8. Scrum

8.1. Introducción y objetivos

En este tema se estudia a fondo *Scrum,* una de las técnicas ágiles más populares, simples y extendidas. Como el resto de este tipo de técnicas, sigue los principios propuestos en el manifiesto ágil y los concreta en una serie de indicaciones relacionadas con la definición de roles específicos, reuniones y artefactos, que soportan el proceso de desarrollo de *software* y permiten guiar el proyecto utilizando una documentación muy ligera a la vez que efectiva.

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

* Conocer el enfoque general del proceso de desarrollo bajo *scrum* y su relación con los principios del manifiesto ágil.
* Conocer los principales roles que intervienen en el proceso, su implicación y el papel que juegan en cada fase del desarrollo.
* Conocer las principales reuniones propuestas por scrum y sus beneficios desde el punto de vista del avance del proyecto y el intercambio de información.
* Comprender y saber utilizar los principales artefactos, pensados para mantener la especificación de historias de usuario y controlar el avance del proyecto en su conjunto.
* Saber extender la técnica básica de Scrum, pensada inicialmente para equipos de tamaños reducidos, a proyectos de mayores dimensiones.

8.2. Scrum

Scrum, cuyo nombre tiene su origen en una formación propia del rugby, en la que dos grupos de jugadores, de equipos diferentes, se colocan alrededor del balón y en oposición violenta para ganar el control de este y desplazarlo alrededor del campo, es un marco de trabajo (*framework*) para la gestión de proyectos *software* (Schwaber y Beedle, 2001), por lo que cada organización deberá adoptar y adaptar scrum en función de sus propias características y necesidades. Scrum fue desarrollada por Jeff Sutherland y Ken Schwaber a comienzos de la década de 1990.

Aunque scrum puede ser de mucha ayuda, en realidad es un compendio de buenas prácticas, por lo que no constituye un método prescriptivo ni una metodología, como podrían ser *Rational Unified Process (*RUP) o Métrica 3 y, por tanto, no va a decir qué es lo que hay que hacer en cada caso, punto por punto (Kniberg, 2007).

No prescribe actividades concretas ni ofrece una especificación detallada de la manera en que deben organizarse las tareas de desarrollo, sino que proporciona un contexto en el que puedan ser los propios involucrados quienes creen su propio proceso.

Según sus creadores, Schwaber y Sutherland (2020):

«[…] Scrum es simple. Pruébalo tal cual y determine si su filosofía, teoría y estructura ayudan a alcanzar metas y crear valor. El marco de Scrum es deliberadamente incompleto, solo define las partes necesarias para implementar la teoría de Scrum. Scrum se basa en la inteligencia colectiva de las personas que lo utilizan. En lugar de proporcionar a las personas instrucciones detalladas, las reglas de Scrum guían sus relaciones e interacciones» (p. 3).

Scrum tiene una fundamentación eminentemente empírica, defendiendo que cualquier decisión debe tomarse a partir del conocimiento acumulado por la experiencia. Para poder establecer un control del proceso empírico, scrum se asienta sobre tres pilares (Schwaber y Sutherland, 2020):

* Transparencia: la información significativa sobre los objetivos, alcance y características del proyecto debe ser accesible y visible para todos los participantes.
* Inspección: los participantes deben revisar con frecuencia los artefactos del proceso y la información disponible para evitar posibles desviaciones.
* Adaptación: tan pronto como se detecte alguna desviación respecto a los objetivos previstos, se deben tomar medidas correctoras.

Con lo cual, hay que tener cierta cautela y mucho cuidado de no perder la agilidad en el camino (muchas empresas se consideran ágiles por adoptar scrum cuando en realidad no lo son, ya que scrum es una herramienta que ayuda a ser ágil pero no lo garantiza). Garzás (2015), describe unos errores muy comunes que se dan en organizaciones que desean ser ágiles y que adoptan scrum como herramienta para tal fin:

* No hay colaboración entre el cliente y el equipo de desarrollo. Lo normal es que el cliente sea poco flexible y comprensivo, continuamente presione con los tiempos de entrega y solicite más funcionalidad sin reajustar el presupuesto. Esto hace que el equipo de desarrollo tenga que trabajar más horas de las estipuladas y producir un *software* que, normalmente, no tendrá la calidad esperada. Esto podría ser un intento de adopción de *scrum,* pero en realidad no es nada ágil. Se está incumpliendo el valor de colaboración que debería existir entre el cliente y el equipo de desarrollo, tal y como es entendido en *scrum.*

La agilidad implica un cambio en la forma de contratación. El cliente y el contratista deben colaborar entre sí de una manera especial. Por ejemplo, el equipo de desarrollo se compromete a ir haciendo entregas incrementales (en forma de iteraciones) del producto *software* que está solicitando el cliente, cada cierto periodo limitado de tiempo, y el cliente acepta que el equipo de desarrollo no tenga del todo claro qué es lo que espera del sistema.

En este sentido, el contratista es flexible a la hora de gestionar los cambios y aceptar la incertidumbre en los requisitos que se han de cumplir. A cambio, el cliente dará prioridad a los requisitos del sistema que se han de implementar y, si hay que hacer cambios por problemas de malentendidos o falta de comunicación, se podrá llegar a un acuerdo y renegociar el contrato en lo que se refiere a condiciones, fecha de entrega, presupuesto, etc.

* Las personas no son valoradas en su justa medida. Situación muy común. Se tienen los roles que define *scrum,* pero los equipos están desmotivados y no se les deja actuar con creatividad, ya que en la organización se sigue una política de mando y control donde la producción se mide por el tiempo que se pasa en la oficina.
* El equipo no es maduro. Para ciertas prácticas ágiles se necesita que el equipo de desarrollo sea profesional y maduro. *Mob programming* es un buen ejemplo de ello. Esta técnica consiste en que todo el equipo de desarrollo trabaje en la misma cosa, al mismo tiempo, en el mismo espacio y en el mismo ordenador. *Mob programming* es una técnica muy efectiva, pero hay que ejecutarla con responsabilidad y suficiente madurez, ya que, en otro caso, se corre el riesgo de que varias personas del equipo de desarrollo estén, por ejemplo, viendo vídeos de entretenimiento en YouTube.
* Falta de liderazgo. En el enfoque ágil es cierto que se da menos importancia a la jerarquía y se ven estructuras más horizontales, pero esto no significa que no haya personas que lideren ciertos aspectos. Cuando se desarrolla de manera ágil hace falta un facilitador que fomente buenas prácticas, sea capaz resolver conflictos y promueva actitudes que ayudaen a que el equipo de desarrollo sea multifuncional.
* Falta de empatía entre los *stakeholders.* En un enfoque ágil tanto los que desarrollan el producto *software,* como los que promocionan desde la parte de desarrollo de negocio, los responsables de calidad, etc., han de entenderse y estar alineados con el negocio para trabajar juntos en una misma dirección cuyo objetivo es la satisfacción del cliente. En otras ingenierías, cuando lo que se fabrican son productos físicos, todas estas divisiones organizativas como, por ejemplo, trabajadores de la cadena de montaje, personal de marketing, personal de calidad, etc., están claramente separadas y no necesitan esa alineación que es necesaria cuando se desarrolla *software.*

Valores de *scrum*

Los principios que sigue *scrum* cumplen con los establecidos en el manifiesto ágil y se utilizan como guía dentro del proceso de desarrollo de *software.*

El objetivo de *scrum* es el de obtener resultados de una manera rápida, adaptándose con facilidad a los cambios en los requisitos que ha de cumplir el sistema (Pressman, 2010, p. 69).

A continuación, se describen los valores del manifiesto ágil según *scrum:*

* Valoración de los individuos y sus interacciones sobre procesos y herramientas: en *scrum* los equipos toman el control y la responsabilidad del desarrollo, resuelven los problemas que encuentran, con la ayuda del *scrum master,* y se comunican a través de una serie de reuniones de carácter generalmente informal.
* Valoración del *software* funcional sobre la documentación exhaustiva: al final de cada iteración o sprint se entrega al cliente una nueva versión del producto lista para su utilización. La documentación es una herramienta intermedia que sirve de apoyo al proceso de desarrollo y no constituye en sí misma un producto entregable.
* Valoración de la colaboración con el cliente sobre la negociación contractual:*scrum* adopta la figura del *product owner* como representante del cliente y otros *stakeholders.* Forma parte del equipo *scrum*(no del equipo de desarrollo) y es responsable, en colaboración con el equipo de desarrollo, de que las iteraciones se orienten hacia la maximización del valor aportado a los clientes o usuarios finales.
* Valoración de la adaptación al cambio frente al seguimiento de un plan: el enfoque de las reuniones y artefactos empleados facilita que todos los participantes tengan información actualizada sobre las características y alcance del proyecto. De esta manera, todos están capacitados para tomar decisiones e introducir modificaciones, y la información para detectar cualquier variación en el entorno se obtiene gracias a la participación del *product owner* y la continua liberación de código funcional.

Estos valores generales son acompañados y reforzados por una serie de valores sobre los que se apoya *scrum* (Alaimo, 2013; Schwaber y Sutherland, 2020):

* Compromiso: de las responsabilidades y la libertad otorgadas a los miembros del equipo en la toma de decisiones sobre el proyecto. Surge un compromiso para sacar adelante el trabajo y cumplir los objetivos de cada iteración.
* Coraje: los miembros del equipo se apoyan entre sí para afrontar retos complejos y vencer obstáculos a lo largo de cada sprint, con el objetivo común de alcanzar las metas planteadas en cada iteración.
* Focalización: en cada iteración el equipo se centra en un conjunto concreto y acotado de características en cada sprint.
* Apertura: durante el desarrollo del proyecto se fomenta la transparencia y el flujo de información, de manera que todos los miembros del equipo tengan acceso permanente a las características del producto y del proceso.
* Respeto: los miembros del equipo *scrum* se respetan mutuamente para ser personas capaces e independientes.

Flujo del proceso *scrum*

La Figura 1 ilustra el flujo general del proceso que define *scrum.* Aunque se explicarán más en detalle en los siguientes apartados, merece la pena introducir los siguientes elementos que permiten obtener una visión general de *scrum* (Pressman, 2010, p. 69-70):

* Backlog. Aunque en la traducción de Pressman (2010) se ha interpretado como ‘retraso del sprint’ y ‘retraso del producto’, estas traducciones no son muy acertadas, de ahí que se prefiera mantener los términos en su versión original en inglés. En este caso, *sprint backlog* y *product backlog,*respectivamente, pues así no hay confusión. En castellano la traducción más aceptada para retraso es la de ‘pila’ (‘pila de sprint’ y ‘pila de producto’), aunque, como para el resto de los elementos vinculados a scrum, utilizamos los términos originales en inglés.

El *product backlog* constituye la lista de prioridades en lo que a funcionalidades del sistema se refiere.

Se podrán añadir funcionalidades en cualquier momento. Si se añaden nuevas funcionalidades, un responsable será quien se encargue de evaluar el *product* *backlog* y lo actualizará con las nuevas prioridades según estime oportuno.

* Sprint. Cada sprint representa las unidades de trabajo que se necesitan para implementar una funcionalidad definida en el *sprint backlog* y que se debe ajustar a un calendario, normalmente de un mes como máximo. Durante la realización de un sprint nunca se van a introducir cambios, de esta manera se conseguirá que el equipo de desarrollo trabaje con entregas a corto plazo y de naturaleza estable, sin alteraciones.
* *Daily scrum*. Son reuniones diarias, breves, normalmente de unos quince minutos, que lleva a cabo el equipo de desarrollo. En este tipo de reuniones hay tres preguntas clave que se han de responder por parte de cada uno de los miembros del equipo de desarrollo:

¿Qué hiciste desde la última reunión del equipo de desarrollo?

¿Con qué problemas te has encontrado a la hora de realizar tu trabajo?

¿En qué vas a trabajar hasta la próxima reunión?

Un líder del equipo de desarrollo, denominado *scrum master,* dirigirá la reunión y evaluará cada una de las respuestas. En este tipo de reuniones, se intenta descubrir los problemas potenciales que puede tener asociados el desarrollo del sistema y se intenta «socializar el conocimiento», de tal forma que se promueve una estructura de equipo de desarrollo con organización propia y más autónoma.

Con las nuevas actualizaciones de la guía de *scrum* se ha intentado que cada vez sea menos prescriptivo, para mejorar la esencia para lo que fue creado. Por ello, se han ido eliminando las preguntas del *daily scrum,* entre otras cuestiones (Schwaber y Sutherland, 2020).

* Demostraciones preliminares. El incremento de *software* que se haya realizado en una iteración se entregará al cliente para que proceda a su evaluación y validación. Es importante destacar que las demostraciones preliminares no contendrán toda la funcionalidad acordada para el sistema, ya que esa funcionalidad completa se entregará en la fecha que se haya establecido para el proyecto.

A diagram of a scrum process

Description automatically generated

Figura 1. Flujo del proceso Scrum. Fuente: Sutherland y Schwaber, 2012.

En este esquema (Figura 1) se muestran los principales roles (Scrum Master, Product Owner y Equipo), reuniones (Planificación de sprint, *scrum* diario, revisión y retrospectiva del sprint) y artefactos (Product Backlog y Sprint Backlog).

Lo fundamental es comprender que se trata de un proceso iterativo con comunicación y retroalimentación constantes.

La definición del *sprint backlog* durante la reunión de «planificación» marca el comienzo de cada iteración, y el equipo de desarrollo se compromete con el *product owner* en la entrega de un conjunto de funcionalidades identificadas en este documento.

En resumen, se puede afirmar entonces que *scrum* es un marco de trabajo que permite obtener resultados de una manera rápida, es flexible a los cambios que se puedan producir en los requisitos, mejora la gestión de riesgos, favorece la comunicación entre los *stakeholders* del proyecto y presenta tres características fundamentales:

* El producto *software* se desarrolla con un enfoque iterativo + incremental.
* Las funcionalidades presentes en cada entrega parcial (incremento de la iteración) serán priorizadas.
* Se realizarán reuniones frecuentes a lo largo de todo el proyecto.

8.3. Roles

De acuerdo con *scrum*, en todo proyecto *software* se darán tres roles (Garzás, Enríquez de S. y Irrazábal, 2013; Schwaber y Sutherland, 2020):

* *Product owner.* Este rol es el responsable de gestionar todas las funcionalidades que se han de cumplir con la implementación del sistema, asegurando también el valor del trabajo que el equipo de desarrollo lleva a cabo, es decir, es el responsable último del éxito del producto, que debe ajustarse a las necesidades del cliente y de los usuarios finales. Tiene la visión general del producto, define las necesidades y comprende los puntos de vista del cliente y otros *stakeholders.* Su principal misión es maximizar la rentabilidad del producto y, para ello, debe considerar en primer lugar la priorización de las funcionalidades deseadas.

De manera resumida, se puede decir que la contribución del *product owner* al proyecto está basada en:

* + Determinar las funcionalidades o historias de usuario.
  + Gestionar y priorizar las funcionalidades.
  + Aprobar el producto *software* que resulta al final de cada iteración.
  + Maximizar el retorno de inversión del proyecto.
* *Scrum master.* Este rol adopta una función de coaching dentro del equipo y es el responsable de asegurarse de que todo el equipo *scrum* (no solamente el equipo de desarrollo) siga las prácticas *scrum*.

No se debe confundir su figura con la de un jefe de proyecto, pues no está encargado de planificar ni de asignar tareas. Lo ideal es que exista un *scrum master* con dedicación completa para el equipo, aunque en proyectos pequeños puede desarrollar esta labor algún miembro del propio equipo de desarrollo (Sutherland y Schwaber, 2012).

Existen diferentes guías y listas de comprobación que pueden ayudar al *scrum master* a desarrollar sus tareas de manera eficiente (CollabNet, 2012; James, 2012; Kniberg, 2013). Estos documentos pueden ayudar a este rol a realizar un correcto seguimiento de las actividades del equipo y a detectar posibles obstáculos.

De manera resumida, se puede decir que sus funciones principales son las que se listan a continuación:

* + Ayudar a que el equipo de desarrollo y la organización en general adopten *scrum.*
  + Liderar el equipo *scrum* con el objetivo de mejorar en la productividad y calidad de las entregas parciales.
  + Ayudar en la autogestión del equipo de desarrollo.
  + Gestionar e intentar resolver los conflictos o problemas con los que el equipo de desarrollo se encuentra en su trabajo.
* Equipo de desarrollo. Este rol es el que adoptan todos los desarrolladores del sistema y serán los que convertirán las necesidades del *product owner* en un conjunto de nuevas funcionalidades o modificaciones del sistema final. El equipo de desarrollo presentará las siguientes características:
  + Autogestionado. Que el equipo de desarrollo sea autogestionado quiere decir que es el mismo equipo quien supervisa su trabajo. En *scrum,* como ya se ha mencionado anteriormente, se promueven las *daily* *scrum* del equipo de desarrollo, por lo que se aumentará y se favorecerá la comunicación entre los distintos miembros del mismo. Solo ellos, sin intervención de ningún otro rol, determinan la manera de planificar el proceso de desarrollo en cada iteración y la descomposición de funcionalidades en tareas o su asignación a desarrolladores concretos.
  + Multifuncional. En *scrum* no existen especialistas, por lo que cada miembro del equipo de desarrollo ha de ser capaz de realizar tareas de programación, pruebas, despliegue, etc. Eso no impide que cada miembro tenga unas capacidades y conocimientos diferentes y que puedan estar más capacitados para unas tareas que para otras, pero lo importante es que puedan ser capaces de realizar cualquier función que sea necesaria en lo relativo al proceso de desarrollo de *software.* Engloba perfiles heterogéneos que cubren todas las necesidades del proyecto y de su implementación.
  + No distribuidos. En *scrum* es importante que el equipo de desarrollo se encuentre ubicado en el mismo lugar físicamente. Así se va a facilitar la comunicación y la autogestión que surge del mismo equipo. Sin embargo, no es un requisito imprescindible, ya que, gracias a las nuevas tecnologías, con herramientas colaborativas se pueden gestionar los equipos distribuidos sin gran dificultad.
  + Tamaño óptimo. Sin tener en cuenta los roles de *scrum master* y *product owner,* lo ideal es que un equipo de desarrollo esté compuesto por al menos tres personas y no más de ocho o nueve personas. Con menos de tres personas la interacción va a ir a menos y, como consecuencia, también la productividad del equipo de desarrollo. Con más de nueve personas la interacción va a hacer que la autogestión se complique demasiado.

A screenshot of a computer screen

Description automatically generated

Figura 2. Tamaño óptimo. Fuente: elaboración propia.

Además, el equipo de desarrollo tiene tres responsabilidades fundamentales (Alaimo, 2013):

A blue and white rectangular box with black text

Description automatically generated

Figura 3. Responsabilidades fundametales del equipo de desarrollo. Fuente: elaboración propia.

Por otro lado, los roles es *scrum* se pueden clasificar en dos categorías principales (ver Figura 2):

* Pigs. En *scrum,* este rol lo adoptarán el *product owner,* el *scrum master* y todos los miembros del equipo de desarrollo, ya que son los roles que realmente estarán comprometidos con el desarrollo del *software.*
* Chickens. En *scrum*, este rol lo adoptarán todas aquellas personas que no están involucradas directamente con el proyecto, pero que son necesarias para su buen funcionamiento, tales como: el cliente y los *stakeholders* del departamento de contabilidad o financiero.

A diagram of scrum and scrum

Description automatically generated

Figura 4. Roles Pigs y Chickens. Fuente: Pacheco Masdíaz, 2017.

8.4. Product backlog

El *product backlog* o ‘pila de producto’, representa el elemento principal que forma parte del modelo de proceso de desarrollo ágil con *scrum.*

El *product backlog* es responsabilidad del *product owner* y constituye la lista priorizada de las historias de usuario o funcionalidades que ha de cumplir el sistema y se encuentra escrito en un lenguaje que el cliente puede entender.

Con lo cual, se podría decir que el *product backlog* es como un listado de requisitos de usuario al mayor nivel de abstracción posible. El *product backlog* es lo que va a permitir que los *stakeholders* tengan una visión única del sistema a lo largo de todo el proyecto, y su continua evolución a través de las entregas parciales será lo que vaya dando mayor valor al producto *software* que se está desarrollando.

Es un documento vivo, que evoluciona a medida que el desarrollo avanza y el entorno cambia. Puede haber modificaciones en historias de usuario existentes debido a cambios en el negocio, el mercado o la propia tecnología, y también es posible la eliminación o aparición de nuevas historias de usuario. El *product owner* puede introducir modificaciones de manera unilateral cuando se considere necesario, atendiendo siempre a criterios de negocio.

De acuerdo a Pichler (2010), todo *product backlog* presenta cuatro características que lo definen:

* Está suficientemente detallado. La granularidad o nivel de detalle de un *product backlog* dependerá de la prioridad que tenga. Las historias de usuario que tengan más prioridad se describirán en mayor detalle. Es por ello por lo que se van a ir encontrando nuevas funcionalidades a medida que se vayan detallando las historias de usuario que se van a implementar, lo que deriva en una evolución continua del *product backlog.*
* Es algo estimado. El *product backlo*g se estimará en duración y en otros términos abstractos, basándose en el tamaño de las historias de usuario que contiene, lo que ayudará a darle prioridad y a realizar las planificaciones.
* Es emergente. Las funcionalidades que ha de cumplir el sistema van a ir cambiando y se van a ver alteradas o eliminadas con relativa frecuencia. Las nuevas historias de usuario o funcionalidades que formarán parte del *product backlog* se irán descubriendo y añadiendo a la lista, teniendo en cuenta los comentarios de los clientes y usuarios finales del sistema.
* Está priorizado. Las historias de usuario, o funcionalidades, se priorizarán de tal formar que las más prioritarias aparecerán primero en el *product backlog*. En un primer momento, puede que no sea necesario priorizar todas las historias de usuario, pero sí resulta conveniente identificar al menos las quince o veinte más importantes.

Adicionalmente al *product backlog,* se pueden crear otros elementos que serán de utilidad para la implementación de las historias de usuario, ya que aportarán una mayor explicación de los contenidosque aparecen en el *product backlog,*como, por ejemplo, prototipos de la interfaz de usuario, diagramas de las reglas de negocio y un resumen de los distintos usuarios que pueden operar con el sistema (Garzás, Enríquez de S. y Irrazábal, 2013, p. 48).

Para Kniberg (2007, p. 17-18), las historias de usuario que forman parte de un *product backlog* podrían contener los siguientes campos (ver ejemplo en Tabla 1):

* Identificador. Este campo representa un identificador único que permite localizar las historias de usuario aunque cambien de nombre.
* Nombre. Este campo consiste en una descripción corta de la historia de usuario, como, por ejemplo, «Ver historial de transacciones». Con esta descripción el *product owner* ha de ser capaz de entender qué es lo que hace la historia de usuario. Lo normal es que contenga de dos a diez palabras.
* Importancia. Este campo indicará el grado de importancia que el *product owner* asigna a una historia de usuario.
* Estimación inicial. Este campo representa la valoración inicial que da el equipo de desarrollo en lo que respecta al trabajo a realizar para implementar una historia de usuario. La unidad de medida, en este caso, son los puntos de historia.
* Pruebas. Este campo representa una descripción de alto nivel de cómo se puede validar una historia de usuario. Una prueba sería, por ejemplo, una especificación del tipo «Haz X, una vez hecho X entonces haz Y, y una vez hecho Y debería cumplirse Z». Si para hacer las pruebas se utiliza un enfoque *Test-Driven Development*(TDD) entonces esta descripción de alto nivel podría utilizarse como pseudocódigo en las pruebas de aceptación.
* Notas. Este campo representa cualquier información adicional, comentario, aclaración o referencia a otras fuentes de información, etc., que pueda constituir un elemento de utilidad para una historia de usuario concreta.

A table with text and numbers

Description automatically generated

Tabla 1. Ejemplos de historias de usuario. Fuente: Kniberg, 2007.

Junto a esto, hay que añadir que existen diversas herramientas que se podrían utilizar para crear un *product backlog*. Kniberg (2007), por ejemplo, sugiere utilizar Jira (Atlassian, 2022) para almacenar el *product backlog* de un proyecto. Microsoft Excel también es una herramienta excelente que ofrece muchas posibilidades para la gestión y mantenimiento del *product backlog* (en realidad, hay mucho desconocimiento en cuanto a las cantidades de funcionalidades que proporciona esta herramienta para muchos temas vinculados con la gestión de proyectos *software)*.

Tal y como destaca Garzás (2013, p. 28-29), la hoja de ruta se complementará muy bien con el *product backlog*, ya que la hoja de ruta ofrecerá la planificación estratégica del producto, capturando las funcionalidades principales de cada entrega, lo que permitirá mirar más hacia el futuro, es decir, ver hacia qué mercado se quiere dirigir el producto. Mientras que, por su parte, el *product backlog* proporcionará la planificación del trabajo más táctico, centrándose en la creación de la siguiente entrega e indicando lo que debe estar terminado para que se pueda aprobar.

Lo ideal es que una misma persona dentro del equipo *scrum* sea quien se encargue de unir los objetivos estratégicos y tácticos del producto *software* a desarrollar, estableciendo así una clara autoridad y responsabilidad sobre los mismos (Garzás, 2013, p. 29).

Un mapa de historias de usuario sería también un mecanismo adicional para priorizar y complementar un *product backlog* tal y como se muestra en la Figura 3. En el nivel superior de los mapas de historias de usuario se suelen situar las epopeyas y, debajo de cada una de ellas, se pondrán las historias de usuario en las que se descomponen.

Además, las epopeyas se pondrán de izquierda a derecha en función del orden en el que se usen en la aplicación, o bien, se puede optar por ponerlas en el orden que mejor describa el sistema a implementar. Por su parte, las historias de usuario que forman parte de una epopeya concreta se ordenarán de arriba abajo en función de su prioridad. Por último, si se desea mostrar las diferentes entregas *(releases)* que se van a realizar, se podrán agrupar las historias de usuario a través de líneas horizontales (Garzás, 2013, p. 28).

A diagram of a history

Description automatically generated

Figura 5. Mapa de historias de usuario. Fuente: Garzás, 2013.

8.5. Sprint

Siguiendo el modelo de proceso iterativo e incremental, en un proyecto ágil, el *software* se va a ir desarrollando en partes (iteraciones) sucesivas, donde cada parte va mejorando la calidad y añadiendo funcionalidad a la anterior. Así hasta obtener el sistema completo, con el visto bueno y aprobación del cliente. En *scrum,* cada una de esas iteraciones se denomina sprint y constituye una versión operativa del software que debe implementar.

Según Schwaber y Sutherland (2020), es cada uno de los períodos de tiempo de duración fija, nunca superior al mes, durante el cual se crea un incremento del producto «terminado, utilizable y potencialmente desplegable» (p. 8).

Para *scrum*, las duraciones de los *sprints* han de ser cortas, en periodos de tiempo que abarcan entre una y cuatro semanas (máximo de treinta días), para que resulten útiles y realmente se considere un desarrollo ágil. La duración de los *sprints* resulta un aspecto fundamental y crítico en todo proyecto de desarrollo ágil, por lo que ha de gestionarse de la manera que sea más realista de acuerdo a las características de la organización que adopta *scrum* y que pueda garantizar el éxito del proyecto. Aunque no existe una normativa regulada, Garzás (2013, p. 46) lista una serie de factores que serían interesantes tener en consideración a la hora de determinar la duración más recomendable para un sprint:

* En base a la frecuencia con la que el cliente espera ver resultados habrá que organizar las funcionalidades para que se ajusten al tiempo requerido.
* En base a las capacidades del equipo de desarrollo, ser capaz de estimar la cantidad de trabajo que es capaz de completar cada miembro del equipo de desarrollo, en lo que respecta a las funcionalidades almacenadas en el *product backlog* para el sprint a planificar.
* Si se estima que los cambios por parte del cliente van a ser muy frecuentes, es decir, se trata de un tipo de cliente que no tiene muy claro qué es lo que necesita y va cambiando los requisitos con relativa frecuencia, los *sprints* no deberían ser muy largos, ya que no se permiten cambios una vez iniciado el sprint y si se altera lo que ya en sí se está implementando, sería mucho más complicado de gestionar.

De lo anterior, podría surgir la duda entonces de si todos los *sprints* que forman parte de un proyecto deberían durar lo mismo. Existen organizaciones que optan por decidir la duración del sprint justo antes de su comienzo, es decir, también se planifica a nivel de sprint. Sin embargo, para equipos de desarrollo que no tengan mucha experiencia en entornos de desarrollo ágil, resulta mucho más recomendable establecer al comienzo del proyecto una duración para todos los *sprints* y planificar e implementar las historias de usuario en base a dicha duración.

El mantener la misma duración para todos los *sprints* también suele ser la opción preferida por muchas organizaciones, independientemente de si ya tienen experiencia o no con *scrum,* o con desarrollos ágiles en general, ya que defienden la idea de que así el equipo de desarrollo se acostumbra a un ritmo de trabajo lo que favorece su rendimiento y forma de trabajar (Garzás, 2013, p. 47).

Todas las funcionalidades que serán implementadas en un sprint y que serán recogidas en lo que se denomina en *scrum* como *sprint backlog*, han de formar parte del *product backlog* del proyecto. El equipo de desarrollo, a través una reunión de planificación del sprint, lo que se conoce como *sprint planning,*seleccionará las historias de usuario que se van a desarrollar en el sprint y que formarán parte del *sprint backlog,* así como el número de puntos que le asigna a cada historia de usuario, tal y como ya se explicó en el tema anterior con la técnica *planning poker*, por ejemplo.

Llegados este punto, Kniberg (2007) remarca e insiste mucho en el hecho de que el *product backlog* ha de estar ya listo y validado en este punto del proyecto, antes de realizar la reunión *sprint planning*. Cualquier cambio sobre el *product backlog* una vez definido o cuando se está organizando el *sprint backlog* podría traer consecuencias fatales en la gestión y desarrollo del proyecto, es por ello por lo que, como se ha comentado anteriormente, una vez iniciado el sprint ya no se admite ninguna modificación.

La Figura 4 ilustra la relación entre *product backlog*, *sprint backlog* y sprint tal y como lo contempla *scrum.*

A diagram of a schematic diagram

Description automatically generated

Figura 6. El Sprint dentro del proceso Scrum. Fuente: Garzás, Enríquez y Irrazábal, 2013.

Como se destaca en Garzás, Enríquez de S. y Irrazábal (2013, p. 48), cada equipo de desarrollo es libre de realizar la descomposición, análisis y desarrollo del sprint *backlog* como mejor estime en cada caso, ya que *scrum* no dictamina nada en este sentido. En cualquier caso, la implementación más usual, tal y como lo explican Garzás, Enríquez de S. y Irrazábal (2013, p. 49), es la de hacer un desglose de las historias de usuario seleccionadas para un sprint concreto en tareas que se han de realizar, para poder implementar la totalidad de dichas historias (ver ejemplo en Figura 5).

A diagram of a product backlog

Description automatically generated

Figura 7. Desglose de las historias de usuario en tareas. Fuente: Garzás, Enríquez y Irrazábal, 2013.

Las tareas identificadas se incluirán en una tabla que detalla la forma en la que el equipo de desarrollo va a llevar cabo la implementación de las historias de usuario, de las que forman parte dichas tareas. Las tareas se dividirán en horas de trabajo y ninguna de ellas ha de superar el límite de 16 horas para su realización, que será llevada a cabo por parte de un miembro del equipo de desarrollo.

Si se viera que una tarea requiere más de 16 horas para poder completarla, entonces lo que la experiencia aconseja es que la tarea, siempre que sea posible, se divida en tareas más pequeñas con una duración menor. La Tabla 2 muestra ejemplos de tareas que podrían formar parte de historias de usuario en un sprint.

A screenshot of a computer

Description automatically generated

Tabla 2. Ejemplos de tareas que podrían formar parte de un sprint. Fuente: Garzás, 2013.

A veces, puede resultar muy complicado desglosar las historias de usuario en tareas debido a falta de información, por lo que, en estos casos, se aconseja recurrir a modelos de análisis de alto nivel de abstracción o modelos de diseño propios de los desarrollos tradicionales (como serían, por ejemplo, los modelos UML) que ayuden a identificar qué es exactamente lo que se debe implementar en cada una de las funcionalidades que contempla el sprint.

Sin embargo, no hay que pensar, como ocurre en algunos casos cuando se adopta *scrum* para hacer un desarrollo ágil, que los *sprints* son como un mini ciclo en cascada. Garzás, Enríquez de S. y Irrazábal (2013, p. 51) indican las diferencias significativas que los diferencian:

* Todas las actividades de diseño, integración o pruebas se hacen de manera continua en el sprint y sin ningún tipo de secuencia, es decir, cualquier tarea se realiza en cualquier momento, sin importar qué le precede o le sigue. Sin embargo, en un ciclo de vida en cascada, estas etapas se realizarían de manera secuencial, aunque permita la retroalimentación.
* En el caso de un sprint, todos los miembros del equipo de desarrollo realizan todas las tareas que forman parte del proceso de desarrollo de *software,* y no están tan diferenciados como ocurre en los modelos de proceso de desarrollo tradicionales, como sería el modelo en cascada, donde existe gente especializada para cada función (analistas, diseñadores, programadores, ingenieros de pruebas...).

Como ya se ha comentado anteriormente, y se vuelve a insistir, una vez que el sprint con sus tareas ha quedado ya definido a través de las reuniones de planificación, es decir, el *sprint backlog* se da por cerrado, tampoco se admitirá ningún cambio en su implementación.

Aunque un sprint siempre va a generar un incremento en el producto *software* que se está implementando, conviene aclarar que no todos los *sprints* implican un paso a producción. Los clientes junto con el *product owner* decidirán cuándo se pone en producción la versión operativa del sistema que se tiene en ese momento. Tal y como se comenta en Garzás, Enríquez de S. y Irrazábal (2013, p. 50), el paso a producción se puede llevar a cabo con lo que se denomina en *scrum* como *«sprint de release»*.

Este tipo de sprint es especial, cuyas tareas estarán relacionadas exclusivamente con el despliegue, instalación y puesta en producción de la última versión operativa del producto *software.* En este caso, nos encontramos ante un tipo de iteración especial, ya que no supone un incremento con respecto al sistema que se está desarrollando.

Aunque puede ser una forma errónea de llamar a lo que se conoce como la planificación previa al primer sprint, pero que en sí no se puede considerar sprint, existe otro sprint especial, conocido como sprint cero(Garzás, 2013), que es el sprint que contiene las tareas a realizar antes de abordar el desarrollo del producto *software:* preparar los entornos de desarrollo, almacenar las historias de usuario ya priorizadas y estimadas en el *product backlog*, estimar la asignación de historias de usuario por sprint... En definitiva, el Sprint cero, de acuerdo con Ambler (2016), se encarga de:

* Estudio de las necesidades iniciales.
* Conceptualizaciones arquitectónicas iniciales.
* Preparación de los puestos de trabajo para cada uno de los miembros del equipo de desarrollo.
* Planificación inicial del proyecto.
* Información adicional necesaria para el arranque del proyecto.

De cara a determinar el trabajo que faltaría por llevar a cabo en el proyecto y comparar el progreso del sprint con la planificación inicial estimada, en *scrum* se recurre mucho a los gráficos *burndown*. Al ser específico para el marco de trabajo *scrum,* en el eje vertical se suele mostrar el *sprint backlog* y en el eje horizontal se suelen representar las jornadas laborales de los que se compone el sprint.

La Figura 6 ilustra un ejemplo de gráfico *burndown*. El gráfico *burndown* es un gráfico que se ha de actualizar diariamente, antes de la *daily scrum* del equipo de desarrollo y resulta muy útil para predecir desviaciones en el proyecto, ya que muestra de manera general el trabajo pendiente y el trabajo que ya ha sido realizado hasta ese momento.

A graph showing the growth of a number of people

Description automatically generated with medium confidence

Figura 6. Ejemplo de gráfico burndown. Fuente: Garzás, Enríquez y Irrazábal, 2013.

La realización de este tipo de gráficos va a proporcionar una serie de ventajas en lo que se refiere a la gestión del proyecto (Garzás, Enríquez de S. y Irrazábal, 2013, p. 101):

* Permite predecir el trabajo que falta para terminar el sprint.
* Permite extrapolar y utilizar la información que proporciona para la planificación de los siguientes *sprints.*

Como complemento a los gráficos *burndown* se pueden utilizar los gráficos *burnup* que muestran el avance del equipo hasta ese momento en lo que se refiere, normalmente, a puntos de historia, en relación con el esfuerzo total necesario para poder terminar el proyecto. Con un gráfico *burnup* se puede obtener una perspectiva de lo que quedaría para terminar el producto *software* y, a diferencia de los gráficos *burndown,* muestran el progreso actual del proyecto.

Este tipo de gráfico resulta muy útil también para el cliente y se debería actualizar siempre en las reuniones de retrospectiva del sprint (*sprint retrospective*), que se describirán más adelante, ya que es cuando se tiene la información completa de las historias de usuario entregadas durante el último sprint (Garzás, Enríquez de S. y Irrazábal, 2013, p. 106). La Figura 7 ilustra un ejemplo de gráfico *burnup* donde en el eje vertical las historias de usuario terminadas en el sprint y en el eje horizontal se muestran los días de duración del sprint.

A graph showing the growth of a number of sprints

Description automatically generated

Figura 7. Ejemplo de gráfico BurnUp. Fuente: Garzás, Enríquez y Irrazábal, 2013.

En lo que respecta a diagramas, un jefe/director de proyectos *software* que adopta *scrum* para la implementación de las nuevas aplicaciones podría preguntarse dónde encajarían los diagramas Gantt en todo el proceso de desarrollo. Los diagramas Gantt están muy ligados a esa idea inicial de la ingeniería de *software* en la que desarrollar *software* es como fabricar productos físicos, y la necesidad de utilizarlos se ha visto incrementada en los últimos años por el uso de determinadas normas de gestión de proyectos como la del *Project Management Institute* (PMI).

Sin embargo, en un desarrollo ágil la utilización de estos diagramas, que en algunos casos pueden llegar a ocupar varios folios que se unen con celo para formar como un póster, puede suponer un riesgo de fracaso en el proyecto, ya que, en demasiadas ocasiones, el equipo de desarrollo está más preocupado por cuadrar las barras, flechas, recursos, dimensión temporal, etc., que se muestran en el diagrama que por la implementación y calidad del producto *software* a desarrollar.

Con lo cual, cuando se adopte *scrum* para desarrollar sistemas *software,* se evitará en lo posible este tipo de diagramas y, en caso de ser necesarios (a veces pueden resultar incluso convenientes, según el tipo de organización), se procurará no caer en las malas prácticas que pueden poner en peligro la finalización o el éxito del proyecto (Garzás, 2013, p. 49).

En *scrum,* los diagramas Gantt se sustituirán por los planes de entrega *(release)*, los *sprints* y el seguimiento del proyecto con técnicas como los diagramas *BurnDown* descritos anteriormente.

Un esquema más detallado para ilustrar el modelo de desarrollo propuesto por *scrum* en base a los *sprints*es el que se muestra en la Figura 8.

A diagram of scrum planning

Description automatically generated

Figura 8. Flujo del proceso Scrum. Fuente: Pacheco Masdíaz, 2017.

8.6. Reuniones

Como ya se ha podido observar a lo largo de los apartados anteriores de este tema, otro elemento muy importante que forma parte del desarrollo ágil con *scrum* viene determinado por los distintos tipos de reuniones o ceremonias del proyecto. En *scrum*, las reuniones se van a clasificar en las siguientes categorías (Garzás, Enríquez de S. y Irrazábal, 2013, p. 52-53):

* *Sprint planning.* Esta reunión se va a realizar al comienzo de cada sprint y en ella se define lo que se va a implementar en la iteración actual. En esta reunión deberían participar todos los roles de *scrum* y de ella se obtiene el *sprint backlog*. En la reunión *sprint planning*, el *product owner*presentará el conjunto de historias de usuario que se encuentran almacenadas en el *product backlog*y el equipo de desarrollo hará una previsión del trabajo a realizar seleccionando las historias de usuario que se van a implementar en el sprint. En ella se establecen los compromisos, entre el equipo de desarrollo y el *product owner,*sobre el alcance y las características de la nueva entrega.

En esta reunión se debe responder a dos preguntas (Schwaber y Sutherland, 2020):

* + ¿Por qué el sprint es valioso? El *product owner*propone cómo el producto podría aumentar su valor y utilidad en el sprint actual. Después, todo el equipo *scrum* colabora para definir el *sprint goal* que comunique por qué el sprint es valioso para las partes interesadas.
  + ¿Qué puede hacerse en este sprint? Se parte del *product backlog*, y el *pr*oduct *owner*propone los elementos que deberían formar parte del objetivo. El número y la configuración final de las historias de usuario seleccionadas depende de la capacidad disponible y la experiencia acumulada en iteraciones previas, siendo algo que debe determinar finalmente el equipo de desarrollo.
  + ¿Cómo se conseguirá alcanzar el objetivo del sprint? El equipo de desarrollo analiza las funcionalidades seleccionadas y decide cómo construirá el nuevo incremento terminado. Las historias de usuario se descomponen en tareas individuales, cada una de las cuales no debería tener una duración superior a un día, y los miembros del equipo se autoasignan estas tareas comprometiéndose con su desarrollo. El resultado de este trabajo, que puede incluir tareas de diseño previo, es *sprint backlog*.
* *Daily scrum.* Aunque ya ha sido introducida anteriormente, en la visión general de *scrum,*caberecordar que esta reunión se realiza diariamente a lo largo de todo el proyecto y su duración suele ser de unos quince minutos. En esta reunión, cada miembro del equipo de desarrollo explica el trabajo que hizo el día anterior, los problemas encontrados para realizar su trabajo y lo que va a hacer ese día. Esta reunión es gestionada y dirigida por el propio equipo de desarrollo, y el *scrum master*se encarga de que se respeten los plazos y de que los flujos de comunicación sean los adecuados. Puede participar el *product owner*para resolver dudas.
* Sprint *review*. Esta reunión se realiza al final de cada sprint y tiene por objetivo demostrar el incremento y obtener retroalimentación de los interesados en el proyecto. En esta reunión participan el *scrum master*, el *product owner*, el equipo de desarrollo y algunos de los *stakeholders* del proyecto que pueden tener interés en lo que se va a presentar. En la reunión *sprint review*se presenta el trabajo realizado al *product owner*y se le explica lo que se ha podido finalizar y lo que no. Además, en esta reunión, el *product owner*, así como los *stakeholders* que asisten a la reunión, verifican el incremento del producto *software* y recogen la información necesaria en lo que a funcionalidad se refiere para poder actualizar el *product backlog*con nuevas historias de usuario.

El resultado de esta reunión es la aceptación (o el rechazo) de la nueva entrega, la decisión de poner la nueva versión en producción y un *product backlog* actualizado a partir del análisis realizado.

* Sprint *re*trospective. Esta reunión también se realiza al final de un *sprint*. Es una reunión de carácter eminentemente técnico, y en ella participa el equipo de desarrollo para analizar su propio trabajo y plantear posibilidades de mejora sobre los procesos. El *scrum master*se encarga una vez más de controlar los tiempos y asegurarse de que el diálogo se desarrolla bajo un espíritu constructivo. Tras analizar el último sprint, evaluando las personas, procesos y herramientas, se crea un plan de mejoras que se incluirá en el siguiente sprint.

El fin de este tipo de reuniones es conseguir una mejora en el proceso de desarrollo y normalmente se tratan exclusivamente dos aspectos: aspectos positivos del sprint y aspectos negativos del sprint.Básicamente en este tipo de reuniones lo que se discute es lo que ha funcionado en el sprint, lo que no ha funcionado en el sprint, lo que se ha hecho bien durante el sprint y lo que se podría hacer de cara a un futuro para evitar los problemas que se han dado en el sprint o mejorar ciertos aspectos.

La Figura 9 muestra la secuencialidad de las reuniones que se consideran en Scrum a lo largo de un proyecto.

A diagram of a process

Description automatically generated

Figura 9. Reuniones establecidas en *scrum.* Fuente: Scrum Institute, 2022.

8.7. Escalado de scrum

*Scrum* es una técnica pensada para equipos de desarrollo pequeños, donde la transparencia de la información y la comunicación habitual y fluida son posibles.

