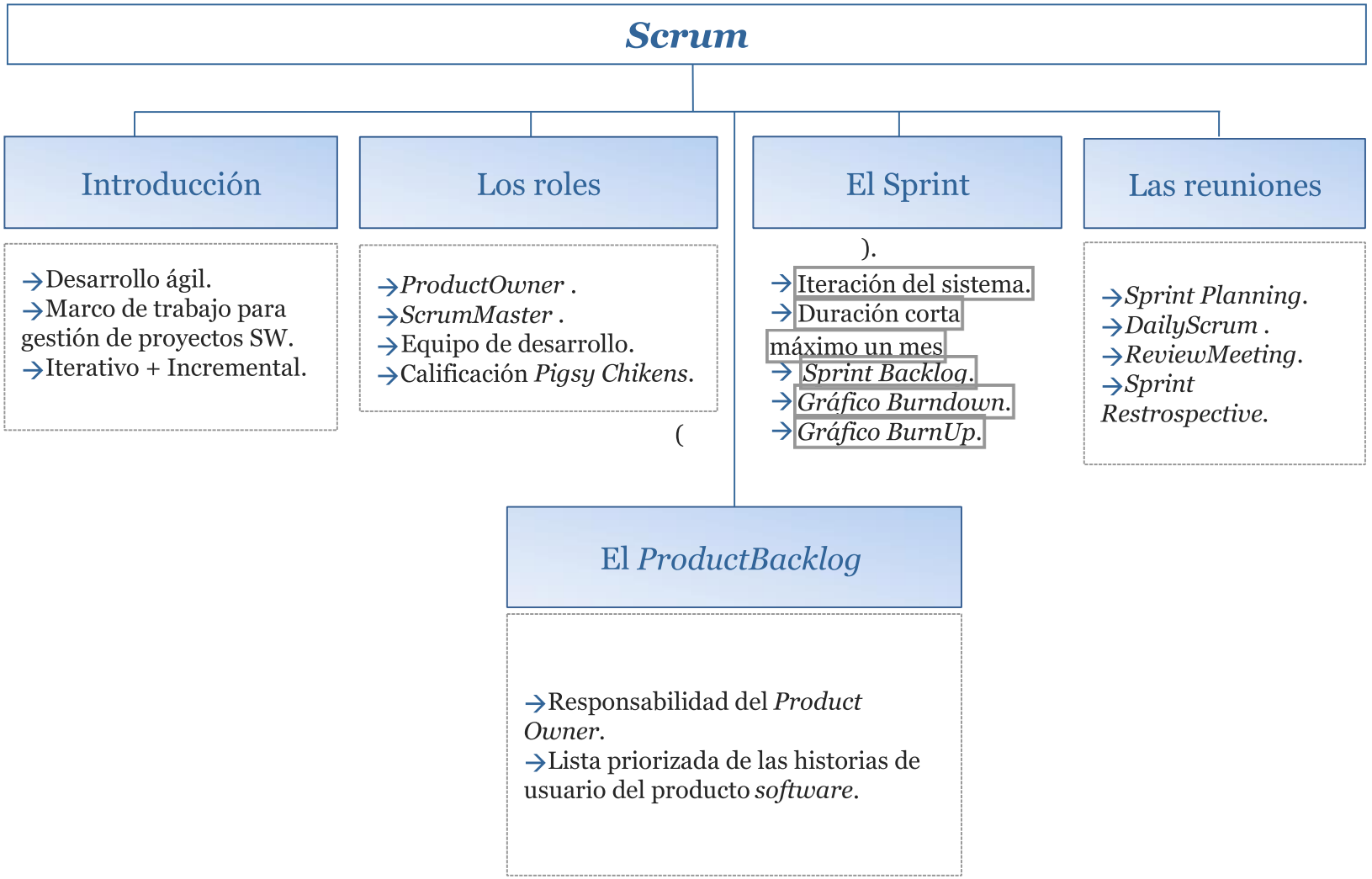
TEMA 10


Scrum
Introducción
Los roles
El Sprint
Las reuniones
→Desarrollo ágil.
→Marco de trabajo para gestión de proyectos SW.
→Iterativo + Incremental.
→ProductOwner .
→ScrumMaster .
→Equipo de desarrollo.
→Calificación Pigsy Chikens.
(
).
→Iteración del sistema.
→Duración corta máximo un mes
→ Sprint Backlog.
→Gráfico Burndown.
→Gráfico BurnUp.
→Sprint Planning.
→DailyScrum .
→ReviewMeeting.
→Sprint Restrospective.
El ProductBacklog
→Responsabilidad del Product Owner.
→Lista priorizada de las historias de usuario del producto software.

