TEMA 9

Metodologías ágiles para el desarrollo de *software*

¿Qué es la agilidad en el *software*?

→Proceso iterativo e incremental.
→Respuesta a los cambios constantes de *software*.
→Mayor relevancia a las personas.

La hoja de ruta de un proyecto ágil

→Carácter estratégico.
→Entregas o pasos a producción que se prevén que se van a realizar.

El *Product Owner*

Planificación y estimación ágil

→Valor.
→Estimación.

El manifiesto ágil

→Establecimiento de valores y principios que permitan el desarrollo de software de manera rápida y buena adaptación a los requisitos cambiantes.
→Cuatro valores.
→Doce principios.

Las historias de usuario

→Funcionalidades que el sistema ha de implementar.

Mediciones en desarrollos ágiles

→Productividad: velocidad y aceleración.
→Progreso del proyecto: gráficos de progreso y RTF.
→Valor entregado y retorno de inversión: EVM-Ágil.
→Gestión de riesgos: *impidemientbacklog*.
→Cuadro de mando ágil.

9.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** que encontrarás a continuación.

En este tema se estudian metodologías ágiles que permiten 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. Se analizará en qué casos merece la pena adoptar el enfoque ágil frente a un enfoque más tradicional.

9.2. ¿Qué es la 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, se sigue inmerso en una «crisis del *software*». El porcentaje de fracaso de los proyectos sigue siendo demasiado elevado y se siguen incumpliendo plazos, 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 metodologías.

En la sociedad actual, donde cada vez de manera más rápida surgen nuevas necesidades, tecnologías, etc., las metodologías ágiles podrían ser una buena opción para dirigir y gestionar cierto tipo de proyectos *software*. Tal y como afirma Kent 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».

En este sentido, las metodologías ágiles se adaptan muy bien a proyectos donde los requisitos cambian constantemente, y cuando los tiempos de desarrollo no pueden extenderse demasiado en el tiempo porque para cuando estuvieran operativos ya estarían obsoletos. Las metodologías ágiles siguen el **modelo de proceso iterativo e incremental** que ya se estudió en el tema 2 (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.Enfoque iterativo

B. Enfoque incremental

Figura 1. Iterativo vs Incremental. Fuente: http://jan-so.blogspot.com.es/2008/01/difference-betweeniterative-and.html

Aparte de los ya vistos en el tema 2 en lo que respecta a otros modelos de procesos, los beneficios más significativos que va a aportar el enfoque conjunto iterativo+incremental a una metodología ágil serían los que se citan a continuación (del curso gratuito *online CIOÁgil, LECCIÓN 2: El ciclo de vida iterativo e incremental*):

- **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 riesgos 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** vinculada 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, 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 muy limitado).

9.3. 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 a día de hoy, como por ejemplo, *Extreme Programming (XP), Scrum, Dynamic Systems Development Method* (DSDM) y Crystal. El objetivo de este 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 una plan».

Por su parte, **los principios vinculados** al *Manifiesto Ágil* son los siguientes:

- 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 valor añadido para el cliente.
- El periodo en el que el cliente debería recibir *software* operativo 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, 60)
- La forma más eficaz y eficiente de transmitir la información a los miembros del equipo de desarrollo es la que se lleva a cabo de manera presencial.
- 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, 60): (i) 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; (ii) el equipo ágil organiza el proceso que mejor se adapte a su entorno laboral; y (iii) 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*, sin embargo, ha dado lugar a malas interpretaciones tal y como se destaca en (Garzás, Enríquez de S. e Irrazábal, 2013):

- **Ausencia total de documentación**. Aunque sea de manera ágil, pero la documentación debe existir. Precisamente 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 ya tal y como afirma Bran Selic: «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».
- **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.** En el desarrollo ágil es cierto que 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 de jefe de proyecto. De hecho, hay que tener en cuenta de que en la mayoría de las metodologías 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 metodología sin ninguna justificación.** Según el tipo de organización le interesará adoptar un método u otro (Garzás, 2010). Los métodos formales o más tradicionales, como los basados en el modelo en cascada, por ejemplo, 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 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.

9.4. 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): (i) **planificación de la hoja de ruta** (del inglés *roadmap*), con un objetivo claramente estratégico, pretende servir para coordinar las distintas divisiones o departamentos de una organización (ventas, *marketing*, desarrollo de negocio, etc.); (ii) **planificación de la creación de una entrega** (del inglés *release*), donde se planifican las iteraciones necesarias para crear una entrega concreta; y (iii) **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».

Figura 2. Planificación a tres niveles en un proyecto ágil (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 *releases* constituyen una versión más del *software*, con la particularidad de que son versiones que sí que 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 prevén 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, por ejemplo, «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».

9.5. El *Product Owner*

El propietario del producto o *Product Owner*, que es como más se le conoce, 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 que se le asignan al *Product Owner* en un proyecto ágil.

Tabla 1: Responsabilidades del *Product Owner*. Fuente: adaptado de Garzás, 2013, 21.

#	Responsabilidad
1	Escribir historias de usuario totalmente inteligibles, sin ambigüedades ni contradicciones.
2	Decidir qué construir… ¡y qué no!
3	Fijar criterios de aceptación para cada historia de usuario.
4	Validar las entregas antes de enviarlas a los usuarios.
5	Definir **productos mínimos viables** (del inglés *Minimum Viable Product*, MVP).
6	Priorizar historias de usuario.
7	Tener disponibilidad para poder explicar al equipo de desarrollo todas las dudas funcionales que pueda tener.
8	Definir la planificación ágil a cualquier nivel de abstracción.

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).

9.6. Las historias de usuario

El término «historia de usuario» tiene su origen en el modelo de proceso ágil XP o **programación extrema** como también se le conoce en castellano (Beck y Andres, 2004), y se ha adaptado muy bien al resto de técnicas ágiles como puede ser Scrum (Schwaber, 2004), que se estudiará en el siguiente tema. 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, 22), aunque también resulta común encontrar otras variantes:

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

Además, las historias de usuario suelen escribirse en tarjetas o notas tipo *post-it* que describirán 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. La Tabla 2 muestra unos ejemplos de lo que sí se consideraría historias de usuario.

Tabla 2: Ejemplos de historias de usuario. Fuente: http://www.pmoinformatica.com/2015/05/historias-deusuario-ejemplos.html .

#	Historia de usuario
1	Como **vendedor**, quiero registrar los productos y cantidades que me solicita un cliente para crear un pedido de venta.
2	Como **supervisor de ventas**, quiero consultar un listado de los pedidos de venta que han sido registrados y aún no han sido procesados.
3	Como **gerente de ventas**, quiero consultar los pedidos de venta procesados clasificándolos por vendedor, región y líneas de producto.
4	Como **analista de logística**, quiero seleccionar un pedido de venta y enviarlo al almacén para que procedan con su preparación.
5	Como **analista de almacén**, quiero listar todos los pedidos de venta recibidos que debo preparar.
6	Como **analista de logística**, quiero poder consultar todos los pedidos preparados listos para ser despachado.

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 llevan 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: 32-33).

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, 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). Las epopeyas se encuentra 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.

Tabla 3: Epopeyas, historias de usuario y tareas. (Garzás, 2013, 22)

Epopeya	Historias de usuario en las que se descompone	Tareas requeridas para implementar la historia de usuario
Como usuario, quiero comprar artículos.	Como usuario, quiero seleccionar el artículo a comprar.	Diseño de la historia. Implementación. Pruebas. Aceptación. Demo. Refactorización.
	Como usuario, quiero seleccionar unidades.	...
	Como usuario, quiero seleccionar la forma de pago.	...
	Como usuario, quiero seleccionar la forma de envío.	...

9.7. 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 metodologías 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 le 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: (i) los beneficios que se obtienen con su implementación; (ii) Las pérdidas que se podrían producir en caso de posponer su

implementación; (iii) los riesgos que tiene asociados; (iv) su alineación con el negocio; y (v) 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 comienzo del proyecto.

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

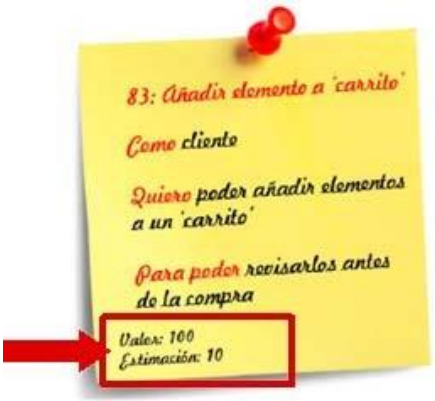

Figura 3. Ejemplos de valor y estimación en una historia de usuario representada en un *post-it* (Garzás, Enríquez de S. e Irrazábal, 2013)

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 porqué, 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. e Irrazábal, 2013, 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*.

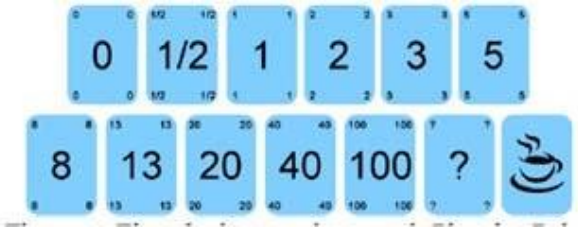

Figura 4. Ejemplo de mazo de cartas para la técnica de estimación *Planning Poker* (Garzás, Enríquez de S. e Irrazábal, 2013).

Figura 5. Etapas de la técnica de estimación *Planning Poker* (Garzás, Enríquez de S. e Irrazábal, 2013).

Para ver cómo convergen las estimaciones iniciales realizadas con *Planning Poker*, la Figura 6 ilustra un ejemplo en el que un equipo formado por cuatro desarrolladores (D1, D2, D3 y D4) ha de implementar tres historias de usuario. Por cada historia de usuario se irán haciendo turnos hasta que todos los desarrolladores alcancen un consenso respecto al número de puntos de historia que se necesitan para implementarla. Cada

desarrollador, en cada turno, realizará su exposición para justificar el número de puntos que ha asignado a la historia de usuario que se está tratando. Así seguirá el proceso hasta que el resto de desarrolladores converjan a la estimación realizada por uno de ellos.

