una reorientación del objetivo comercial. El equipo tiene potencial para realizar desarrollos en otras áreas de las nuevas tecnologías, por lo que se decide invertir en formación para el desarrollo de proyectos de implantación de ERPs, y aplicaciones de Realidad Virtual y Realidad Aumentada.

En este periodo, se realizan actividades como:

1. **PÁGINAS WEB**

El desarrollo de páginas Web sigue siendo una de las actividades más representativas de este periodo. Continúan siendo proyectos cortos pero de baja rentabilidad (***Fig 14***).

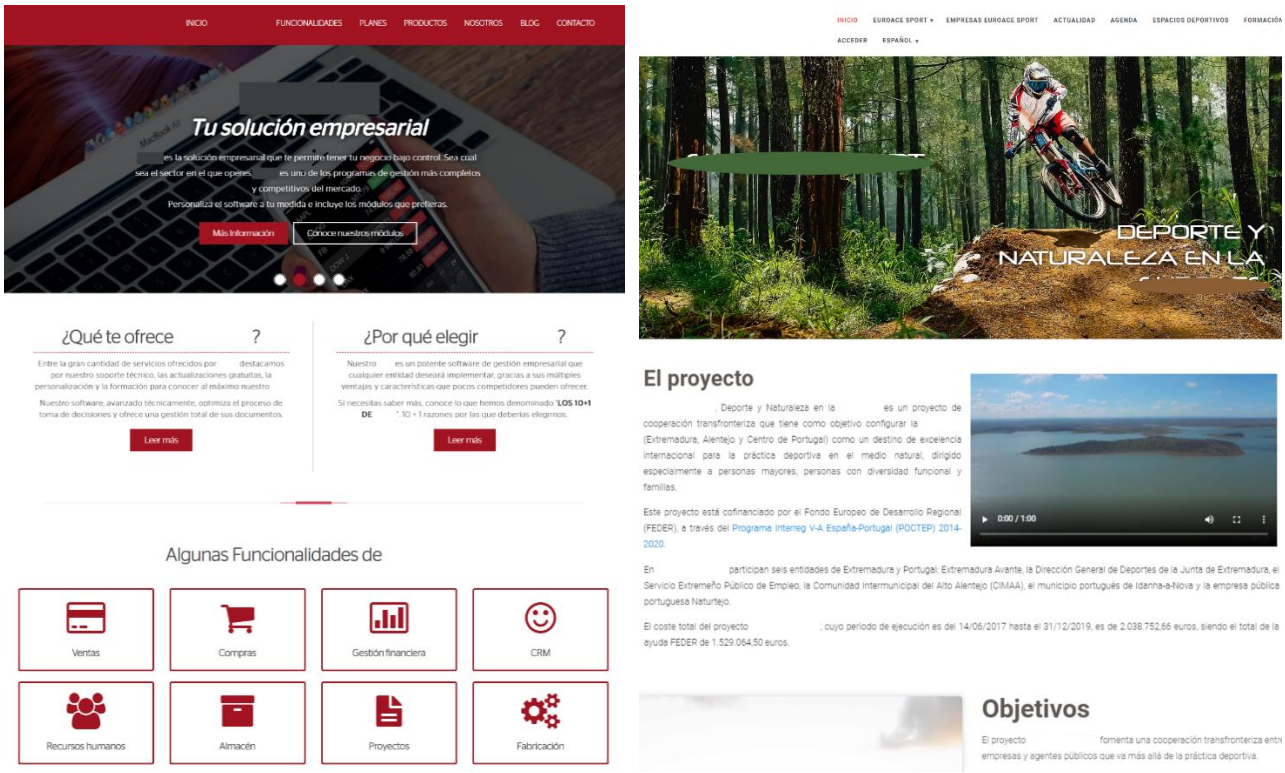


Fig 14 Ejemplo de Páginas Web (diciembre 2018)

2. APLICACIONES MÓVILES

Al igual que ocurre con las páginas Web, se mantienen los proyectos de desarrollo de aplicaciones móviles (***Fig 15***). No obstante, al ser más extensos en plazo, su ejecución se ve más afectada (positivamente) por la metodología de trabajo en ciclos iterativos.

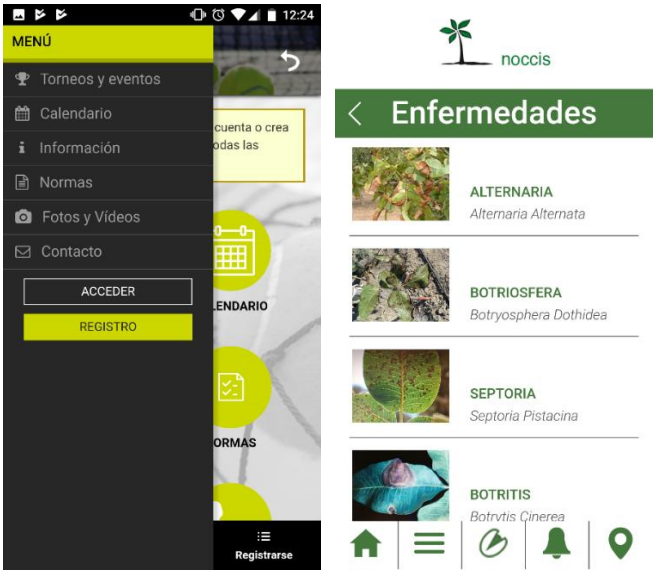


Fig 15 Ejemplo de aplicación móvil (febrero 2019)

3. REALIDAD VIRTUAL

Como marca la nueva línea estrategia corporativa, se apostar por diversificar la actividad del equipo y buscar nuevos desarrollos en otras tecnologías, como la Realidad Virtual (***Fig 16***). Esto implica una inversión importante, tanto en formación como en la incorporación de nuevos equipos y licencias de Software. La falta de experiencia hace que, en este periodo, los desarrollos sean lentos y poco rentables.

Fig 16 Ejemplo de aplicación de Realidad Virtual (febrero 2019)

4. REALIDAD AUMENTADA

Al igual que ocurre con las aplicaciones de Realidad Virtual, se realizan los primeros desarrollos de Realidad Aumentada (***Fig 17***). En este caso, el salto tecnológico es menos grande que en los proyectos de Realidad Virtual, y los recursos materiales necesarios son más económicos.

No obstante, el hecho de utilizar esta tecnología implica que los proyectos sean de mayor entidad. De esta manera, aunque la extensión en plazos se extienda más de lo previsto, la rentabilidad de estos desarrollos puede observarse desde un primer momento.

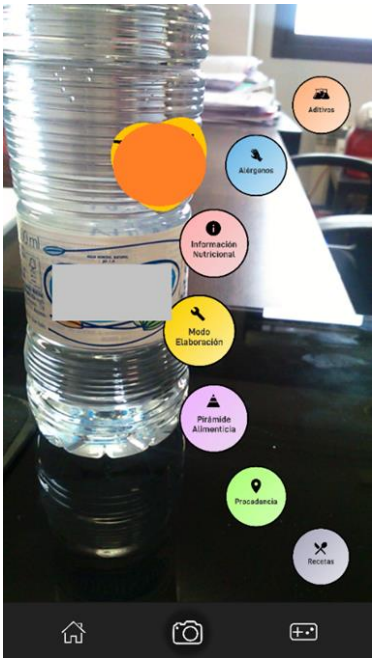


Fig 17 Ejemplo de aplicación de Realidad Aumentada (septiembre 2018)

5. IMPLANTACIÓN DE ERP

El equipo de desarrollo Software, además de realizar proyectos como los citados hasta el momento, da soporte informático a la propia empresa llevando a cabo, entre otras actividades, la implantación del ERP corporativo.

Ante el éxito de este desarrollo, se decide expandir el negocio y comercializar dicha herramienta, ejecutando proyectos de implantación de ERP para empresas (***Fig 18***). Estos proyectos son extensos en el tiempo, complejos y requieren de un gran esfuerzo de desarrollo, tanto en diseño como en programación.

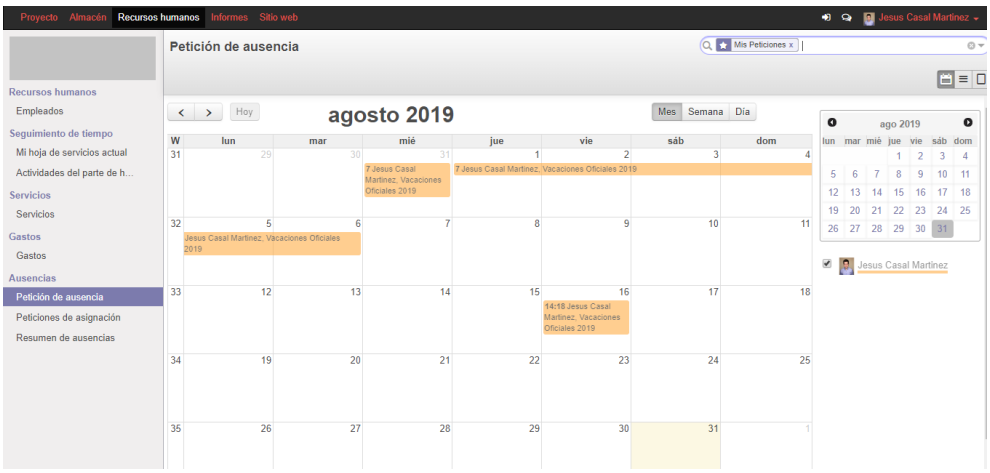


Fig 18 Ejemplo de desarrollo ERP (enero 2019)

Una vez se da por concluida la adaptación de estas prácticas a la metodología de trabajo, y transcurridos tres meses del inicio de la implantación de las citadas medidas, se hace un nuevo estudio de situación (Tabla 3):

		FEBRERO 2019
GESTIÓN DE PROYECTOS	ALCANCE	Tanto al inicio como durante el transcurso del proyecto, no solo van quedando definidas las funcionalidades que debe cumplir el Software final, sino que el cliente forma parte del proceso de diseño de las mismas. En los proyectos, desde un primer momento, se acepta que el desarrollo es flexible y, siguiendo criterios lógicos, se aceptan cambios sobre los requisitos de forma natural.
	TIEMPO	El hecho de expandir el objetivo comercial y desarrollar productos de tecnologías nuevas para el equipo, ha tenido como consecuencia que la planificación no se haya cumplido. Por norma general, al final de cada iteración, no se han conseguido los objetivos fijados, y es habitual hacer continuas adaptaciones en el calendarios
	COSTE	El volumen de negocio de los proyectos ejecutados en este periodo ha aumentado significativamente, lo que no implica, por norma general, que sean económicamente rentables. La reorientación del objetivo comercial ha significado desarrollar productos que han aumentado la facturación, pero que consumen más recursos materiales y humanos.
EQUIPO DE TRABAJO		La nueva definición de roles no ha traído consecuencias negativas, sino que ha servido para aclarar las funciones de cada miembro del equipo dentro de cada proyecto. Las responsabilidades quedan bien repartidas y el equipo considera que existe un equilibrio aceptable entre capacidad y experiencia de cada miembro.
CONTACTO CON EL CLIENTE		Uno de los pilares de la metodología XP es incluir al cliente en el equipo de trabajo. Este objetivo no se ha alcanzado, pero se ha suplido aumentando significativamente la implicación del cliente, tanto en la definición de funcionalidades, como comprobando el progreso de su producto tras cada ciclo de trabajo.
ENTORNO		La implementación del *Pair Programming* ha sido una buena medida, al menos, para mejorar las relaciones interpersonales dentro del equipo. Cada persona comprende y asume sus responsabilidades dentro de cada proyecto, lo que ha supuesto que el ambiente de trabajo mejore continuamente.

Tabla 3 Resumen de la situación a febrero de 2019

1.3.1. SCRUM

A pesar de que el ambiente en el equipo sea bueno, la dirección de la compañía está poco satisfecha con el rendimiento en los últimos meses. Considera que se ha hecho una inversión muy importante en formación y en la adquisición de equipos informáticos de elevado coste, y no se están encontrando los resultados esperados.

Además, algunas de las medidas adoptadas no han sido vistas con buenos ojos. La programación en parejas se entiende como una sobreasignación de recursos para el desarrollo de determinadas tareas. Mientras, acercar tanto al cliente a la planificación del proyecto puede estar implicando excesivas actualizaciones continuas en las funcionalidades, y una extensión en los plazos del proyecto no acorde al proyecto en cuestión.

Se considera que deben implantarse medidas de metodologías ágiles mucho más orientadas a mejorar la productividad del equipo y, por tanto, la rentabilidad de los proyectos.

La metodología SCRUM, desde su concepción, está orientada al desarrollo de Software, y sus principios pueden resultar de gran utilidad para alcanzar los objetivos propuestos, por lo que en febrero de 2019 se decide abandonar algunas de las prácticas de la metodología XP y adoptar estos. Durante este periodo, se implantan las siguientes medidas:

- Nueva definición de roles

  En primer lugar, se decide adoptar los roles propuestos por la metodología SCRUM. El número de roles queda, de esta manera, reducido a 3. La responsabilidad de representar al cliente (*Product Owner*) dentro del equipo pasa a ser del Gerente de negocio. Se pierde parte del contacto directo entre cliente y desarrolladores, y el *Product Owner* pasa a ser la representación del cliente dentro del equipo.

  Asegurar el cumplimiento de los principios de la metodología SCRUM (*SCRUM Master*) pasa a ser responsabilidad del coordinador técnico del equipo, que además realizará el seguimiento de los desarrollos. El *SCRUM Master* también planificará las actividades a realizar en cada *Sprint*.

  El rol de desarrollador lo asumen el resto de miembros del equipo. No desaparece la figura de responsable del proyecto, que seguirá cumpliendo las funciones de integración de los desarrollos y las pruebas funcionales de los mismos.

- **Ciclo de vida de los proyectos**

Se adopta el ciclo de vida propuesto por la metodología SCRUM y se implementa en la ejecución de los proyectos: Iniciación – Planificación – Implementación – Revisión – Lanzamiento.

Por primera vez, aparece el concepto de *Sprint*, que conserva las ideas ya implementadas de realizar desarrollos iterativos incrementales. Igualmente, al final de cada *Sprint* (que, inicialmente, abarca una semana de trabajo), debe resultar un producto funcional.

- **Seguimiento de tareas en Tablero *Kanban***

En cada reunión de planificación semanal (o del *Sprint*) se asignan tareas por persona y proyecto. El tablero se divide en cinco estados (***Fig 19.***):

- Pendiente: Tareas que se deben hacerse pero no han entrado en la planificación del *Sprint* en curso.
- Planificado: Son las tareas que está previsto ejecutar en el *Sprint* en curso.
- En Ejecución: Tareas en curso.
- Revisión: Aquellos desarrollos cuya validación está pendiente de una prueba de calidad dentro del propio equipo, o por parte del cliente si fuera necesario.
- Finalizado: Aquellas tareas que se pueden dar por concluidas dentro de un *Sprint*.

Cada tarea se describe en un *Post-it*, y cada persona tiene asignado un color, de manera que todos pueden ver de forma ágil el estado de los desarrollos de cada miembro del equipo.

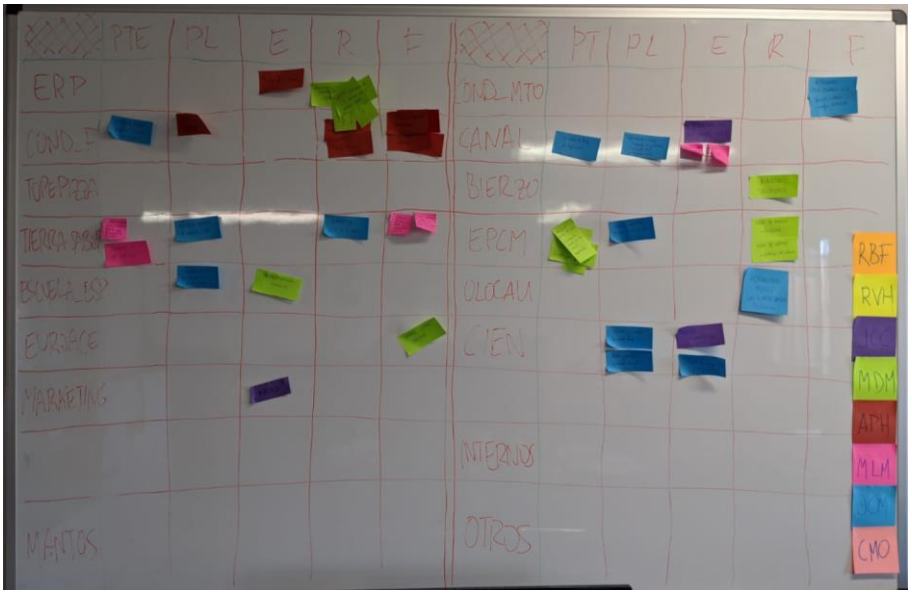


Fig 19 Imagen de tablero Kanban en abril de 2019

A lo largo de este periodo, continuando con la línea estratégica establecida en los meses anteriores, los proyectos más representativos dejan de ser sobre páginas Web y aplicaciones móviles sencillas.

Además, de manera interna, se toma la decisión estratégica de invertir recursos propios en el desarrollo de herramientas software comercializables, es decir, productos que puedan tener su espacio en el mercado. Por lo tanto, no existe un cliente final como tal, y los requisitos y funcionalidades deben definirse dentro del propio equipo del proyecto.

Las actividades más representativas de este periodo son:

1. **PÁGINAS WEB Y APLICACIONES MÓVILES**

Aunque el número de proyectos de páginas Web y Apps se reduce significativamente, sigue representando una actividad habitual para el equipo (***Fig 20***). Los proyectos son cortos, y los *Sprints* se adaptan a los módulos de desarrollo, pero los bajos ingresos que generan hacen que no reporten una elevada rentabilidad.

Fig 20 Ejemplo de página Web y Aplicación móvil (abril 2019)

2. **SOFTWARE COMERCIAL**

En lugar de ejecutar proyectos de Software a medida, a petición de un cliente, se toma la decisión corporativa de realizar distintos desarrollos que puedan generar productos comercializables (***Fig 21***). Inicialmente, estos proyectos no son productivos y consumen recursos materiales y humanos sin que exista ningún ingreso, y empezarán a ser rentables a partir de un determinado número de compras.

Se buscan realizar desarrollos sencillos, de un Software que pueda tener una gran aceptación en el mercado, y que pueda comercializarse de manera '*low cost*'.

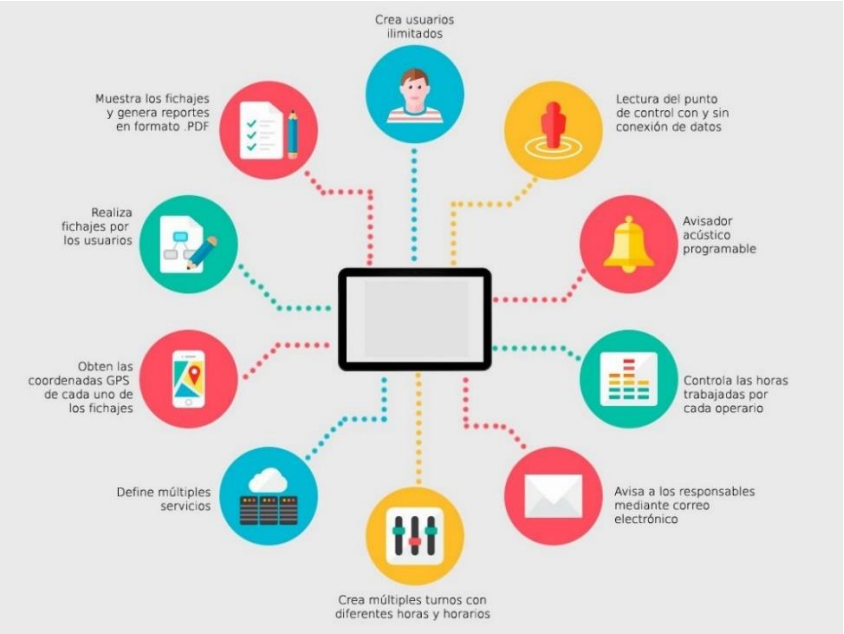


Fig 21 Ejemplo de software comercial (marzo 2019)

3. APLICACIONES DE REALIDAD VIRTUAL

Se continúan realizando desarrollos de Realidad Virtual (***Fig 22***). Estos proyectos son, por norma general, de gran entidad, y consumen una gran cantidad de recursos materiales porque los equipos tienen un coste muy elevado.

Implementar esta tecnología en los desarrollos Software que solicita el cliente es un proceso largo y complejo, y en *Sprints* de una semana no se consiguen los objetivos que se proponen. No obstante, aunque no se cumplan con los plazos previstos en la planificación, se cierran varios proyectos este tipo.

Fig 22 Ejemplo de aplicación de Realidad Virtual (junio 2019)

4. APLICACIONES DE REALIDAD AUMENTADA

La apuesta corporativa de orientar el equipo al desarrollo de este tipo de proyectos conduce a que esta actividad sea la más representativa de este periodo (***Fig 23***). El mercado es muy amplio, y el equipo ha conseguido adaptar muy bien el ritmo de trabajo a los *Sprints*.

Por norma general, estos desarrollos se adaptan correctamente a las exigencias del cliente en plazo y calidad, y con la experiencia que se adquiere en los primeros meses, pronto estos proyectos empiezan a reportar una buena rentabilidad.

Fig 23 Ejemplo de aplicación de Realidad Aumentada (mayo 2019)

5. SOFTWARE A MEDIDA

Durante este periodo también ha sido representativo el número de proyectos de Software a medida para algunas empresas (***Fig 24***). Estos desarrollos, además de ser extensos en el tiempo, son principalmente complejos. El representante del cliente (*Product Owner*) en el equipo hace una labor muy importante, pues las funcionalidades a desarrollar son muy específicas.

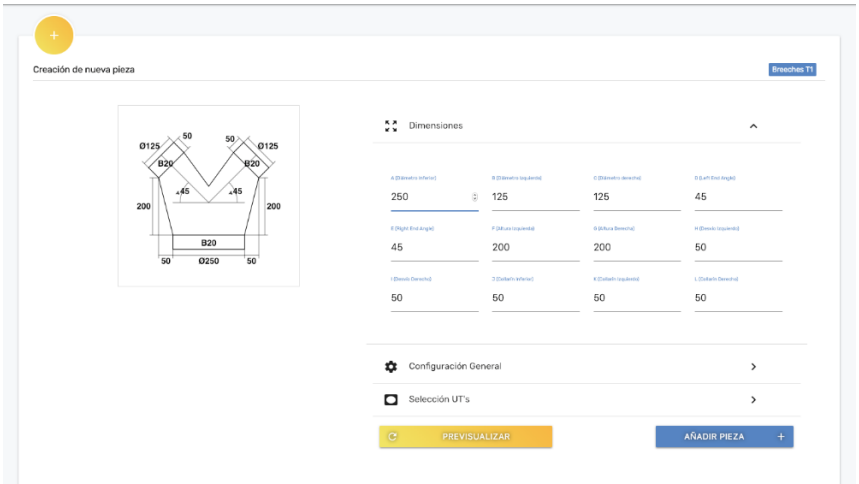


Fig 24 Ejemplo de software empresarial (abril 2019)

Transcurridos unos meses desde la implantación de las medidas, la directiva considera que la productividad ha aumentado considerablemente, por lo que se continúa aplicando las prácticas de la metodología SCRUM durante seis meses. Transcurrido este periodo, se hace un nuevo análisis de situación:

		JULIO 2019
GESTIÓN DE PROYECTOS	ALCANCE	Los requisitos quedan definidos al inicio del proyecto, y el *Product Owner* actúa como representación del cliente dentro del equipo. Se reduce la flexibilidad en los cambios de funcionalidades, que gestiona el *Product Owner* con el cliente y el responsable del proyecto.
	TIEMPO	El *SCRUM Master* es el responsable de gestionar el *Product Backlog* e ir asignando módulos funcionales a *Sprints* dentro de un proyecto. Gracias al tablero *Kanban* se realiza un seguimiento general continuo dentro de un *Sprint*, y todos los miembros del equipo conocen el estado de las tareas del resto, lo que ayuda a realizar actualizaciones realistas en la planificación de manera más sencilla.
	COSTE	La estrategia comercial del equipo ha conducido a que los desarrollos en Realidad Virtual y Realidad Aumentada acaparen la mayor parte de los proyectos en este periodo, lo que implica que el volumen de negocio aumente considerablemente. Estos proyectos se han adaptado bien al ciclo de vida del SCRUM, lo que en el corto - medio plazo está conduciendo a una rentabilidad aceptable.
EQUIPO DE TRABAJO		La nueva asignación de roles ha cambiado parcialmente las responsabilidades dentro del equipo, aunque no ha tenido consecuencias negativas sobre el entorno de trabajo. El cambio más importante ha sido alejar al cliente del equipo de desarrollo, y ha hecho fundamental el trabajo del *Project Owner*.
CONTACTO CON EL CLIENTE		El contacto del cliente con el equipo sigue siendo directo y frecuente, pero solo implica al *Project Owner* y al responsable del proyecto como representación de los desarrolladores. No obstante, el principal vínculo entre el cliente y el equipo será el *Project Owner*, que durante la ejecución del proyecto realizará una labor de intermediario, por lo que su sincronización con ambas partes debe ser efectiva.
ENTORNO		El distanciamiento con el cliente por parte del equipo de desarrollo ha sido positiva para la estabilidad laboral individual. Las relaciones interpersonales son buenas y, en general, el ambiente de trabajo es agradable.

Tabla 4 Resumen de situación en junio 2019

4. CONCLUSIÓN Y LÍNEAS FUTURAS

Transcurridos 11 meses desde la implantación de las primeras prácticas de metodologías ágiles en desarrollo de Software, se puede hacer una retrospectiva de la evolución en todos los ámbitos del equipo durante el periodo completo.

		SEPTIEMBRE 2018	NOVIEMBRE 2018	FEBRERO 2019	JULIO 2019
GESTIÓN DE PROYECTOS	ALCANCE	- Desarrollo en cascada - Poca flexibilidad	- Desarrollo en cascada - Flexibilidad moderada - Equipo de programación participa en la definición de funcionalidades	- Desarrollos interativos e incrementales en ciclos de trabajo - Flexibilidad muy alta - Colaboración cliente - desarrollador	- Sprints como ciclos de trabajo - Product Owner como enlace cliente - desarrolladores - Flexibilidad moderada
	TIEMPO	- Etapas finales de proyecto con sobrecarga de trabajo - Mala planificación	- Etapas finales de proyecto con sobrecarga de trabajo - Mala planificación	- Ciclos de trabajo mal planificados	- Mejora continua en la planificación - Tablero Kanban - Algunos proyectos no pueden seguir los ciclos establecidos
	COSTE	- Proyectos de bajo coste y baja rentabilidad	- Proyectos de bajo coste y baja rentabilidad	- Cambio en estrategia comercial - Aumenta volumen de negocio pero no la rentabilidad	- Afianzamiento en el mercado de nuevas tecnologías - Aumenta volumen de negocio - Aumenta rentabilidad
EQUIPO DE TRABAJO		- Responsabilidad total y gestión integral centralizada - Se imita el funcionamiento de otras divisiones de la compañía	- Reparto de tareas de gestión - Figura de responsable de proyecto. Responsabilidad descentralizada	- Nuevos roles, similares responsabilidades - Figura de Rble. de Proyecto.	- Cambio de roles y responsabilidades - Buena adaptación - Mucha importancia del Project Owner
CONTACTO CON EL CLIENTE		- Centralizado en una persona - El equipo de proyecto no tiene contacto con el cliente - Vía mail	- Contacto directo desarrollador - cliente - Contacto frecuente presencial / semipresencial	- Contacto directo desarrollador - cliente - Contacto muy frecuente presencial - Cliente o representacion del cliente puede formar parte del equipo	- Contacto directo entre cliente - Rble de proyecto - Se procura que exista intermediación de Project Owner - Contacto frecuente presencial /semi
EQUIPO DE TRABAJO		- Desarrollos independientes - Pocas relaciones personales	- Reuniones, comunicación constante con Rble. de proyecto - Mejora en relaciones interpersonales	- Pair Programming - Mejora en relaciones personales - Entorno agradable	- Entorno colaborativo agradable

Tabla 5 Evolución de la situación a lo largo del caso práctico

En términos generales, se puede afirmar que las prácticas implementadas, en el medio plazo, han tenido un efecto positivo en la dinámica de trabajo. La dirección de la compañía ha confiado en el grupo, haciendo sucesivos esfuerzos de inversión en formación y múltiples adquisiciones para mejorar la productividad y la calidad de los desarrollos.

Tanto en la dirección como el propio equipo hay una sensación de satisfacción por los resultados obtenidos, pero es necesario continuar con trabajando para corregir deficiencias y seguir mejorando el rendimiento en los desarrollos.

4.1. Líneas futuras

Tras las conclusiones obtenidas del último análisis de situación, se trazan las líneas futuras para establecer nuevas medidas que subsanen las debilidades encontradas:

- **Reducción del número de proyectos**

  Tradicionalmente, se ha venido trabajando con una cantidad muy elevada de proyectos. Como los ingresos que estos generaban eran bajos, era necesario desarrollar simultáneamente una cantidad muy elevada para obtener rentabilidad. Esto implicaba que todos los desarrolladores estuvieran implicados simultáneamente en un número grande de proyectos, lo que muchas veces provoca falta de concentración en la programación y, por ende, improductividades.

  Con la nueva línea estratégica corporativa, se espera que en el medio plazo desaparezca los proyectos de menor envergadura, para poder centrar esfuerzos en desarrollos complejos de alto valor añadido. A la larga, se pretende ampliar el grupo y poder separar los equipos de desarrollo por proyecto, con un 100% de implicación en ellos.

- **Simplificación tablero *Kanban*, mejora de la planificación**

  A veces, dentro del propio equipo, existen dudas de en qué estado se encuentran determinadas actividades, y el tablero puede llegar a ser confuso. Cuando esto ocurre, se dan por finalizadas tareas que no han sido correctamente revisadas y aparecen incidentes externos con el cliente.

  Van a desaparecer las columnas de Planificación (que gestionará internamente el *SCRUM Master* por medio del *Product Backlog*), y de Revisión. El responsable de cada proyecto será el único que podrá dar por válidos los desarrollos y, por tanto, autorizará que las tareas de su proyecto se puedan dar por terminadas.

  Además, a cada tarea se le asignará una duración (cuya unidad de medida se decidirá en un futuro), que se añadirá a las tarjetas (en forma de Post – it), con el objetivo de sacar el máximo rendimiento de cada desarrollador y asegurar que no existan improductividades.

- **Acercamiento al cliente**

  Con la implementación de las prácticas SCRUM se ha dado un paso hacia atrás en el acercamiento del cliente al equipo de desarrollo.