10.1. ¿Cómo estudiar este tema?

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

En este tema se estudian los principios y particularidades de *Scrum*, el marco de trabajo ágil por excelencia que se utiliza a nivel de proyectos *software*.

10.2. Introducción

Scrum, cuyo nombre tiene su origen en un tipo de jugada que se da durante un partido de rugby, es un **marco de trabajo** (del inglés *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.

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).

Con lo cual, hay que tener cierta cautela y mucho cuidado de no perder la agilidad por 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). García Oterino (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 a 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, 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 y donde la producción se mide por el tiempo que se pase 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 hará falta un facilitador que fomente buenas prácticas, sea capaz de resolver conflictos y promueva actitudes que ayudarán 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 lo que promocionan desde la parte de desarrollo de negocio, como los responsables de calidad… 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, la separación de todas estas divisiones organizativas, como por ejemplo, trabajadores de la cadena de montaje, personal de *marketing*, personal de calidad, etc., está claramente separado y no necesitan esa alineación que es necesaria cuando se desarrolla *software*.

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, 69).

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, 69-70):

- **Retraso**. 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 Spring y pila de producto**), aunque como para el resto de elementos vinculados a *Scrum*, en esta asignatura se utilizarán los términos originales en inglés.

  El retraso o *backlog*, como preferimos nombrarlo, 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 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 15 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**: (i) ¿qué hiciste desde la última reunión del **equipo de desarrollo**?; (ii) ¿con qué problemas te has encontrado a la hora de realizar tu trabajo?; y (iii) ¿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 asociado 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.

- **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.

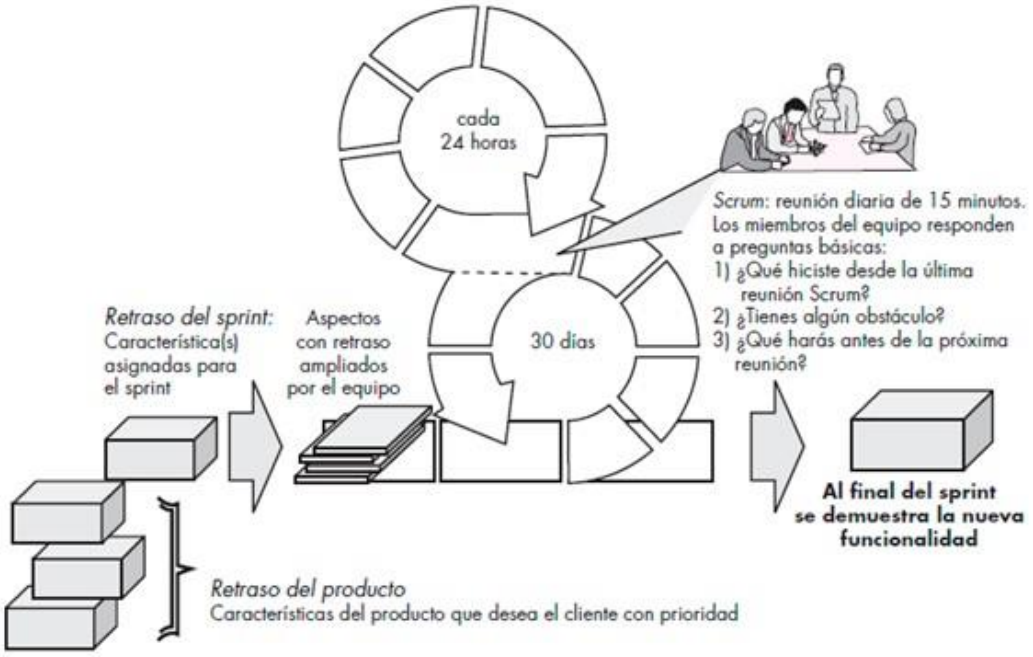


Figura 1. Flujo del proceso *Scrum* (Pressman, 2010)

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 en los requisitos que se puedan producir, 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.

10.3. Los roles

De acuerdo a *Scrum*, en todo proyecto *software* se darán tres roles (Garzás, Enríquez de S. e Irrazábal, 2013, 44-45):

- ***Product Owner.*** Este rol ya se describió en detalle en el tema anterior, por lo que simplemente recordar que 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. 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 es el responsable de asegurar que de que todo el equipo *Scrum* (no solamente el equipo de desarrollo) siga las prácticas *Scrum*. En *Scrum* no se da como tal la figura de jefe de proyecto que existe en la gestión de un proyecto *software* tradicional, y el *Scrum Master* es el rol que en cierto modo le representa, aunque con otras responsabilidades. 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.
- **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*.
- **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 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.

Por otro lado, los roles en *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.

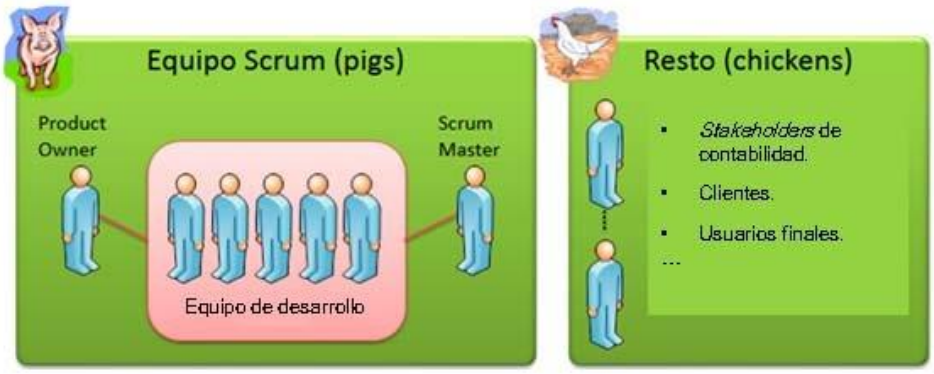


Figura 2. Roles *Pigs* y *Chickens*. Fuente: adaptada de http://slideplayer.es/slide/3057200/ .

10.4. *El Product Backlog*

El *product backlog* o pila de producto, como se le conoce en su traducción al castellano, 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.

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 que se van a ir encontrando nuevas funcionalidades a medida que se van 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 Backlog* 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 los más prioritarios aparecerán primero en el *Product Backlog*. En un primer momento, puede que no sea necesario priorizar todas las historias de usuario, pero si resulta conveniente identificar al menos las 15 o 20 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 más explicación de los contenidos que aparecen en el *Product Backlog*, como por ejemplo, prototipos de la interfaz de usuario, diagramas de la reglas de negocio y un resumen de los distintos usuarios que pueden operar con el sistema (Garzás, Enríquez de S. e Irrazábal, 2013, 48).

Para Kniberg (2007, 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 y lo normal es que contenga de 2 a 10 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 ya descritos con anterioridad.

- **Pruebas.** Este campo representa una descripción a 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 pseudo-código en las pruebas de aceptación.
- **Notas.** Este campo representa cualquier información adicional, comentario, aclaración, referencia a otras fuentes de información, etc., que pueda constituir un elemento de utilidad para una historia de usuario concreta.

Existen diversas herramientas que se podrían utilizar para crear un *Product Backlog*. Kniberg (2007) por ejemplo, sugiere utilizar Jira para almacenar el *Product Backlog* de un proyecto y Microsoft Excel también es una herramienta excelente que ofrece muchas posibilidades para la gestión y mantenimiento del *Product Backlog* (en realidad, hay mucho desconocimiento de las cantidad de funcionalidades que proporciona esta herramienta para muchos temas vinculados con la gestión de proyectos *software*).

Tal y como destaca Garzás (2013, 28-29), la hoja de ruta se va a complementar 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, 29).

Id.	Nombre	Importancia	Estimación inicial	Pruebas	Notas
1	Depósito	30	5	Entrar, abrir una página de depósito, depositar 10€, ir a la página de balance y comprobar que se ha incrementado en 10€.	Necesita aclaración con diagramas UML. No hay que preocuparse todavía por el cifrado de datos.

Id.	Nombre	Importancia	Estimación inicial	Pruebas	Notas
2	Ver historial de transacciones	18	8	Entrar, ver transacciones. Realizar un depósito de 10€. Ir a transacciones y comprobar que se ha actualizado con el nuevo depósito.	Utilizar paginación para no hacer consultas muy grandes a la base de datos. Diseño similar a la página de usuario.

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, 28).

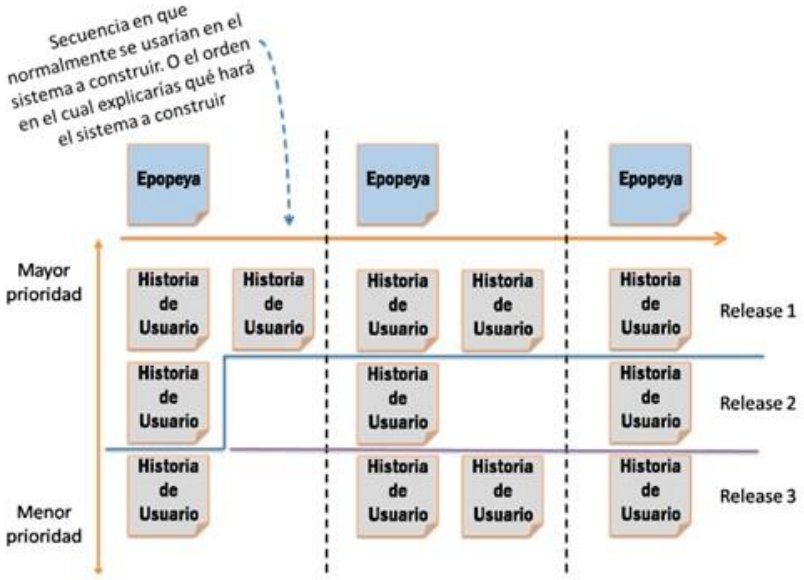


Figura 3. Mapa de historias de usuario (Garzás, 2013).

10.5. El 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 y 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.

Para *Scrum*, las duraciones de los *Sprints* han de ser cortas, en periodos de tiempo que abarcan entre 1 y 4 semanas (máximo de 30 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, 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 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, 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, y es por ello 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*.

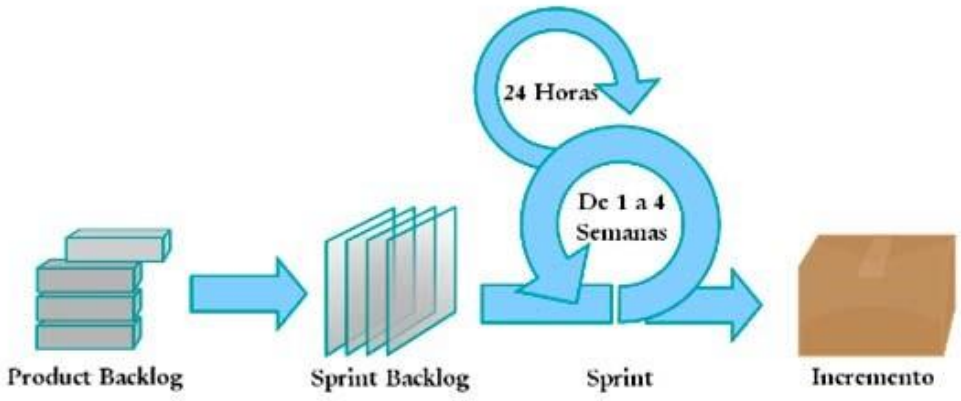


Figura 4. El *Sprint* dentro del proceso *Scrum* (Garzás, Enríquez de S. e Irrazábal 2013).

Como se destaca en (Garzás, Enríquez de S. e Irrazábal, 2013, 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. e Irrazábal (2013, 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).

Figura 5. Desglose de las historias de usuario en tareas (Garzás, Enríquez de S. e Irrazábal (2013).

Las tareas identificadas se incluirán 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*.

Tabla 2: Ejemplos de tareas que podrían formar parte de un Sprint (Garzás, 2013).

Tarea	Descripción
Diseño de la historia de usuario	Esta tarea se utiliza para promover un debate acerca de cómo se va a implementar la historia de usuario.
Implementación de la historia de usuario	Esta tarea se puede utilizar para definir las interfaces y los métodos necesarios para implementar la funcionalidad requerida en la historia de usuario. Podrían existir varias tareas de este tipo.
Pruebas unitarias asociadas a la historia de usuario (TDD)	Realizar pruebas unitarias vinculadas a la historia de usuario debería ser una tarea obligatoria.
Pruebas de aceptación asociadas a la historia de usuario	Esta tarea se utiliza para llevar a cabo las pruebas automáticas de aceptación que van a facilitar la validación de la historia de usuario por parte del cliente y/o de un usuario final.
Requisitos no funcionales de la historia de usuario	Esta tarea se utiliza para considerar los requisitos de seguridad, rendimiento, escalabilidad, etc., que van asociados a la historia de usuario y que podrían derivar en la aparición de nuevas tareas para mantener la calidad del producto *software*.
Revisiones de código	Esta tarea se utilizar para revisar el código, pues dicha labor constituye una tarea en sí.
Refactorización de código	Esta tarea se utiliza en cada historia de usuario para evitar complicar el código más de lo necesario.
Emular interfaces	Esta tarea se utiliza para emular interfaces, en el caso de que se sufra mucho retraso en la implementación de las mismas.
Pruebas exploratorias de la historia de usuario	Esta tarea se utiliza para realizar pruebas *ad-hoc* de la historia de usuario
Corrección de errores de la historia de usuario	Esta tarea se utiliza para la resolución de errores o incidencias cuando se considere que, debido al preocupante número de los mismos, ha de crearse una tarea en sí para su resolución.
Verificación de errores de la historia de usuario	Esta tarea se utiliza para la verificación de errores cuando se considere que, debido al preocupante número de los mismos, ha de crearse una tarea en sí para su verificación.
Demo de la historia de usuario	Esta tarea se utiliza para realizar una demostración interna de la implementación de la historia de usuario al equipo de desarrollo. La demo se realizará normalmente en la *Daily Scrum*, una vez implementada la historia de usuario.
Actualizar la wiki o el repositorio de documentos	Esta tarea se utiliza para documentar el diseño y los resultados de la historia de usuario.
Documentación de usuario	Esta tarea se utiliza para generar los manuales de usuario, manuales de instalación, etc. A veces esta labor se puede integrar dentro de otra tarea y no crear una específica.

A veces, puede resultar muy complicado desglosar las historias de usuario en tareas por 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 los modelos UML, por ejemplo, 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. e Irrazábal (2013, 51) indican las diferencias significativas que los diferencian:

- Todas las actividades de diseño, integración o pruebas se hace de manera continua en el *Sprint* y sin ningún tipo de secuencia, es decir, que 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 a ese momento. Tal y como se comenta en Garzás, Enríquez de S. e Irrazábal (2013, 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 un tipo de *Sprint* 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 para Schawber (2008), co-autor de *Scrum*, es una forma errónea de llamar 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 a Ambler (2008), se encarga de:

- Estudio de los requisitos 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 la metodología *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.

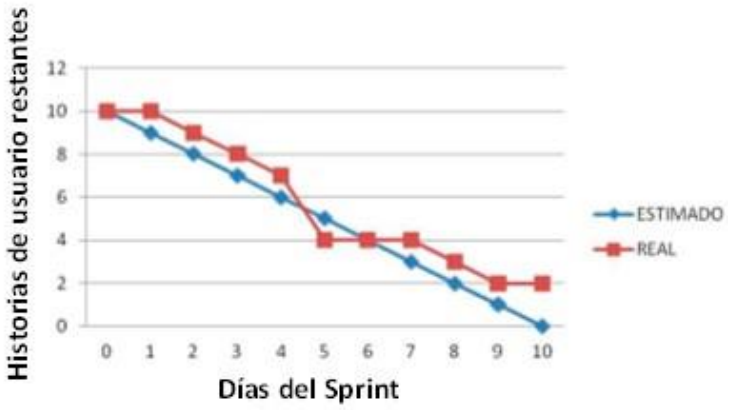


Figura 6. Ejemplo de gráfico *BurnDown* (Garzás, Enríquez de S. e Irrazábal 2013)

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. e Irrazábal, 2013, 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. e Irrazábal, 2013, 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*.

Figura 7. Ejemplo de gráfico BurnUp (Garzás, Enríquez de S. e 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 de cuadrar las barras, flechas, recursos, dimensión temporal… que se muestran en el diagrama que en 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, de hecho, 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, 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.