Existen proyectos de grandes dimensiones donde es necesaria la participación de equipos más voluminosos. En estos casos, el enfoque general es la distribución del personal en varios equipos de *scrum,*definiendo cauces de comunicación adicionales que favorecen la aplicación del método general. Kniberg (2015) ha participado en proyectos con equipos de hasta cuarenta personas y ofrece una serie de indicaciones para gestionar este problema.

La distribución de los equipos

Trabajar con equipos de gran tamaño introduce dificultades, como la excesiva duración de las reuniones diarias o la falta de visión general de los miembros del equipo por falta de comunicación. Cuando trabajamos con equipos experimentados y es factible dividir el desarrollo en varias líneas diferenciadas que no comparten código, es posible la división del equipo. En otro caso suele ser preferible mantener un único equipo.

Dada una conformación de equipos, el *scrum master* debe observar cómo se establecen los cauces de comunicación entre sus miembros. En la Figura 10 observamos dos posibles situaciones:

A blue and black symbols of a scrum

Description automatically generated with medium confidence

Figura 10. Formación de emergente de equipos virtuales. Fuente: adaptada de Kniberg, 2015.

* A la izquierda podemos ver un proyecto con un único equipo grande, pero los participantes tienden a agruparse en sus conversaciones en dos o más equipos virtuales de tamaño reducido.
* A la derecha, por el contrario, se ha definido un total de tres equipos de trabajo y, sin embargo, dos de ellos tienden a establecer, entre sí, interacciones más frecuentes.

Si estos comportamientos emergentes de agrupación en equipos virtuales se prolongan en el tiempo puede ser un síntoma de que la agrupación no es la correcta. Se debe comentar en la reunión de retrospectiva del sprint y dejar que los equipos decidan por sí mismos. En todo caso, es recomendable no superar las diez personas por equipo.

Otro factor es el grado de especialización de cada uno de los equipos. Podríamos pensar que es preferible agrupar individuos con conocimientos tecnológicos similares, especializados en el desarrollo de componentes o subsistemas concretos.

Esta situación aparece representada en la mitad izquierda de la Figura 11, pero, como vemos, puede suceder que haya historias de usuario que afecten a varios componentes, lo que obliga a una colaboración más estrecha entre diferentes equipos que puede ser difícil de gestionar e introduce dependencias innecesarias.

A diagram of a customer service

Description automatically generated

Figura 11. Especialización o multifuncionalidad de los equipos de desarrollo. Fuente: adaptada de Kniberg, 2015.

A la derecha de la Figura 11 observamos un enfoque con equipos multidisciplinares, donde cada uno por sí mismo es capaz de implementar en su totalidad cualquiera de las historias de usuario planteadas. La elección dependerá en cada caso del tamaño de los diferentes componentes, sus interdependencias, y la variedad de perfiles y especialidades con que contemos en el equipo.

La sincronización de los *sprints*

Cuando tengamos varios equipos trabajando en paralelo en el mismo proyecto, es necesario determinar una manera de sincronizar sus flujos de trabajo.

La recomendación es que trabajemos con *sprints* sincronizados, con todos los equipos comenzando y finalizando sus iteraciones al mismo tiempo.

Las ventajas de este enfoque son las siguientes:

* Disponemos de un punto natural en el que realizar redistribuciones de los equipos, sin que ello entorpezca el normal funcionamiento de uno de ellos.
* Se fomenta la colaboración entre equipos, que pueden comunicarse entre sí a la hora de planificar los objetivos de cada sprint.
* Se reduce el número de reuniones y la carga administrativa. Las reuniones de revisión se pueden tener de manera conjunta, realizando una única demostración.

Scrum de *scrums*

Cuando trabajamos en proyectos grandes, con varios equipos scrum trabajando simultáneamente, es necesario establecer cauces de comunicación y sincronización entre los diferentes equipos.

El *scrum* de *scrums* es una reunión periódica, frecuentemente diaria y con posterioridad al *scrum* diario de cada equipo, en la que uno o dos representantes de cada uno de ellos acude para compartir información.

Es preferible que se trate de representantes con perfil técnico, con capacidad para decidir sobre el proceso de desarrollo y asumir compromisos, y con una visión general de la iteración actual. En algunos casos, el enviado es precisamente el *scrum master* de cada equipo, aunque no sea lo idóneo.

A diagram of scrum of people

Description automatically generated

Figura 12. Representación jerárquica del Scrum de Scrums. Fuente: Ferrer, 2016.

En estas reuniones, los representantes de los equipos deben responder a las siguientes preguntas (tres de ellas similares a las planteadas en el *daily scrum*):

* ¿Qué tareas ha realizado tu equipo desde la última reunión?
* ¿Qué tareas va a realizar tu equipo hasta la siguiente reunión para lograr el objetivo del sprint?
* ¿Existe algún impedimento que afecte de manera negativa a tu equipo?
* ¿Existe algún factor con origen en tu equipo que puede afectar a los objetivos del resto de equipos?

La gestión del *product backlog*

Cuando tenemos varios equipos de *scrum,* cada uno de ellos con su *scrum master,* existen tres estrategias a la hora de gestionar las historias de usuario del sistema en su totalidad:

* Un *product owner* y un *product backlog*. Este es el enfoque preferido. Toda la información sobre las historias de usuario del producto está centralizada, y el *product owner* tiene una visión global del proceso de desarrollo. Las reuniones de planificación de cada sprint se suelen realizar en el mismo lugar y momento, pero en espacios diferenciados para los diferentes equipos. Es un trabajo intenso, pero al mismo tiempo se fomenta la comunicación entre todos los participantes.
* *Un product owner* y varios *product backlog*. El mayor problema de este enfoque consiste en la asignación de historias a los diferentes equipos. Puesto que el *product backlog* es gestionado por el *product owner*, se espera que sea este rol quien haga la asignación, pero esta es una tarea que requiere, en general, de un perfil técnico.
* Varios *product owner*, cada uno con su *product backlog*. Este enfoque es el más problemático, pues se diluye la visión general del producto entre varios *product owner* que deben coordinarse entre sí. Además, en el caso de que los diferentes *product backlog* tengan bases de código comunes, se puede dificultar el proceso de desarrollo.

En un caso ideal en el que los diferentes equipos estuvieran asignados a componentes con distintas bases de código, estaríamos próximos a un escenario en el que el producto se descompone en varios subproyectos relativamente independientes y, solo en este caso, este enfoque sería adecuado.