  La intención de acercar cliente y equipo de desarrollo, es resolver dudas que puedan existir en la ejecución del proyecto y generar confianza en el cliente. No obstante, con las medidas que se habían implementado, lo que se había conseguido es que el cliente estuviera muy cerca del desarrollo, llegando incluso a la intromisión dentro del propio código.

  En esta situación, se detectó un exceso de permisividad, y que la situación óptima es que el desarrollador esté cerca del cliente, y no lo contrario (que es lo que se había alcanzado), por lo que es necesario trabajar en este aspecto para continuar mejorando en la relación cliente – equipo.

4.2. Conclusiones generales

Tras los conocimientos adquiridos y las medidas que se han implementado, poniendo en práctica las metodologías ágiles en el desarrollo de Software en un equipo real, se extraen las siguientes conclusiones:

- Descentralizar y repartir las responsabilidades dentro de un equipo de trabajo es fundamental para facilitar tareas de gestión y mejorar la comunicación dentro del propio grupo.

- El mejor seguimiento de los proyectos se ha conseguido juntando reuniones rutinarias del equipo y el tablero *Kanban*, a través del cual cada miembro pueden conocer el estado de las tareas del resto.

- Un ambiente de trabajo agradable basado en un entorno colaborativo fomenta la productividad y favorece la formación continua del equipo.

- El ciclo de trabajo óptimo para cada proyecto puede variar en función de su complejidad, pero el ciclo de vida del SCRUM (que divide el desarrollo en iteraciones continuas llamadas *Sprints*) se adapta muy bien a los proyectos de desarrollo de Software.

5. BIBLIOGRAFÍA

[1] *610.12-1990 IEEE Standard Glossary of Software Engineering Terminology.* .

[2] I. Jacobson, G. Booch, and J. Rumbaugh, "El proceso unificado de desarrollo de software/The unified software development process.," 2000.

[3] Ming Huo, J. Verner, Liming Zhu, and M. A. Babar, "Software quality and agile methods," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pp. 520–525.

[4] "Explicación de los valores ágiles | Cátedra Viewnext USAL." [Online]. Available: https://viewnext.usal.es/blog/explicación-de-los-valores-ágiles.

[5] A. Cockburn and J. Highsmith, "Agile software development: The people factor," *Computer (Long. Beach. Calif).*, vol. 34, no. 11, pp. 131–133, Nov. 2001.

[6] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," *Computer (Long. Beach. Calif).*, vol. 34, no. 9, pp. 120–127, 2001.

[7] J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, vol. 12. 2000.

[8] "Introducción a las Metodologías Ágiles - Instinto Binario." [Online]. Available: https://instintobinario.com/introduccion-a-las-metodologias-agiles/.

[9] P. Letelier, "Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)," 2006.

[10] K. Beck and C. Andres, *Extreme programming explained : embrace change*. Addison-Wesley, 2005.

[11] "Blog de un apóstol de Scrum y Kanban: ¿Cómo se complementan eXtreme Programming y Scrum?" [Online]. Available: http://scrum.menzinsky.com/2017/07/como-se-complementan-extreme.html.

[12] M. Pezzè and M. Young, *Software testing and analysis : process, principles, and techniques*. Wiley, 2008.

[13] M. Fowler, "Refactoring: improving the design of existing code," 2018.

[14] K. S.-B. object design and implementation and undefined 1997, "Scrum development process," *Springer*.

[15] M. Cohn, "Succeeding with agile: software development using Scrum," 2010.

[16] L. Rising, N. J.-I. software, and undefined 2000, "The Scrum software development process for small teams," *ieeexplore.ieee.org*.

[17] K. Schwaber and M. Beedle, "Agile software development with Scrum," 2002.

[18] "Kanban for Law: How I Use Kanban to Manage LegalRnD Projects | The Law Project." [Online]. Available: https://www.lawprojectblog.com/2017/11/kanban-law-use-kanban-manage-legalrnd-projects/.

[19] F. Balle and M. Balle, “Lean Development,” *Bus. Strateg. Rev.*, vol. 16, no. 3, pp. 17–22, Aug. 2005.

[20] C. Ebert, P. Abrahamsson, and N. Oza, “Lean Software Development,” *IEEE Softw.*, vol. 29, no. 5, pp. 22–25, Sep. 2012.

[21] M. Poppendieck and M. A. Cusumano, “Lean Software Development: A Tutorial,” *IEEE Softw.*, vol. 29, no. 5, pp. 26–32, Sep. 2012.