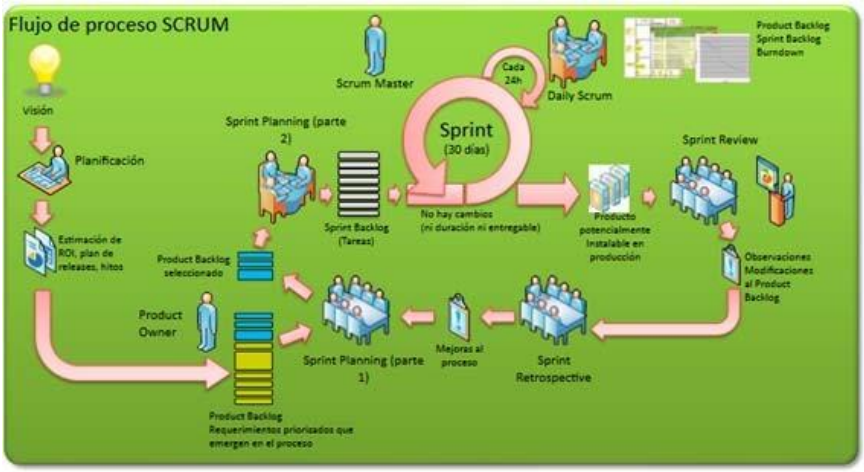


Figura 8. Flujo del proceso *Scrum*. Fuente: http://slideplayer.es/slide/3057200/ .

10.6. Las reuniones

Como ya se ha ido pudiendo 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 del proyecto. En *Scrum*, las reuniones se van a clasificar en las siguientes categorías (Garzás, Enríquez de S. e Irrazábal, 2013, 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*. Se aconseja que la duración de esta reunión no supere las 8 horas.

- ***Daily Scrum***. Aunque ya introducida al comienzo de este tema, en la visión general de *Scrum*, recordar que esta reunión se realiza diariamente a lo largo de todo el proyecto y su duración suele ser de unos 15 minutos. En la reunión *Daily Scrum* participan el *Scrum Master*, el *Product Owner* y el equipo de desarrollo. 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.

- ***Sprint Review***. Esta reunión se realiza al final de cada *Sprint*. 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. Se aconseja que la duración de esta reunión no supere las 4 horas.

- ***Sprint Retrospective.*** Esta reunión también se realiza al final de un *Sprint*, pero a diferencia de la *Sprint Review*, esta reunión no tiene que hacerse en cada *Sprint*, solamente cuando se estime necesario. En la reunión *Sprint Retrospective* deberían participar el *Scrum Master* y todos los integrantes del equipo *Scrum* para poder intercambiar impresiones sobre el *Sprint* que se acaba de revisar.

  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. Al igual que las reuniones *Sprint Review*, se aconseja que la duración de una reunión *Sprint Retrospective* no supere las 4 horas.

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

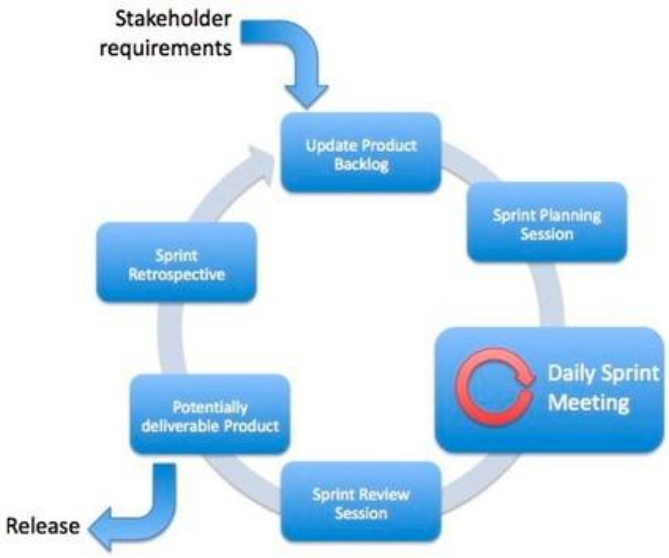


Figura 9. Reuniones establecidas en *Scrum*. Fuente: http://www.scruminstitute.org/Introduction_to_Scrum_A_Real_World_Example.php .