7. Agilidad en el Desarrollo de Software

7.1. Introducción y objetivos

En ingeniería de *software* las técnicas ágiles suponen un enfoque para gestionar proyectos donde existe mucha incertidumbre, los requisitos son muy cambiantes y se requiere una alta retroalimentación con el cliente para minimizar los riesgos. El desarrollo ágil implica una nueva filosofía a la hora de abordar el desarrollo de un producto *software* y esta nueva filosofía también implica una nueva forma de planificar y estimar considerando su refinamiento progresivo para alcanzar una visión más realista del avance del proyecto.

En este tema se estudiará la importancia de la agilidad en el desarrollo de *software* con la finalidad de ayudar a mejorar el proceso de desarrollo de *software* en base a un enfoque iterativo e incremental,donde se da un mayor valor al individuo y a la colaboración con el cliente.

Con el estudio de este tema pretendemos alcanzar los siguientes objetivos:

* Comprender el concepto de «agilidad» en el contexto del desarrollo de *software.*
* Conocer los principios y valores del manifiesto ágil.
* Conocer las principales características de las técnicas ágiles y sus ventajas con respecto a las técnicas tradicionales.
* Reflexionar sobre la importancia de seleccionar la técnica más adecuada en función de las características de cada proyecto.
* Comprender conceptos importantes en el desarrollo ágil como *Product Owner* o historias de usuario.
* Conocer diferentes técnicas de priorización de historias de usuario en base al valor y la estimación.

7.2. Agilidad en el software

A pesar de que la ingeniería de *software* ha ido madurando y evolucionando a lo largo del tiempo desde su aparición oficial como disciplina en 1968, continuamos inmersos en una «crisis del software». El porcentaje de fracaso de los proyectos sigue siendo demasiado elevados, se siguen incumpliendo plazos y presupuestos, además de producir *software* de baja calidad, en muchos casos, de comportamiento impredecible.

Ya se ha visto que construir *software* no es igual que construir edificios o coches, productos físicos en general, por lo que no se pueden aplicar los mismos principios y teorías como, por ejemplo, caer en el error de medir la productividad por las líneas de código escritas y pensar que, si un proyecto va retrasado, la solución es contratar a más gente. La ingeniería de *software* es una disciplina «especial» que requiere otro tratamiento y necesita seguir otro tipo de técnicas o metodologías.

Durante el desarrollo de los proyectos de *software* debemos enfrentarnos con un número de incertidumbres que a *priori* pueden resultar difíciles de valorar. Esto ha sido así desde los comienzos de la ingeniería del *software,*pero durante las últimas décadas (con la popularización del uso de Internet y el auge de las aplicaciones web y móviles), los siguientes temas están aún más presentes:

* La dificultad para anticipar la volatilidad de los requisitos o pronosticar las prioridades del cliente respecto a los requisitos existentes a medida que el proyecto avanza.
* Muchos productos *software* están en continua evolución, de manera que las actividades de análisis, diseño e implementación se suceden de manera muy rápida, a menudo solapada y muy acoplada. Ello obliga a desarrollar estas actividades de manera simultánea en períodos de tiempo reducidos.

Se trata, pues, de desarrollar técnicas, marcos de trabajo o guías capaces de afrontar «lo impredecible»,manteniendo al mismo tiempo el grado de avance del proyecto.

Por ello, en la sociedad actual, donde cada vez de manera más rápida surgen nuevas necesidades, tecnologías, etc., las técnicas ágiles podrían ser una buena opción para dirigir y gestionar cierto tipo de proyectos *software.* Tal y como afirma Beck (Beck y Andres, 2004):

«[…] las cosas vinculadas al software cambian. Cambian los requisitos, el diseño, el negocio, la tecnología, los miembros del equipo de desarrollo, así como el equipo de forma global. Todo cambia. Sin embargo, el problema no está en el cambio en sí, porque los cambios se van a producir de manera inevitable, sino que el problema está en nuestra incapacidad para hacer frente a los cambios que se producen».

Las técnicas ágiles se adaptan muy bien a proyectos donde los requisitos cambian constantemente y en los cuales cuando los tiempos de desarrollo no pueden extenderse demasiado en el tiempo porque para cuando estuvieran operativos ya estarían obsoletos.

Las técnicas ágiles siguen el modelo de proceso iterativo e incremental que ya se estudió en el tema 1 (ver Figura 1), de tal forma que, cada poco tiempo, se va liberando una parte del producto *software* en forma de prototipo que permite su evaluación, así como la definición de mejoras y nuevos requisitos hasta llegar al producto final.

Por un lado, con el enfoque iterativo se consigue mejorar el producto *software* que se lleva implementado, ya que en cada iteración se revisa el trabajo realizado y se proponen mejoras y nuevas funcionalidades. En el desarrollo ágil, las iteraciones deben ser cortas, de un mes como máximo. Por otro lado, con el enfoque incremental, se consigue eliminar los riesgos asociados a los proyectos *software* de gran tamaño o complejidad, ya que el *software* se va implementando en partes que se van a ir integrando gradualmente.

A collage of pictures of a person

Description automatically generated

Figura 1. Iterativo vs Incremental. Fuente: Jan, 2008.

Aparte de los ya vistos en el tema 1, en lo que respecta a otros modelos de procesos, los beneficios más significativos que va a aportar el enfoque conjunto iterativo e incremental a una técnica ágil son:

* Visión de progreso. Desde las etapas iniciales del proyecto de *software* se puede ver trabajo operativo real del sistema a implementar.
* *Feedback* del cliente/usuario sobre un prototipo. Se van ajustando las funcionalidades que ha de contemplar el sistema y permite realizar las adaptaciones que sean necesarias para ir cumpliendo los objetivos del proyecto.
* Gestión de riesgos. Los riesgos se van a gestionar mejor, ya que al resolver los problemas del sistema a implementar por partes no se ha de realizar un análisis de riegos global del *software.*
* Aprendizaje y experiencia del equipo de desarrollo. Al realizar prototipos operativos en cada iteración del sistema, se mejora exponencialmente el trabajo, aumentando la productividad y optimizando el proceso de desarrollo en cada iteración.

La agilidad vincula da a la ingeniería de *software* ha sido descrita por Jacobson (2002) como la mejor manera de responder a los cambios constantes del *software.* Además, un equipo ágil de desarrollo de *software* es un equipo que colaborará de una manera real para el éxito del proyecto (Jacobson, 2002), es decir, en la agilidad aplicada al desarrollo de *software,* el trabajo en equipo y los aspectos humanos resultan fundamentales.

Sin embargo, tal y como destaca Pressman (2010), la agilidad aplicada a la ingeniería de *software* es algo más que una respuesta efectiva al cambio y el trabajo en equipo. El *software* se debe diseñar de tal manera que permita al equipo del proyecto adaptarse a las tareas que van surgiendo de manera continua, muy rápida, eliminando todo trabajo que no vaya orientado al cliente, poniendo énfasis en la entrega incremental para que se pueda ir validando y mejorando lo que ya esté implementado.

En Pressman (2010, p. 58), se identifican los aspectos vinculados al desarrollo de *software* en los que se asienta el proceso ágil para su implantación:

* Predecir qué requisitos del sistema van a mantenerse y cuáles se van a cambiar no es una tarea sencilla. También resulta complicado saber a *priori* cómo van a ir evolucionando los intereses y las expectativas del cliente en lo que al *software* se refiere.
* A la hora de desarrollar cierto tipo de *software* las fases de diseño e implementación se solapan en el tiempo, por lo que los modelos de diseño se van probando a medida que se van desarrollando. Sin embargo, resulta complejo determinar el diseño necesario que se ha de realizar para implementarlo e ir probando el sistema.
* A la hora de planificar el proyecto *software,* intentar estimar lo que va a llevar el análisis, diseño, implementación y prueba del *software* se convierte en una tarea bastante complicada, ya que no resulta nada predecible en la mayoría de los casos (desde un principio, los requisitos deberían estar muy claros y cerrados, sin opción de cambio o siendo este muy limitado).

Como se ha comentado, la agilidad es mucho más que la capacidad de responder de manera rápida y efectiva a los cambios; supone una filosofía completa y general del proceso de desarrollo. Durante las dos últimas décadas diferentes equipos de desarrollo adoptaron nuevos métodos y prácticas que buscaban este nuevo enfoque general, en 2001 mantuvieron una reunión en la que formalizaron estas ideas y las materializaron en lo que conocemos como el manifiesto ágil.

### **7.3. El manifiesto ágil**

El **manifiesto ágil** (Beck et al., 2001) se escribió en Utah el 12 de febrero de 2001, por profesionales de renombre en ese momento vinculados a la ingeniería de software y a algunas de las propuestas de desarrollo ágil más famosas hoy en día, como por ejemplo: Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM) y Crystal.

El objetivo del manifiesto era establecer una serie de valores y principios que permitieran el desarrollo de software de calidad, de una manera rápida y que se adaptara bien a los requisitos cambiantes.

Muchos de los aspectos que presenta el manifiesto ágil no eran novedosos cuando se escribió, pero sirvió precisamente para **plasmar y sintetizar** todo aquello que esos profesionales, eruditos, de la ingeniería de software entendían como las cosas a tener en cuenta y el modelo a seguir a la hora de desarrollar software.

El manifiesto ágil se dividió en **cuatro valores y doce principios.** Los valores que establece el manifiesto son los que se indican a continuación:

* «Valorar a los individuos y las interacciones del equipo de desarrollo prima sobre el proceso y las herramientas».
* «Desarrollar software que funciona prima sobre una documentación exhaustiva».
* «La colaboración con el cliente prima sobre la negociación de un contrato».
* «Responder a los cambios prima sobre el seguimiento estricto de un plan».

### **Valoración de individuos e interacciones**

El software está construido por personas. Un factor clave del éxito de los proyectos es el **equipo,** las relaciones entre sus miembros y la habilidad para construir el propio entorno de trabajo. Es un error operar a la inversa, construyendo primero el entorno de trabajo y esperando que la gente se adapte a él (Fernández, 2012).

Importan tanto las **interacciones** entre los miembros del equipo como el **flujo de información** entre cliente o usuario final y desarrolladores. Idealmente, el cliente debe encontrarse próximo al equipo o formando parte de este, de manera que los canales de comunicación sean los adecuados. Si se consigue un **nivel de comunicación eficiente,** los procesos y herramientas concretos pasan a un segundo plano o, en todo caso, deben ser elegidos y configurados por los propios actores del proceso de desarrollo.

### **Valoración del software en funcionamiento**

La**documentación de un proyecto** es una herramienta fundamental para comprenderlo. Resulta especialmente útil en la fase de mantenimiento y siempre que hay que modificar o mejorar el sistema (especialmente cuando el equipo de desarrollo y el de mantenimiento no coincidan). Pero generar y mantener la**documentación actualizada** es costoso y, en muchos casos, inútil.

Los firmantes del manifiesto, por su experiencia, descubrieron que, en la mayoría de los casos, cuando hay que resolver un problema lo importante es hacer que el software funcione, atender a los usuarios y conseguir que el sistema avance. La documentación pasa a un segundo plano, ya sea por falta de tiempo o de otros recursos.

La documentación es necesaria, pero su desarrollo debe mantenerse en los niveles estrictamente necesarios para que cumpla su misión, siendo breve y centrándose en lo fundamental.

Los **recursos** deben orientarse hacia tareas que verdaderamente aportan valor al proyecto, centradas en el desarrollo del propio sistema. A partir de esta idea, las técnicas ágiles proponen artefactos que vienen a sustituir a la documentación tradicional.

### **La colaboración con el cliente**

Uno de los principales puntos de fricción en los proyectos de desarrollo de software es la **negociación contractual** entre cliente y desarrollador y, especialmente, las posibles **modificaciones** que surgen a lo largo del desarrollo del proyecto. Cambios de alcance, modificaciones de requisitos o variaciones de plazos suponen escoyos para el proyecto y obligan a los desarrolladores a adoptar posiciones a caballo entre las labores de analista y abogado.

El manifiesto ágil parte de la base de que, tanto el cliente como el equipo de desarrollo, tienen el objetivo común de desarrollar un **sistema funcional** que satisfaga a los usuarios finales. Para ello, es importante conseguir los canales de comunicación adecuados y la complicidad con el cliente, que es considerado como uno más del equipo.

### **La respuesta al cambio**

El cambio resulta inevitable. Las necesidades de las complejas organizaciones de hoy en día son cambiantes y el propio entorno tecnológico, la competencia, los consumidores, etc., evolucionan a un ritmo cada vez más vertiginoso. Cualquier plan de proyecto puede verse afectado por múltiples imprevistos y una clave del éxito es la **adaptación rápida** y la**replanificación ágil.**

No tiene sentido elaborar grandes planes de proyecto, consumiendo un número de recursos elevado, cuando el futuro puede resultar incierto o cambiante.

Por su parte, asociados a estos cuatro valores, los firmantes del manifiesto elaboraron también un listado de **doce principios,** que siguen todos los proyectos desarrollados bajo las denominadas técnicas ágiles:

* La prioridad más alta en el desarrollo de software es cumplir con las expectativas del cliente a través de entregas útiles del software, de manera continua y en periodos cortos.
* Los cambios en los requisitos son aceptados en cualquier momento del proceso de desarrollo de software, incluso cuando la aplicación está prácticamente implementada en su totalidad. Precisamente, los procesos ágiles para el desarrollo de software toman los cambios en los requisitos como una ventaja competitiva o un valor añadido para el cliente.
* Entrega de software funcional frecuentemente. El periodo en el que el cliente debería recibir softwareoperativo y útil ha de ser entre dos semanas a un par de meses, pero no más.
* Tanto los expertos del dominio o negocio como los desarrolladores han de trabajar en equipo, colaborando ambas partes juntas, a diario si es posible, y a lo largo de todo el proceso. Entender bien el proceso de negocio favorecerá la introducción de mejoras y la identificación de nuevas oportunidades, como por ejemplo innovación.
* El desarrollo del software ha de realizarse en un ambiente de trabajo que motive al equipo implicado. Los miembros del equipo de desarrollo han de sentirse bien en su entorno laboral, saber que cuentan con el apoyo necesario y que podrán realizar su trabajo de manera satisfactoria. Además, el equipo de desarrollo dispondrá de la autonomía suficiente para poder tomar decisiones tanto a nivel técnico como a otros niveles del proyecto (Pressman, 2010, p. 60)
* La forma más eficaz y eficiente de transmitir la información a los miembros del equipo de desarrollo es la que se lleva cara a cara.
* La medida más relevante vinculada al desarrollo de software es ver un software que funciona y tiene calidad.
* Los procesos ágiles abogan por un desarrollo sostenible, es decir, todos los stakeholders han de poder mantener un ritmo constante durante todo el tiempo que dure el proyecto.
* Un buen diseño y procurar utilizar las mejores tecnologías en cada proyecto mejorará la agilidad.
* En la agilidad, la simplicidad de las tareas a realizar es fundamental.
* Las mejores arquitecturas, requisitos y diseños van a surgir de equipos de desarrollo que tengan una organización propia. En el desarrollo ágil tener organización propia implica tres cosas (Pressman, 2010, p. 60):
  + El equipo ágil se organiza a sí mismo para hacer el trabajo, es decir, no recibe instrucciones de cómo hacer las cosas de las jerarquías más altas.
  + El equipo ágil organiza el proceso que mejor se adapta a su entorno laboral.
  + El equipo ágil organiza la programación del trabajo para que así se obtenga la mejor implementación posible del incremento que se ha de entregar al cliente.
* El equipo de desarrollo ha de ser capaz de valorar cómo se podría ser más eficaz y actuar en consecuencia para conseguirlo.

A pesar de la sencillez aparente de los valores y principios a seguir cuando se construye software, el manifiesto ágil ha dado lugar a **malas interpretaciones,** tal y como se destaca en Garzás, Enríquez de S. y Irrazábal (2013):

* **Ausencia total de documentación.** Aunque sea de manera ágil, la documentación debe existir. Tal como se ha comentado anteriormente, la documentación sirve para ayudar a comprender mejor el sistema a implementar (o ya implementado), por tanto, aunque sea «ligera», antes de la codificación debe existir documentación que la acompañe y que recoja toda la semántica del dominio de la aplicación que está muy lejos de la semántica que recogen los lenguajes de programación. Lo que ocurre es que no hay que excederse en la documentación, tal como afirma Selic (2009): «[…] debería haber documentación de diseño ágil, que mantenga y enlace el diseño con la codificación. Y el esfuerzo realizado en documentar debería ser proporcional al tamaño del diseño» (p. 11).
* **Ausencia total de planificación.** Ser flexible a la hora de planificar no es lo mismo que improvisar. Por tanto, dado que la planificación ayuda a anticipar decisiones y a gestionar los posibles riesgos del proyecto, en todo desarrollo ágil deberán existir reuniones de planificación ágil que involucrarán a los clientes para el correcto desarrollo del sistema.
* **El cliente hará todo el trabajo y adoptará el rol de jefe de proyecto.** Es cierto que en el desarrollo ágil el cliente se involucra mucho más en el proceso de desarrollo de la aplicación. Sin embargo, el cliente adopta su propio rol dentro del desarrollo ágil, no el del jefe de proyecto. De hecho, hay que tener en cuenta que, en la mayoría de las técnicas ágiles que se siguen en las organizaciones, el rol de jefe de proyecto ni siquiera existe, ya que son los equipos de desarrollo, de manera global, los que se autogestionan.
* **El equipo de desarrollo puede cambiar de técnica o de metodología sin ninguna justificación.** Según el tipo de organización, se interesará adoptar un método u otro (Garzás, 2010). Los métodos formales o más tradicionales, como por ejemplo los basados en el modelo en cascada, serán más apropiados en organizaciones rígidas con proyectos muy estables, mientras que los métodos ágiles serán la opción más conveniente en las organizaciones más flexibles, donde los proyectos sí son cambiantes por las características y/o el mercado de la propia organización. También están los métodos iterativos-formales, que serán los más convenientes en organizaciones rígidas con proyectos cambiantes y los métodos ágiles optimizados o con algo de diseño up-front orientados en este caso a organizaciones flexibles con proyectos estables.

### **7.4. Ágil vs. tradicional**

Las técnicas ágiles introducen beneficios indudables. Sin embargo, en ocasiones, muchos de sus principios son difíciles de cumplir. En este apartado se consideran dos aspectos clave de esta problemática: la posibilidad de escalar las**técnicas ágiles a grandes proyectos** y los factores a considerar a la hora de**elegir una técnica.**

### **La escalabilidad de las técnicas ágiles**

Todas las **técnicas ágiles** nacen para ser empleadas por **equipos de desarrollo pequeños,** donde es posible la misma localización física de los participantes y el desarrollo de una comunicación oral. Sin embargo, las necesidades que cubren son comunes a los sistemas de grandes dimensiones y, por tanto, resulta interesante plantear posibilidades de escalado.

Los **sistemas de grandes dimensiones** presentan en general una serie de particularidades (Sommerville, 2011):

* **Formados por otros subsistemas comunicados entre sí,** asignados a distintos equipos distribuidos geográficamente. Es difícil que los equipos tengan una visión completa del sistema en conjunto y se comuniquen entre sí de manera fluida.
* **La conexión de diferentes sistemas.** Dificulta el enfoque de desarrollo incremental o la integración frecuente del sistema en su conjunto.
* **Suelen estar sujetos a reglas y regulaciones externas.** Estas imponen restricciones adicionales sobre los procesos permitidos y el tipo de documentación generada.
* **Su desarrollo suele abarcar períodos dilatados de tiempo.** Durante este tiempo, aumenta la posibilidad de que aparezcan cambios imprevistos.
* El **número de** stakeholders**suele ser elevado,** lo que dificulta su participación e implicación en el proceso.

Para poder mantener los **principios fundamentales** que sustentan las técnicas ágiles (flexibilidad en la planificación, integración continua y liberación frecuente o la buena comunicación entre los participantes) es necesario resolver, al menos, las siguientes cuestiones:

* Hacer énfasis en el diseño de la documentación, describiendo claramente aspectos como la arquitectura del sistema o la división del trabajo entre equipos.
* Favorecer la comunicación entre equipos, utilizando los medios tecnológicos necesarios.
* Usar herramientas de gestión de configuración que faciliten la integración continua cuando sea necesario, tras un cambio realizado.

Finalmente, las grandes organizaciones presentan una cierta oposición a este tipo de técnicas, cuyas **causas** son las siguientes:

* La reticencia por desconocimiento y falta de experiencia, al no ser empleados en el pasado.
* La existencia de normas transversales internas sobre procedimientos y estándares de calidad, que por su carga burocrática resultan difícilmente compatibles con las técnicas ágiles.

### **La elección de una técnica ágil**

¿Cuándo conviene emplear una técnica ágil en nuestro proyecto y qué factores debemos considerar?

El hecho de que las técnicas ágiles presenten una serie de ventajas no significa que su utilización sea obligada o siempre preferible. Deberemos **considerar** las características del proyecto, el perfil de los miembros del equipo, los recursos disponibles, los plazos de entrega, el entorno empresarial y social del desarrollo, etc.

Algunos aspectos que deberíamos considerar son los siguientes (Fernández, 2012):

* **El tamaño del proyecto.** Muchas de las técnicas ágiles fueron inicialmente ideadas para equipos de tamaño reducido. Aunque la mayoría disponen de extensiones a proyectos grandes, esta adaptación no resulta siempre sencilla. Cuando el proyecto implica la coordinación de muchas personas y agentes, puede ser preferible emplear técnicas de planificación y control más rigurosos.
* **La volatilidad de los requisitos.** El gran punto fuerte de las técnicas ágiles es su capacidad de adaptación al cambio, sea cual sea su origen. Cuando los requisitos del proyecto están claramente definidos desde un comienzo, y los cambios son previsibles, puede resultar más práctica una técnica tradicional.
* **La criticidad del sistema.** Ciertos sistemas, como los sistemas de control de vehículos, los relacionados con aplicaciones biomédicas o los sistemas bancarios, se prestan poco al enfoque de entregas tempranas y frecuentes que proponen las técnicas ágiles. Se desea que el sistema funcione perfectamente desde su primera puesta en producción y se dispone, en general, de una especificación de requisitos inicial muy detallada, que hace prever pocos cambios durante el desarrollo.
* **La disponibilidad del cliente.** Las técnicas ágiles hacen énfasis en la involucración del cliente en todas las fases del desarrollo. Esto no siempre es posible y, en muchos casos, es preferible negociar una participación inicial intensiva, de manera que los requisitos queden claramente definidos y formalizados, y el proyecto se ajuste a este compromiso en lo sucesivo.
* **La experiencia del equipo de desarrollo.** Las técnicas ágiles son, en gran medida, muy «elitistas». Asumen una capacidad y nivel de conocimientos por parte de los miembros del equipo que no siempre resulta asequible para las organizaciones. A pesar de que son siempre los miembros más jóvenes los que se muestran más partidarios y menos amedrentados por el empleo de estas técnicas, su nivel de madurez como programadores puede resultar un inconveniente.
* **La cultura organizacional del entorno.** Las técnicas ágiles introducen un grado de aparente informalidad que no es compatible con todas las organizaciones. La escasez de documentación formal, la manera en que se gestionan las relaciones interpersonales entre los diferentes roles o el grado de comunicación con el cliente, no siempre son viables.

### **7.5. La hoja de ruta de un proyecto ágil**

Como ya se ha comentado anteriormente, la planificación de proyectos software no constituye una tarea sencilla, lo que hace que continuamente se generen problemas con la planificación definida y se cometan errores de estimación. Cuando una organización se plantea enfrentarse a una planificación ágil, en primer lugar, debe ser consciente de las **herramientas** de las que dispone para su realización, así como del orden a seguir a la hora de hacer uso de ellas, ya que dependiendo de la magnitud del proyecto se deberán establecer distintos **niveles de abstracción** en la planificación (Garzás, 2013).

La planificación ágil plantea hasta **tres niveles de abstracción** diferentes, tal y como se muestra en la Figura 2 (Garzás, 2013):

* **Planificación de la hoja de ruta** (roadmap): con un objetivo claramente estratégico, pretende servir para coordinar las distintas divisiones o departamentos de una organización (ventas, marketing, desarrollo de negocio, etc.).
* **Planificación de la creación de una entrega** (release): donde se planifican las iteraciones necesarias para crear una entrega concreta.
* **Planificación de cada iteración del** software: constituye la parte más operativa del desarrollo del sistema y es donde se describen y estiman las tareas que se van a llevar a cabo en las semanas que dura una iteración. Las tareas constituyen especificaciones muy concretas sobre lo que se debe realizar en una iteración. Un ejemplo de tarea sería, por ejemplo, «refactorización del código».

A diagram of a release process

Description automatically generated

Figura 2. Planificación a tres niveles en un proyecto ágil. Fuente: Garzás, 2013.

Si la magnitud del **proyecto es grande,** se deberían utilizar los tres niveles de planificación. Sin embargo, para **proyectos medianos** bastarán los niveles de entrega e iteración, y para **proyectos pequeños** será suficiente con aplicar el nivel de iteración.

Profundizando un poco más en el nivel estratégico de la planificación ágil, la hoja de ruta describe la evolución del software en el tiempo. La **hoja de ruta** se utilizará siempre que se tenga que gestionar un proyecto de gran magnitud, cuando se sabe que va a evolucionar durante bastantes años y que los prototipos que se implementen normalmente no pasarán a producción. Las entregas o releasesconstituyen una versión más del software, con la particularidad de que son versiones que sí van a pasar a producción y que, por tanto, van a ser utilizadas por los usuarios reales del sistema (Garzás, 2013).

En la hoja de ruta se tendrán que reflejar todas las **entregas o pasos a producción** que se prevé que se van a realizar del software a implementar, que podrán ser por semestres o por años (Pichler, 2010).

Para definir los objetivos de una entrega, en la planificación ágil, se utilizan los **temas.** Cada tema recogerá las necesidades del sistema a nivel de hoja de ruta. Ejemplos de temas serían: «Los usuarios podrán realizar sus compras online», «Los usuarios podrán descargar los contenidos disponibles» y «Los usuarios autenticados podrán acceder a los contenidos privados del recurso educativo».

#### **Planificación de entrega (release)**

La **planificación de entrega** (release planning) pretende crear un plan para un período más largo que una iteración. Un plan de lanzamiento típico cubrirá quizás de tres a seis meses y contendrá tal vez de tres a doce o más iteraciones, dependiendo del tiempo total que se disponga para hacer el proyecto. Un plan de lanzamiento es importante por varias razones:

* Primero, ayuda al Product Owner y a todo el equipo a decidir cuánto debe ser desarrollado y cuánto tiempo llevará tener un producto entregable.
* En segundo lugar, un lanzamiento transmite expectativas sobre lo que es probable que se desarrolle y en qué plazos. Muchas organizaciones necesitan esta información porque estará relacionada con otras actividades en la planificación estratégica de la organización.
* En tercer lugar, un plan de lanzamiento sirve de guía, sobre la que el equipo de proyecto puede progresar. Sin el marco del plan de lanzamiento, los equipos se moverán sin rumbo fijo desde una iteración a la siguiente. Un plan de lanzamiento proporciona un contexto que permite integrar las iteraciones en un todo. Esta es una preocupación fundamental con cualquier proceso iterativo, no solo con procesos ágiles.

Parte del trabajo de la planificación de lanzamiento es determinar **cuánto del trabajo se puede hacer.** En algunos casos, comenzamos con una fecha y luego se verá cuánto se puede terminar. En otros casos, comenzamos con un conjunto de historias de usuario y vemos cuánto tiempo llevará desarrollarlos. En ambos casos, una vez que un equipo tiene una respuesta inicial, se evalúa según los objetivos de la organización para el proyecto.

La planificación de entrega **no tiene el objetivo** de crear un plan detallado que indique la asignación desarrolladores-historias de usuario o tareas, o la secuencia en la que el trabajo se realizará dentro de una iteración. Crear un plan con ese nivel de detalle es peligroso y confuso durante la planificación del lanzamiento. Las decisiones sobre quién hará el trabajo y la secuencia de actividades es mejor dejárselo a las personas que trabajan en esas tareas y es mejor **diferirlas** tanto como sea posible. Además, es importante recordar que los elementos de un release plannning son historias de usuario, es decir que son descripciones de funcionalidad a entregar, no tareas de ingeniería individuales a realizar.

Durante la release planning es todavía pronto y es posible que algunas historias de usuarios no se comprendan lo suficiente como para desagregarlas en tareas de ingeniería. Como veremos a continuación, en la planificación de cada iteración, el equipo eventualmente **desagrega historias de usuario** del plan de liberación en sus tareas constitutivas, pero ellos no harán esto hasta el comienzo de la iteración que contiene esas historias.

### **Planificación de cada iteración**

Un plan de release ofrece una vista excelente de alto nivel de cómo un equipo de proyecto **pretende entregar el nuevo producto.** Sin embargo, un plan de release proporciona precisamente solo eso, una vista de muy alto nivel del producto que se está construyendo. No proporciona la vista breve y detallada que los equipos utilizan para conducir el trabajo que se produce dentro de una iteración.

Con un plan de iteración, el equipo analiza de manera más detallada y está enfocado a lo que será necesario para implementar completamente solo las historias de usuario seleccionadas para cada nueva iteración.

El plan de iteración se crea en una **reunión de planificación de iteración.** A esta reunión deben asistir el Product Owner, analistas, programadores, evaluadores, ingenieros de bases de datos, diseñadores de interacción con el usuario, etc. Cualquier persona involucrada en tomar una idea en bruto y convertirla en un producto funcional debería estar presente.

Un plan de iteración puede ser tan simple como una hoja de cálculo o un conjunto de

notas. Una de las ventajas de usar cartas o información visual de la planificación de iteración es que permite que **cualquiera pueda participar** en el proceso.

Mientras planeamos una interacción, las tareas no se asignan a los desarrolladores de forma individual. Al comienzo de la iteración puede parecer obvio quién trabajará en cada tarea, sin embargo, basándonos en el progreso del equipo completo respecto al conjunto de tareas, lo que es obvio al principio podría no ser lo que ocurre durante la interacción. Por ejemplo, cuando se planea una interacción asumimos que nuestro administrador de base de datos completará la tarea «optimizar el motor de búsqueda» porque sabemos que él tiene los conocimientos necesarios en SQL de todo el equipo. Sin embargo, si no puede hacerlo, por el motivo que sea, algún otro miembro del equipo podrá hacerlo.

Por último, hay que tener en cuenta las **diferencias** entre los dos niveles de planificación. Mientras que la planificación de entrega intenta alcanzar el lanzamiento de un nuevo producto, por lo general en un periodo de tres a seis meses desde el inicio del proyecto, el plan de iteración muestra la visión del resultado en un periodo de una a cuatro semanas. Las historias de usuario del plan de release se descomponen en tareas en el plan de iteración. Mientras que las historias de usuario se estiman en puntos de historia o días ideales, las tareas en el plan de iteración se estiman en **horas ideales.**

### **7.6. Product owner**

El propietario del producto o product owner, que es como se le conoce más, representa a la persona que tiene una visión muy clara y conocimiento profundo del software que se va a desarrollar, y que es capaz de **transmitir esa visión** a todo el equipo de desarrollo. Además, tiene la disponibilidad suficiente para transmitirla siempre que sea necesario (Garzás, 2013). De este rol depende en gran medida que el proyecto tenga éxito y no se debe subestimar, ya que, si no realiza correctamente su labor, el proyecto puede dejar de ser ágil o, lo que es peor, estar abocado al fracaso. Aparte de esa responsabilidad general, el product owner también tiene las siguientes responsabilidades:

* Gestionar la comunicación entre todos los stakeholders.
* Gestionar el trabajo que se ha de realizar.
* Determinar el orden de las entregas.
* Determinar las funcionalidades que se tienen que contemplar en cada entrega.

La Tabla 1 resume todas las responsabilidades del Product Owner en un proyecto ágil.

A screenshot of a computer

Description automatically generated

Tabla 1. Responsabilidades del Product Owner. Fuente: Adaptado de Garzás, 2013.

El rol de Product Owner lo suele adoptar normalmente uno de los futuros usuarios del sistema que se está desarrollando o alguien de marketing, alguien que, sin ser un usuario final, tiene un **conocimiento elevado** de lo que se necesita del sistema (Garzás, 2013).

### **7.7. Historias de usuario**

El término «historia de usuario» tiene su origen en el **modelo de proceso ágil XP** o programación extrema (Beck y Andres, 2004) y se ha adaptado muy bien al resto de marcos de trabajo ágiles como puede ser Scrum (Schwaber, 2004). La historia de usuario fue definida como «la unidad más pequeña posible de valor para el negocio» (Beck y Andres, 2004).

Las historias de usuario en un desarrollo ágil representan el **mecanismo para recoger los requisitos** que el sistema ha de implementar. Normalmente, las historias de usuario adoptan el siguiente patrón (Garzás, 2013), aunque también resulta común encontrar otras variantes:

«Como (rol o tipo de usuario), (quiero | podré | seré capaz | necesitaré |...) con el beneficio de...» (p. 22).

Además, las historias de usuario suelen escribirse en tarjetas o notas tipo post-it que describen de manera muy breve la funcionalidad que se ha de satisfacer, es decir, indican la funcionalidad que han de desarrollar, pero no especifican cómo se ha de desarrollar ni tampoco hacen referencia a los requisitos no funcionales, por lo que historias de usuario que describan aspectos de diseño como, por ejemplo, «El sistema se desarrollará utilizando el lenguaje de programación C#» o «El sistema soportará 100 usuarios concurrentes» no serían historias de usuario correctas. Para poder recoger este tipo de información habrá que recurrir a documentación adicional que estará vinculada a las historias de usuario.

Aunque los requisitos tradicionales tratan de ser lo más detallados posible, una historia de usuario se definirá de forma incremental en varias etapas como: en primera instancia, como una breve descripción de la necesidad; en segundo, a partir de las conversaciones durante planificación y la iteración para estabilizar detalles y, en una tercera etapa, como a partir de unas pruebas que confirman la finalización satisfactoria de la historia de usuario.

La Tabla 2 muestra unos ejemplos de lo que sí se considerarían historias de usuario.

A screenshot of a web page

Description automatically generated

Tabla 2. Ejemplos de historias de usuario Fuente: PMOinformatica, 2015.

Sin embargo, una **historia de usuario** no representa solamente la descripción de una funcionalidad en un post-it, sino que lleva asociados otros dos aspectos de gran importancia (Jeffries, 2001):

* La **conversación** que lleva de manera implícita, ya que las historias de usuario constituyen una herramienta para que los stakeholders puedan interactuar.
* El **determinar cómo se confirma su implementación,** las pruebas a realizar y la verificación de las mismas.

Sin embargo, hay que tener cuidado y no confundir una historia de usuario con un **requisito** software. Una historia de usuario nunca dará tanto nivel de detalle ni tanta información como la que proporciona un requisito software. De ahí que se recomiende escribir las historias de usuario en post-it o tarjetas de tamaño reducido, para que se intente **expresar la funcionalidad** en poco más de una frase, fácil de memorizar y que pueda ser programada de manera sencilla y en poco tiempo, de tal forma que no se necesite más de una iteración para su implementación.

No obstante, como ya se ha comentado anteriormente, se puede hacer uso de la **trazabilidad** para enlazar cada historia de usuario con documentación adicional de soporte que permita ampliar información de la historia de usuario sin dejar de ser ágil. Para ello, se suele poner un **identificador** a cada historia de usuario y será el identificador al que haga referencia toda la documentación adicional enlazada a esa historia de usuario.

Por otro lado, tampoco hay que confundir las historias de usuario con los **casos de uso,** aunque los casos de uso pueden servir también como información adicional a las historias de usuario. Mientras las historias de usuario ayudan a entender «qué» es lo que quiere el cliente, los casos de uso puede ayudan a entender «cómo» lo quiere desde un punto de vista de análisis del sistema, es decir, con los casos de uso, los desarrolladores conseguirán entender qué es lo que debe hacer el software exactamente, porque los casos de uso están describiendo su comportamiento, así como la interacción del sistema con los usuarios u otros sistemas (Garzás, Enríquez de S. y Irrazábal, 2013).

Por último, tampoco se deberían confundir las historias de usuario con las **tareas,** ya que las tareas describen de manera sencilla el trabajo que se ha de realizar para implementar una parte de una historia de usuario, por lo que para completar una historia de usuario se deberán realizar varias tareas (Garzás, 2013, p. 34).

Además de las historias de usuario, en el desarrollo ágil también se puede hacer uso de lo que se denomina **epopeyas**.

Una epopeya es «una historia de usuario de mayor tamaño y mayor funcionalidad, que sigue el mismo formato que una historia de usuario, pero con mayor alcance» (Garzas, 2013, p. 62).

 Las epopeyas se encuentran a un **mayor nivel de abstracción** que las historias de usuario y con un objetivo más estratégico (Garzás, 2013). La Tabla 3 recoge lo que sería un ejemplo de epopeya, las historias de usuario en las que se descompone y las tareas necesarias para implementar cada historia de usuario.

A table with text on it

Description automatically generated

Tabla 3. Epopeyas, historias de usuario y tareas. Fuente: Garzás, 2013.

### **7.8. Planificación y estimación ágil**

A la hora de planificar un proyecto software, lo primero que hay que llevar a cabo, también en la planificación ágil, es **determinar la lista de funcionalidades** que el sistema debe implementar. Como ya se ha mencionado anteriormente, con las técnicas ágiles esta lista va a tomar la forma de historias de usuario que van a permitir que se vaya dando más valor al negocio de una manera incremental e individual (Garzás, Enríquez de S. y Irrazábal, 2013). Una vez que las historias de usuario se encuentran disponibles para su procesamiento, el siguiente paso dentro de la planificación sería darles **prioridad** teniendo en cuenta los siguientes criterios (Garzás, Enríquez de S. y Irrazábal, 2013):

* **Valor.** Este valor, que puede ser numérico o no, se corresponde con la importancia que les da el cliente o usuario final a las historias de usuario. Este valor, que le asigna el cliente o usuario, debería ir incrementando en cada iteración del proceso de desarrollo para que el proyecto vaya por un buen camino. El valor de una historia de usuario se puede calcular en base a:
  + Los beneficios que se obtienen con su implementación.
  + Las pérdidas que se podrían producir en caso de posponer su implementación.
  + Los riesgos que tiene asociados.
  + Su alineación con el negocio.
  + El valor añadido que puede aportar con respecto a productos similares de la competencia.

De esta manera, por ejemplo, se implementarán primero las historias de usuario que lleven menos tiempo, con menos riesgos asociados, más estratégicas y con mayor facturación al cliente.

* **Estimación.** Se debe estimar el coste de desarrollo en unidades que representan el tiempo teórico de desarrollo/persona que se ha definido al comiendo del proyecto.

La Figura 3 ilustra los conceptos de valor y estimación a nivel de historia de usuario.

A yellow note with red push pin

Description automatically generated

Figura 3. Ejemplos de valor y estimación en una historia de usuario representada en un pósit. Fuente: Garzás, Enríquez y Irrazábal, 2013.

### **Planning poker**

Respecto a la estimación, una técnica muy utilizada en los proyectos ágiles por su simplicidad y bajo coste es la que se conoce como planning poker (Cohn, 2005). Esta técnica **se basa en el consenso,** de forma que, cada una de las personas que participan en el proceso de estimación toma una baraja de cartas, que suelen estar numeradas siguiendo (normalmente pero no tiene por qué) la secuencia Fibonacci, como una forma de reflejar la incertidumbre inherente a la estimación de las tareas en un desarrollo de software.

El número de cartas no ha de ser elevado y cuando se tengan dudas sobre el **número de puntos** que se le debería asignar a una historia de usuario, se le asignará el número de la carta que se le acerque más. De esta manera, por ejemplo, si se estima que a una historia de usuario se le debería asignar 7 puntos y no hay ninguna carta con el número 7, pero sí con el número 8, entonces se elegirá la carta marcada con el número 8.

Además, en la técnica planning poker, se utiliza una carta adicional con una interrogación que representa el caso en el que se considera que falta información necesaria relativa a la historia de usuario como para poder estimarla, y otra carta adicional con una taza de café que puede ser mostrada por cualquier miembro del equipo de desarrollo cuando necesite realizar un descanso durante el proceso de estimación (Garzás, Enríquez de S. y Irrazábal, 2013, p. 66-67). En los proyectos ágiles, son los puntos de historia, y no los puntos de función, la unidad que se utiliza para medir el tamaño de las funcionalidades o requisitos. Cohn (2005), define un**punto de historia** como «una fusión de la cantidad de esfuerzo que supone desarrollar una historia de usuario, la complejidad de su desarrollo y el riesgo inherente».

Normalmente, los puntos de historia representarán **horas de trabajo,** por lo que cada organización determinará las que va a considerar en su caso, aunque resulta bastante frecuente que cada punto de historia represente **un día** de jornada laboral. La Figura 4 muestra un ejemplo de las cartas que se pueden utilizar en una estimación con planning poker y la Figura 5 identifica las etapas a realizar durante el proceso de estimación con la técnica planning poker.

A blue rectangular box with numbers

Description automatically generated

Figura 4. Ejemplo de mazo de cartas para la técnica de estimación planning poker. Fuente: Garzás, Enríquez y Irrazábal, 2013.

A diagram of a business diagram

Description automatically generated

Figura 5. Etapas de la técnica de estimación planning poker. Fuente: Garzás, Enríquez y Irrazábal, 2013.

Planning poker constituye un **proceso que es iterativo** y se puede describir a grandes rasgos de la siguiente forma:

* En primer lugar, el product owner lee una historia de usuario que será objeto de la estimación.
* En segundo lugar, el equipo le hará preguntas si procede para poder entender mejor el alcance de la historia de usuario.
* En tercer lugar, cada miembro del equipo pensará acerca del esfuerzo necesario para completar la historia de usuario que se está tratando.
* En cuarto lugar, todos los miembros que participan en la estimación muestran sus cartas de estimación simultáneamente.

Los miembros del equipo que hayan realizado las **estimaciones más extremas** numéricamente (es decir, más bajas o más altas) del conjunto deberán explicar el porqué de su decisión. El motivo de requerir esta explicación es que pueden haber visto un problema de mayor magnitud que el resto o, por el contrario, haber observado menos problemas que el resto y es interesante por tanto debatirlo.

Es común que en este punto las estimaciones difieran bastante. Se procede a continuación a votar otra vez hasta que se alcanza un acuerdo. En muchos casos, las

estimaciones convergen en la segunda vuelta, pero si no la hacen se continúa el proceso. El objetivo es que los estimadores **convergen en una sola estimación** que pueda usarse para la historia. Rara vez se usan más de tres rondas, pero continúa el proceso siempre que las estimaciones se aproximen. No es necesario que todos den la misma estimación. Si se está moderando una estimación y, en la segunda ronda, cuatro estimadores dicen 4, 4, 4 y 5, se preguntará al estimador más alto si está bien con una estimación de 5. Una vez más, no se trata de buscar precisión absoluta sino una aproximación sensata.

En una planificación planning poker intervienen todos los desarrolladores del equipo, existen diversos estudios que concluyen que la **sinergia** que produce esta estimación conjunta es mucho mejor que la estimación que realiza una única persona por separado (típicamente la del product owner) desvinculada del esfuerzo necesario que conlleva la implementación de cada historia de usuario.

El objetivo de la planificación planning poker no es obtener una estimación que resista todo, más bien, es alcanzar una estimación equilibrada entre esfuerzo y precisión.

Los equipos deberán estimar usando planning poker dos veces en el ciclo de desarrollo ágil:

* Normalmente, al comienzo del proyecto, se hará un gran esfuerzo para estimar una gran cantidad de elementos (ítems de proyecto) antes de que el proyecto comience oficialmente o durante sus primeras iteraciones. El estimar un conjunto inicial de historias de usuario puede llevar dos o tres reuniones de una a tres horas. Naturalmente, esto dependerá de cuántos elementos haya que estimar, del tamaño del equipo y de la capacidad del product owner para aclarar las necesidades.
* En segundo lugar, los equipos deberán realizar un esfuerzo continuo para estimar cualquier nueva historia de usuario que se identifique durante cada iteración. Una forma de hacer esto es planificar una reunión de estimación muy corta cerca del final de cada iteración. Normalmente, esto es suficiente para estimar cualquier trabajo que se realizó durante la iteración y permite que se considere el trabajo nuevo en la priorización de la próxima iteración.

Planning poker es una técnica de **estimación de historias** **de usuario** que funciona bien por las siguientes razones:

* En primer lugar, planning poker reúne las opiniones de múltiples expertos. Debido a que estos expertos forman un equipo multifuncional, estarán mejor preparados para la tarea de estimación que cualquier otro. Otro de los fundamentos de su buen funcionamiento se basa en que «las personas más competentes para resolver la tarea son quienes deberían estimarla». Esta premisa contradice la idea tradicional de que una persona puede estimar convenientemente sin ser ella parte del equipo de desarrollo.
* En segundo lugar, en planning poker se produce un diálogo durante la planificación, en donde los estimadores tienen que justificar sus estimaciones. Se ha encontrado que esto mejora la precisión de la estimación, especialmente en ítems de proyectos con gran cantidad de incertidumbre. El hecho de que justifiquen las estimaciones también resulta apropiado en estimaciones donde la información es escasa. Esto es importante en un proyecto ágil porque las historias de usuario que se estiman son a menudo intencionalmente vagas.
* En tercer lugar, los estudios han demostrado que promediar las estimaciones individuales conduce a mejores resultados, al igual que las discusiones grupales de estimaciones. La discusión grupal es la base del planning poker y esas discusiones conducen a un promedio de los tipos de estimaciones individuales.

En la actualidad, existen multitud de aplicaciones web y móviles que trabajan con la técnica planning pokerde forma online.

### **Técnicas de priorización**

Además de la técnica planning poker, existen otras **técnicas** que pueden utilizarse para la priorización de historias de usuario y que, en muchos casos, también se utilizan en la priorización tradicional de requisitos.

**Dentro o fuera** (in or out)

Lo más sencillo es proporcionar una **lista de historias de usuario** a los participantes y pedirles que tomen una decisión binaria para cada uno de ellos: ¿está dentro o está fuera? Para tomar su decisión deben tener presentes los objetivos de la organización, intentando recortar la lista a la cantidad mínima de funcionalidades necesarias para la iteración o el release en curso. De esta manera, es posible obtener un conjunto común de necesidades identificadas por todos ellos. Esta técnica es apropiada para utilizarla durante un workshop y cuando la cantidad de historias de usuario es muy grande, porque permite un filtrado inicial.

**Comparación por pares**

Generalmente es difícil asignar una prioridad absoluta a cada historia de usuario considerada en un conjunto grande. Es más eficiente establecer un ranking mediante **comparaciones dos a dos,** de manera que es posible evaluar la importancia relativa entre ellos. Algunos métodos de ayuda a la decisión, como el proceso analítico jerárquico (AHP) se pueden utilizar en este caso (Berumen y Llamazares, 2007). No es recomendable su utilización para grandes conjuntos de historias de usuario.

**MoSCoW**

Esta técnica de priorización está compuesta de una escala de cuatro niveles (IIBA, 2015):

* Must(debe): se debe implementar la historia de usuario para que la solución se considere exitosa.
* Should (debería): la historia de usuario es importante y debería incluirse en la solución si resulta posible, pero no es obligatorio para su éxito.
* Could (podría): la historia de usuario expresa una capacidad deseable, pero no es urgente su implementación e incluso podría ser eliminada. Se debe implementar solo si el tiempo y los recursos lo permiten.
* Won’t(no se hará): se indican así aquellas historias de usuario que no se consideran necesarias en este momento, aunque pueden reservarse para una futura versión del producto.

Esta técnica no proporciona guías específicas sobre cómo establecer la distinción entre las historias de usuario y, para algunos autores, resulta ambigua y consideran que no debe ser utilizada. En particular, **introduce ambigüedad** por la propia terminología empleada. No queda claro qué es un requisito de la categoría «wont’t». Si no es necesario, parece que no debería formar parte de la especificación.

**La técnica de los 100 dólares o de la puja**

Como hemos comentado, la priorización de las historias de usuario tiene mucho que ver con **aumentar el retorno** de la inversión de un producto (valor). Se trata de dar mayor importancia a aquellas capacidades más valiosas para la organización o sus usuarios. Una manera de reflejar esta valoración es emplear términos económicos.

La técnica consiste en repartir **100 dólares** a cada uno de los participantes en un workshop de priorización. Cada uno de ellos tiene que distribuir su dinero entre las historias de usuario presentadas, dando más valor a aquellas que quieran «comprar», porque las consideran más prioritarias. Si un usuario considera una historia de usuario el doble de importante que otra, puede asignar 10 dólares a la primera y 5 a la segunda. Pero esta es la cantidad máxima que puede emplear para pujar por las historias de usuario. La priorización global se alcanza ordenando las historias de usuario según la cantidad total que el conjunto de participantes está dispuesto a pagar por cada una de ellas.

Esta técnica presenta algún problema:

* Si un usuario desea realmente alguna de las características, puede utilizar todo su dinero (los 100 dólares) para «comprarla». De esta manera, se introduce una desviación en el proceso, puesto que, siendo objetivos, nadie estaría dispuesto a desarrollar un producto con una única característica y puede dejar fuera historias de usuario que son más valiosas para un conjunto de usuarios.
* No tiene en cuenta el esfuerzo real que supone el desarrollo de las historias de usuario en términos relativos. Por ejemplo, si supone el mismo esfuerzo el desarrollo de tres historias de usuario, con un valor de 10 dólares cada una, que el de una única historia de usuario, valorada en 15 dólares, parece que sería preferible desarrollar las tres primeras, aunque sus valoraciones individuales sean menores.

Una extensión de esta técnica, mucho más sofisticada, consiste en asignar **valoraciones económicas reales**a cada una de las historias de usuario. Se trata de estimar en dinero real el valor que representa cada característica del producto, desde el punto de vista de los objetivos de la organización.

Quality Function Deployment

La técnica QFD (Quality Function Deployment) evalúa cada historia de usuario desde diferentes puntos de vista para obtener una **valoración global** de cada una, que permite priorizarla dentro del conjunto (Cohen, 1995).

Para aplicar el método se deben seguir los siguientes pasos:

* **Seleccionar las historias de usuario sujetas a priorización.** De entrada, puede haber historias de usuario cuya prioridad es claramente máxima. Aquellas que se refieren a actividades principales o fundamentales del producto, o a elementos requeridos por regulaciones externas. De esta manera, se puede realizar un filtrado previo de las historias de usuario, manteniendo solo aquellas con las cuales existen dudas.
* **Escribir la lista de historias de usuario en la primera columna de una tabla.** En cada tabla conviene utilizar historias de usuario homogéneas, pertenecientes al mismo nivel jerárquico dentro de la especificación. En el caso de que existan dependencias entre las historias de usuario (por ejemplo, la implementación de B requiere que se haya implementado previamente A), se debe incluir solo la historia de usuario independiente (A).
* **Estimar el beneficio relativo de cada historia de usuario respecto de los objetivos del producto.**Emplear una escala de uno (no supone ninguna utilidad o beneficio) a nueve (la característica es extremadamente valiosa).
* **Estimar la penalización relativa que el cliente percibiría por cada historia de usuario,** en caso de que no se incluyera en el producto. Emplear una escala de uno (no es un problema que la característica no se incluya) a nueve (si la característica no está presente se produce un grave perjuicio).
* **Calcular el valor total de cada historia de usuario como la suma ponderada del beneficio relativo y la penalización relativa.** Es necesario definir estas ponderaciones o pesos, que representan que tan importante consideramos un atributo con respecto a otro.
* **Estimar el coste de implementación de cada historia de usuario con ayuda del equipo de desarrollo.**Una vez más se emplea una escala de 1 (la implementación es rápida y sencilla) a 9 (la implementación es enormemente costosa en términos de tiempo y recursos). Este parámetro está muy ligado a los puntos de historia de cada característica.
* **Estimar el riesgo técnico de implementación de cada historia de usuario con ayuda del equipo de desarrollo.** Emplear nuevamente la escala de 1 (la historia de usuario no presenta ninguna dificultad y es de implementación inmediata) a 9 (que indica que existen riesgos severos en su implementación). Los riesgos técnicos están relacionados con aspectos como falta de experiencia del equipo, la necesidad de emplear tecnologías o herramientas desconocidas, o la complejidad algorítmica de una determinada funcionalidad.
* **Calcular el valor de la prioridad de cada historia de usuario,** que debe ser directamente proporcional al valor porcentual de cada historia de usuario dentro del conjunto e inversamente proporcional a la suma ponderada del coste porcentual y del riesgo porcentual de cada historia de usuario.

Aparte de estas técnicas de priorización, existen otras técnicas que también son usadas para la **priorización de historias** **de usuario** en desarrollos ágiles (Garzás, 2015; Pérez, 2021):

* Filtro de priorización.
* Pirámide de priorización.
* Modelo kano.
* Clasificación de lista.
* Business value y story points.
* Urgente.
* Basados en riesgos.

### **7.9. Mediciones en desarrollos ágiles**

Cuando una organización se plantea adoptar un enfoque ágil para desarrollar software, siempre surge la duda de qué tipo de **mediciones** se deberían utilizar para este tipo de proyectos. La Tabla 4 resume las **métricas, gráficos y herramientas** más relevantes utilizadas para realizar mediciones en proyectos ágiles de desarrollo de software y que han sido descritas por Garzás, Enríquez de S. y Irrazábal (2013).

Un **cuadro de mando ágil** incluiría todas estas métricas, gráficos y herramientas con el fin de agrupar en un mismo sitio todo aquello que facilite el trabajo del equipo de desarrollo y su comunicación con el cliente, de cara a un buen funcionamiento y control del proceso de desarrollo de software. La Figura 6 muestra un ejemplo de cuadro de mando ágil.

A close-up of a graph

Description automatically generated

Figura 6. Ejemplo cuadro de mando ágil. Fuente: Garzás, Enríquez y Irrazábal, 2013.

A close-up of a document

Description automatically generated

Tabla 4. Métricas más utilizadas cuando se realizan proyectos ágiles. Fuente: Garzás, Enríquez y Irrazábal, 2013.

A close-up of a screen

Description automatically generated

Figura 7. Ejemplo de salida mostrada por una herramienta Impediment backlog. Fuente: Garzás, Enríquez y Irrazábal, 2013.

Por último, mencionar la existencia de una amplia variedad de **herramientas** para ayudar en la planificación y gestión de proyectos ágiles de desarrollo de software.De todas ellas, la más utilizada y recomendada es **Atlassian Jira** (Atlassian, 2022), como puede verse en la Figura 8.

A graph of a bar chart

Description automatically generated with medium confidence

Figura 8. Recomendación en el uso de herramientas ágiles de planificación en base a la experiencia. Fuente: Digiital.ai, 2021.