Capítulo 5
Construir y priorizar el Product Backlog

1. ¿Por qué invertir en el Product Backlog?

Estemos en un contexto Ágil o no, es muy evidente que para que un proyecto tenga éxito debemos tener una buena visión del sistema o producto que vamos a desarrollar, a través de la definición adecuada de la necesidad a la que responde.

Por lo tanto, en Scrum no existe un proyecto exitoso sin un Product Backlog bien construido e inteligentemente priorizado. Evidentemente, para esto el Product Owner juega un rol fundamental, por su conocimiento del negocio y/o del mercado, así como por su capacidad de separar, ordenar y transcribir eficazmente lo que esperan los usuarios del software.

Ahí donde teníamos la costumbre como "Cliente" de escribir las especificaciones funcionales detalladas que expresaban las necesidades, nos tenemos que familiarizar con un nuevo método. Esta familiarización implica un tiempo de aprendizaje y se debe compartir por todo el conjunto de las partes integrantes del proyecto, lo que significará formación, asimilación y puesta en marcha de nuevas técnicas.

Por lo tanto en este capítulo vamos a recorrer los conceptos y métodos que permiten construir, ordenar, afinar y gestionar de manera cotidiana el Product Backlog.

2. La pieza básica del Product Backlog: la User Story

Por increíble que parezca, Scrum no atiende a un formato o contenido concreto del Backlog. Sin embargo, se impone un formalismo y este se encuentra casi de manera sistemática en todos los proyectos Scrum: se trata de la noción de **User Story**, que se toma prestada de XP.

Una User Story es una descripción sencilla y comprensible de un elemento de funcionalidad con valor "de negocio" para el sistema. Por tanto, se debe expresar correctamente desde el punto de vista del usuario. Se guía por las respuestas a estas tres preguntas:

– ¿Quién realiza la petición o quién se beneficia de la petición? (rol usuario)
– ¿Cuál es la petición? (la necesidad)
– ¿Cuál es el valor para el negocio que se deriva de la realización de esta necesidad?

A continuación se enumeran algunos ejemplos (veremos más adelante cómo redactar correctamente las User Stories):

"Deseo que el IVA se calcule automáticamente en las facturas".

"Deseo poder eliminar los clientes que hayan realizado pedidos hace más de un año".

Además, también se usa la noción de **Epic Story**. Se puede considerar una Epic como una "**macro User Story**", es decir, engloba en su definición un subconjunto de User Stories.

Por ejemplo, en relación con las User Stories descritas anteriormente, podemos tener las siguientes Epics:

"Deseo que mis facturas se creen automáticamente".

"Deseo tener una gestión de mis clientes".

Por lo tanto, es el conjunto de las Epic y User Stories lo que conforma el Product Backlog.

En la práctica, resulta útil agrupar las Epic o User Stories (en particular para la priorización): para hacer esto habitualmente se usan los conceptos de **Temas** o **Actividades**, que son los grupos temáticos de Stories.

3. ¿Cómo redactar las User Stories y Epics?

3.1 Regla de las 3C

Para guiar la redacción de las User Stories, Ron Jeffries propone un principio muy sencillo que es necesario tener en cuenta. Se trata de la regla de las 3C:

Tarjeta (Card, en inglés)	La Story se escribe en una tarjeta de tamaño muy reducido. Estas fichas pueden contener notas (estimación, etc.).
Conversación	Los detalles de la Story se expresarán durante las conversaciones con el Product Owner.
Confirmación	Se describen las pruebas de validación para la Story (servirán para validar que se ha realizado correctamente).

En la práctica, la Story se escribirá en una pequeña tarjeta de color colgada en un tablero (si recuerda, este principio se hereda de Kanban), bajo la siguiente forma:

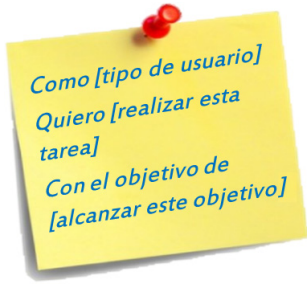
– Empezamos con un título.

– Se añade una descripción resumida de la "tarea de usuario" y de sus objetivos.