Historia de usuario 1				
Turno	D1	D2	D3	D4
1	2	5	6	10
2	5	7	9	9
3	6	8	8	9
4	8	8	8	9
Final	8 puntos de historia			
Historia de usuario 2				
Turno	D1	D2	D3	D4
5	8	3	3	8
6	7	7	7	7
Final	7 puntos de historia			
Historia de usuario 3				
Turno	D1	D2	D3	D4
7	7	2	6	14
8	7	5	6	10
9	7	7	6	7
Final	7 puntos de historia			

Figura 6. Ejemplos de estimaciones *Planning Poker* (Garzás, Enríquez de S. e Irrazábal 2013).

9.8. 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. e 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 7 muestra un ejemplo de cuadro de mando ágil.

Tabla 4: métricas más utilizadas cuando se realizan proyectos ágiles (Garzás, Enríquez de S. e Irrazábal, 2013).

Clasificación	Indicador	Descripción
Productividad	Velocidad	Cantidad de trabajo que el equipo de desarrollo puede realizar en una iteración del proyecto. Se debe indicar el ratio con el cual el equipo convierte historias de usuario en incrementos potencialmente entregables. Así, se va viendo la velocidad estimada y la velocidad real que tiene el equipo. Una vez conocida la velocidad del equipo de desarrollo se puede utilizar para estimar en la siguiente iteración del proyecto. Además, con el histórico de la velocidad del equipo se puede ir ajustando la estimación del proyecto para que sea más realista.
	Aceleración	Tasas de crecimiento, o de decrecimiento, de las velocidades del equipo de desarrollo respecto a un periodo de referencia en base a la siguiente fórmula: $$Aceleración = \frac{(velocidad\ iteración\ (n) - velocidad\ iteración\ (0))}{velocidad\ iteración\ (0)}$$
Progreso del proyecto	Gráficos de progreso	Gráfico que refleja el avance del equipo de desarrollo hasta un momento concreto. Se suele medir el número de historias que se llevan implementadas hasta ese momento, en relación al esfuerzo total que hace falta para poder terminar el proyecto. Permite apreciar lo que falta para implementar el sistema en su totalidad.
	Funcionalidades probadas y aceptadas (del inglés *Running Tested Features*, RTF)	Cantidad de funcionalidades que han superado las pruebas de aceptación. Este indicador favorece también la agilidad y productividad del equipo de desarrollo. Para que realmente sea útil es importante que las historias de usuario se definan con un buen nivel de granularidad. Además, según se vayan añadiendo más funcionalidades al proyecto se deberán volver a aplicar las pruebas de aceptación a todas las funcionalidades ya implementadas.
Valor entregado y retorno de inversión	EVM-Ágil (del inglés AgileEVM)	Adaptación de la técnica tradicional de gestión de proyectos conocida como *Earned Value Management* (EVM) que mide el rendimiento del proyecto en relación a su línea base, es decir, en relación a lo que se había planificado. Con el uso de esta técnica se trata de identificar las posibles desviaciones en tiempo y presupuesto. EVM-Ágil dará a conocer a los clientes y al equipo de desarrollo de qué forma se está obteniendo el valor esperado, cuál es el retorno de inversión y su velocidad haciendo uso en este caso de los puntos de historia como la medida a utilizar para realizar los cálculos. Por tanto, para poder hacer uso de esta técnica, será necesario realizar una planificación inicial de las iteraciones, una planificación de las historias de usuario que se van a incluir en cada iteración y el valor que se le da a cada historia de usuario.

Gestión de riesgos	*Impediment Backlog*	Herramienta que muestra la lista priorizada de incidencias abiertas, con el objetivo de solventar los problemas que están impidiendo un progreso adecuado del desarrollo de las funcionalidades en cada iteración (Ver ejemplo en Figura 8).

Figura 7. Ejemplo cuadro de mando ágil (Garzás, Enríquez de S. e Irrazábal, 2013).

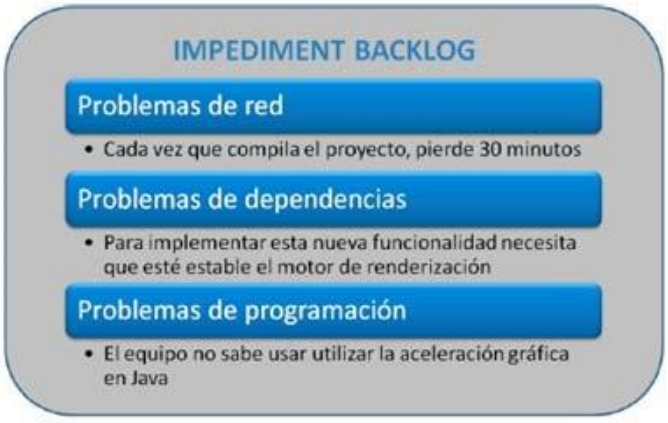

Figura 8. Ejemplo de salida mostrada por una herramienta *Impediment backlog* (Garzás, Enríquez de S. e Irrazábal, 2013)