– Se pueden agregar todas las notas, dibujos o informaciones útiles. Por ejemplo: estimaciones, indicador de prioridad o valor de negocio.

– De manera ideal se anotan los criterios de validación (por ejemplo en el reverso, si no hay espacio suficiente).

De una forma muy sencilla, resultará en algo parecido a lo que se muestra a continuación:



La tarjeta es un concepto de información compartida extremadamente resumida y eficaz, pero tan pronto como se añadan datos adicionales como consecuencia del análisis, preste atención con no sobrecargarla. Con demasiada rapidez, vemos cómo llega la necesidad de pasar por una herramienta más sofisticada (e informatizada) para gestionar la información: hablaremos de esto más adelante.

3.2 Redactar una buena User Story: el principio INVEST

Cuando se entra en el proceso de redacción, rápidamente nos podemos perder. En este caso hay otro principio muy útil que le puede guiar: se trata del principio INVEST.

Indica que una buena User Story debe ser:

- **Independiente** del resto de historias de usuario (en la medida de lo posible).
- **Negociable**: se debe poder discutir con el equipo encargado de la realización del producto, fundamentalmente durante la estimación.
- Origen del **Valor**: debe tener valor para el cliente o el usuario.
  - Una User Story no describe los objetivos técnicos.
- **Estimable**: se debe poder estimar por el equipo de realización con un riesgo de error bajo.
  - Con este objetivo se debe redactar de manera clara y comprensible.

- De un tamaño lo **suficientemente pequeño** con el objetivo de facilitar su estimación y garantizar que se puede diseñar, desarrollar y probar dentro de un Sprint.
  - Si la Story es demasiado grande sin duda será una Epic, que se deberá descomponer de manera más fina antes de la realización.
- **Verificable** (del inglés, Testable): una User Story se debe acompañar de criterios de validación que permitan su validación.
  - Veremos en el capítulo dedicado a las pruebas cómo formalizar esto.

3.3 Errores habituales

Para redactar correctamente nuestras Stories, es muy instructivo conocer los errores que se deben evitar:

- Detallar demasiado la descripción y entrar demasiado pronto en un nivel de detalle final.

  No olvide que no se busca hacer especificaciones detalladas completas antes del desarrollo, como en los métodos tradicionales.
- Olvidar la noción de usuario en la descripción o utilizar actores demasiado genéricos: esto puede crear ambigüedades e imprecisiones.

  Ejemplo:
  - Como cliente suscrito, quiero poder consultar las tarifas de los billetes de avión.
  - Como cliente estándar, quiero poder consultar las tarifas de los billetes de avión.

  En lugar de:
  - Como usuario, quiero poder consultar las tarifas de los billetes de avión.

  En efecto, es probable que queramos ofrecer ofertas o tarifas diferentes a los suscritos y no al resto de personas que consulten las tarifas. Esto no aparece en la segunda descripción.
- Partir de un diseño funcional detallado y descomponerlo en User Stories, basándose en su estructura textual. Este puede ser un enfoque cómodo para un equipo debutante, pero no es una práctica recomendada (veremos más adelante un método práctico para inicializar el Product Backlog).

– Tener un nivel de detalle constante. Este punto es muy importante, porque está en el núcleo del enfoque ágil. El contenido de las User Stories puede evolucionar a lo largo del tiempo, a medida que se afina en el entendimiento de la necesidad y del análisis. De manera típica, una Story cuya realización se prevé en varios Sprints solo se describirá de manera muy sencilla (título y descripción resumida), mientras que una User Story cuya realización está cerca de terminarse debe completarse con pruebas de aceptación, ejemplos, reglas de gestión, maquetas de las pantallas, etc.
– Asimilar una User Story a un **Use Case**. Aunque la comparación entre las dos nociones normalmente pueda ser útil, son dos enfoques con características sensiblemente diferentes.

Sin entrar a un nivel de detalle demasiado fino, se puede decir que una modelización basada en Use cases es una descripción de procesos (por definición, un Use case representa una secuencia de acciones que un sistema o cualquier otra entidad puede realizar interactuando con los actores del sistema) que se basa normalmente en diagramas UML, como en el siguiente ejemplo: