

# HISTORIAS DE USUARIO

# Historias de Usuario

Ingeniería de Requisitos Ágil

Versión 3.01 – Agosto 2022

Imagen de cubierta: Miguel Angel Sobrino.

Autores: Alexander Menzinsky, Gertrudis López, Juan Palacio, Miguel Ángel Sobrino, Rubén Álvarez y Verónica Rivas.

Producción: Scrum Manager®.

Información de derechos y licencia de uso registrados en Safe Creative. ID: [2009135322450](https://www.safe-creative.com/2009135322450).

# Índice

<b>Ingeniería de requisitos ágil.....</b>	<b>4</b>
<i>Historias de Usuario.....</i>	<i>4</i>
<i>De Temas y Epics a Tareas.....</i>	<i>7</i>
<b>Información necesaria en historias de usuario.....</b>	<b>9</b>
<i>Descripción.....</i>	<i>12</i>
<i>Valor de negocio.....</i>	<i>14</i>
<i>Estimación.....</i>	<i>15</i>
<i>Prioridad.....</i>	<i>17</i>
<i>Criterios de aceptación.....</i>	<i>20</i>
<i>Requerimientos no funcionales.....</i>	<i>23</i>
<i>Definición de hecho.....</i>	<i>25</i>
<b>Historias de usuario de calidad.....</b>	<b>27</b>
<b>División de historias de usuario.....</b>	<b>30</b>
<i>Estrategia 0: construir un spike.....</i>	<i>33</i>
<i>Estrategia 1: división por pasos de flujo de trabajo.....</i>	<i>34</i>
<i>Estrategia 2: división por reglas de negocio.....</i>	<i>36</i>
<i>Estrategia 3: división por happy/unhappy flow.....</i>	<i>37</i>
<i>Estrategia 4: división por opciones/plataformas de entrada.....</i>	<i>38</i>
<i>Estrategia 5: división por tipos de datos o parámetros.....</i>	<i>39</i>
<i>Estrategia 6: división por operaciones.....</i>	<i>40</i>
<i>Estrategia 7: división por casos/escenarios de test.....</i>	<i>41</i>
<i>Estrategia 8: división por roles.....</i>	<i>43</i>
<i>Estrategia 9: división por optimizar ahora o más tarde.....</i>	<i>44</i>
<i>Estrategia 10: división por compatibilidad de navegador.....</i>	<i>45</i>
<b>Otras formas de toma de requisitos.....</b>	<b>46</b>

<i>Casos de uso</i> .....	46
<i>Requisitos funcionales</i> .....	47
<b>Historias técnicas</b> .....	<b>48</b>
<b>User Story Mapping</b> .....	<b>50</b>
<b>Apéndices</b> .....	<b>53</b>
<i>Historias de usuario en otros ámbitos</i> .....	53
<i>Herramientas digitales</i> .....	54
<i>Epics</i> .....	57
<i>Resumen Historias de Usuario</i> .....	58
<i>Bibliografía</i> .....	59
<i>Tabla de ilustraciones</i> .....	60

## **Mejora continua y control de calidad Scrum Manager**

Gracias por elegir los servicios de formación de Scrum Manager.

El control de calidad de Scrum Manager se basa en las valoraciones de sus estudiantes. Con tu opinión nos ayudas a mantener el nivel de nuestros materiales, cursos, centros y profesores.

Si has participado en una actividad de formación auditada por Scrum Manager®, te rogamos y agradecemos que valores la calidad de la formación recibida. La información que se recoge es anónima.

Puedes enviarnos tus comentarios desde el «Área de miembros» en <https://scrummanager.com>.

# Ingeniería de requisitos ágil

---

## Historias de Usuario

La ingeniería de requisitos dispone de dos medios de comunicación para recoger y transmitir requisitos, cada uno con sus características:

### 1. Comunicaciones escritas.

- Registran la información de forma permanente.
- Son más fáciles de compartir con grupos y personal remoto.
- Se pueden pensar y revisar bien y de forma completa.
- Son más susceptibles de malinterpretarse.

### 2. Comunicaciones verbales.

- Permiten recibir *feedback* de inmediato.
- Son dinámicas; la conversación se adapta para maximizar su eficiencia.
- Se adaptan con facilidad a nuevos desarrollos.
- Generan ideas nuevas.
- Permiten alcanzar comprensión y claridad comunes con menos esfuerzo.

Las historias de usuario se usan, en el contexto de la ingeniería de requisitos ágil, como una herramienta de comunicación que combina las fortalezas de ambos medios: escrito y verbal. Describen, en una o dos frases, una funcionalidad de *software* desde el punto de vista del usuario, con el lenguaje que éste emplearía.<sup>1</sup> El foco está puesto en qué necesidades o problemas soluciona lo que se va a construir.

Su origen viene de la metodología *eXtremeProgramming* (programación extrema, abreviado normalmente como *XP*), donde las historias de usuario deben ser escritas por los clientes. *XP* fue creada por Kent Beck y descrita por primera vez en 1999, en su libro *eXtreme Programming Explained*. Hoy día se utilizan en la mayoría de los métodos ágiles, incluyendo *scrum*.

---

<sup>1</sup>Mike Cohn, 2004.

Las historias de usuario son una herramienta que agiliza la administración de requisitos, reduciendo la cantidad de documentos formales y tiempo necesarios. Forman parte de la fórmula de captura de funcionalidades definida en 2001 por Ron Jeffries de las tres Cs:

**Card:** cada historia de usuario se reduce hasta hacerla fácil de memorizar y de sintetizar en una tarjeta o post-it. La tarjeta sirve como recordatorio y promesa de una conversación posterior.

**Conversation:** el equipo de desarrollo y el propietario del producto añaden criterios de aceptación a cada historia poco antes de su implementación. Los cambios son bienvenidos en agilidad, por lo que no tiene sentido profundizar en estos detalles antes. La situación puede variar mucho desde el momento en el que se sintetiza la funcionalidad en la tarjeta hasta que se implementa.

**Confirmation:** el propietario del producto o usuario de negocio confirma que el equipo de desarrollo ha entendido y recogido correctamente sus requisitos revisando los criterios de aceptación. A veces se pueden presentar transformados en escenarios de pruebas.

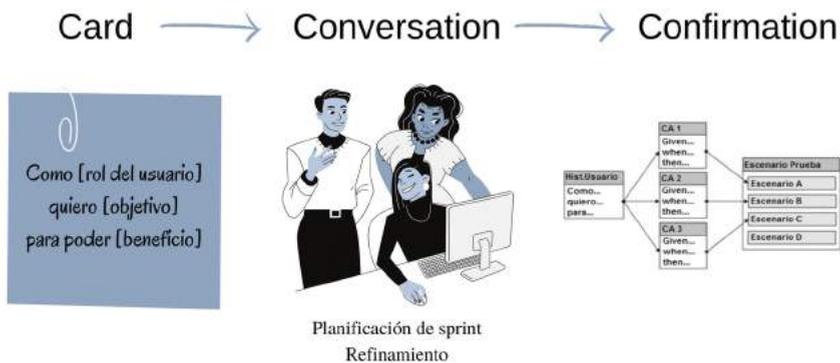


Ilustración 1: gráfico con las tres fases de la fórmula de las tres Cs.

## **Ventajas de las historias de usuario:**

- Proporcionan la documentación necesaria fomentando a la vez el debate.
- Fomentan la colaboración entre todos interesados y el equipo ágil.
- Se escriben en el lenguaje del usuario, manteniendo así una relación cercana con el cliente.
- Involucran y captan al cliente para el proceso y para el producto.
- Por su naturaleza son independientes.
- Facilitan la planificación e implementación.
- Son ideales para proyectos con requisitos volátiles o no muy claros.
- Fomentan aplazar los detalles no imprescindibles.
- Son pequeñas y por tanto fáciles trabajar.
- Permiten dividir los proyectos en pequeñas entregas.
- Permiten estimar fácilmente su esfuerzo de desarrollo.
- Funcionan para el desarrollo iterativo, ya que al ser pequeñas representan requisitos del modelo de negocio que pueden implementarse en poco tiempo (días o semanas).
- Necesitan poco mantenimiento.

## De Temas y *Epics* a Tareas

En la jerarquía de los requisitos ágiles y por encima de las historias de usuario se encuentran los *epics* y los temas. Tienen en común que todos se centran en describir *qué* se va a construir.

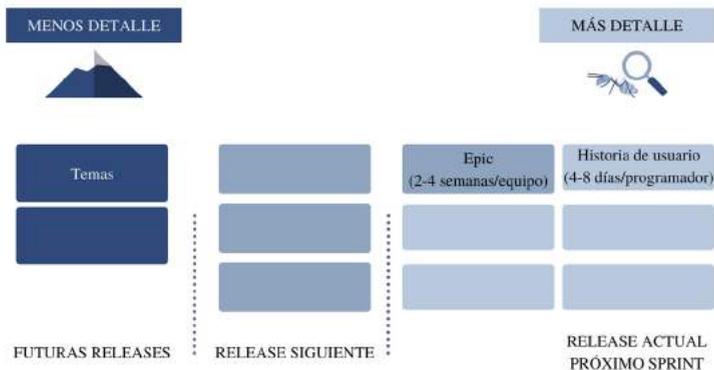


Ilustración 2: granularidad de la pila de producto.

**Epic:** una historia de usuario de gran tamaño o alta granularidad<sup>2</sup>, y que tiene por tanto un mayor grado de incertidumbre. Marcar una historia como epic implica que no puede completarse de una sola vez o en un único *sprint*. Lo normal es que el equipo de desarrollo lo descomponga cuando se acerque el momento de su implementación. Las historias de usuario resultantes estarán íntimamente relacionadas y su menor tamaño permitirá gestionarlas de forma ágil, estimando mejor el tiempo requerido para completarlas y siguiendo su avance con detalle.

**Tema:** una colección de *epics* e historias de usuario relacionadas que describen un sistema o subsistema. Se trata de un elemento que forma parte de la visión del producto, más que una funcionalidad.

Por ejemplo: en un sistema de software para gestión contable, el conjunto de *epics* «altas, bajas y mantenimiento de clientes», «facturaciones puntuales y recurrentes», «consultas de navegación y acciones de fidelización», «pedidos» y «devoluciones» se podrían denominar como el tema de la «gestión de clientes».

**Tareas:** están por debajo de las historias de usuario. Describen *cómo* construir en lugar de *qué*. Resultan de la descomposición de las historias de usuario en unidades de trabajo adecuadas para gestionar y seguir el avance de su ejecución.

<sup>2</sup>La granularidad es el nivel de detalle de cada elemento en la pila de producto. Las *epics* tienen una granularidad alta o gruesa; las tareas, una granularidad baja o fina. A mayor prioridad, menor granularidad.

## Requisitos ágiles

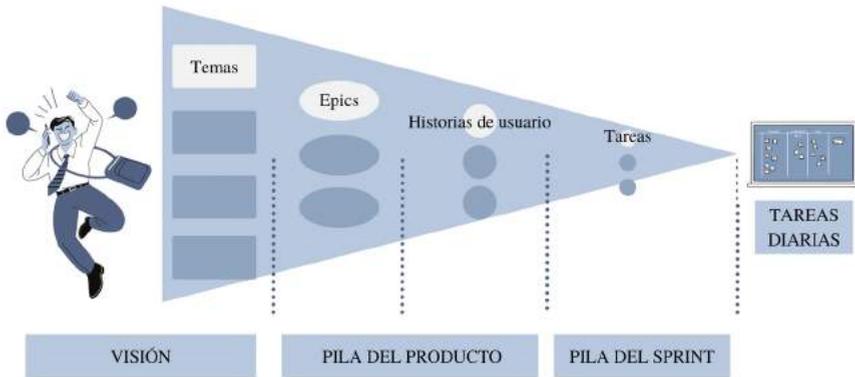


Ilustración 3: gráfico con los cuatro niveles de tamaño con que trata los requisitos la gestión ágil.

En scrum y en los métodos ágiles en general, una pila de producto puede contener tanto historias de usuario como temas y epics. Las tareas forman la pila del sprint.

Al final de la pila del producto está lo menos prioritario: los epics, historias de usuario de gran tamaño y los temas, que describen requisitos más generales de la visión del producto. Cada uno de éstos se descompondrá en elementos menores a medida que avancen en la pila.

Las historias susceptibles de entrar en el próximo sprint necesitan mayor detalle para que el equipo las pueda desglosar en tareas durante la reunión de planificación. Por tanto, las historias de usuario son el elemento que debe encabezar la pila de producto.

# Información necesaria en historias de usuario

---

Para decidir qué información incluir en una historia de usuario es preferible no adoptar formatos rígidos. Los resultados de scrum y agilidad no dependen de las formas, sino de la institucionalización de sus principios y la implementación adecuada a las características de la empresa y del proyecto. Por tanto, aparte de tres campos que se consideran necesarios, se puede incluir cualquier campo que proporcione información útil para el proyecto. Recordemos que el objetivo de las historias de usuario es construir un entendimiento compartido.

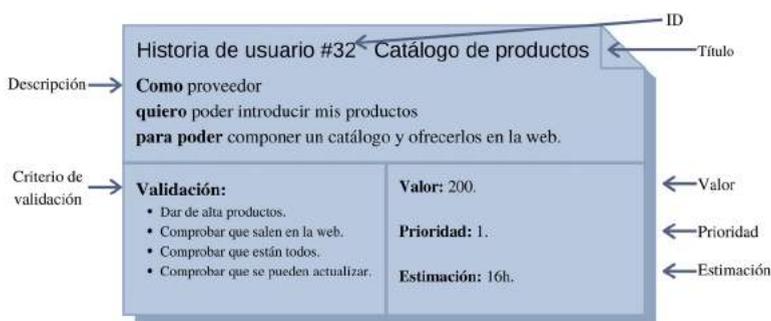


Ilustración 4: ejemplo de una tarjeta de historia de usuario.

## Campos esenciales

Los campos que se consideran más necesarios para describir las historias de usuario son:

- **Descripción:** síntesis de la historia de usuario. El estilo puede ser libre pero debe responder a tres preguntas: ¿quién se beneficia? ¿qué se quiere? y ¿cuál es el beneficio?

Mike Cohn recomienda seguir el siguiente patrón para garantizar que la funcionalidad esté descrita a un alto nivel y de manera breve.

**Como** [rol del usuario], **quiero** [objetivo], **para poder** [beneficio].

- **Estimación:** aproximación del esfuerzo necesario (en tiempo ideal) para implementar la historia de usuario. Puede estimarse usando unidades de desarrollo (puntos de historia<sup>3</sup>), si el equipo lo prefiere y está familiarizado con este sistema.
- **Prioridad:** se indica siguiendo un sistema que permita establecer el orden de implementación de las historias.

## Otros campos

Dependiendo del tipo de proyecto, el funcionamiento del equipo y la organización, pueden ser aconsejables otros campos como:

- **ID:** identificador único de la historia de usuario, funcionalidad o trabajo.
- **Título:** título descriptivo de la historia de usuario.
- **Valor de negocio:** valor (normalmente numérico) que aporta la historia de usuario al cliente o usuario. El objetivo del equipo es maximizar el valor y la satisfacción percibida por el cliente en cada iteración. Este campo servirá junto con la estimación para decidir la prioridad de implementación.
- **Criterio de aceptación:** pruebas de aceptación consensuadas con el cliente o usuario. A veces se transforman en pruebas que el código debe superar para dar como finalizada la implementación.
- **Requerimiento no funcional:** cualidades generales y restricciones, como la usabilidad o la seguridad, que afectan a aplicaciones y sistemas enteros, y por tanto a las historias de usuario individuales.
- **Definición de hecho** (en inglés DoD, *Definition of Done*): incluye las actividades o criterios necesarios para dar por terminada una historia de usuario (desarrollada, probada, documentada...), según lo convenido por el propietario de producto y el equipo.
- **Dependencias:** una historia de usuario no debería depender de otra, pero a veces es necesario mantener la relación. Este campo contendría los identificadores de otras historias de las que depende.
- **Persona asignada:** cuando queramos sugerir la persona que pueda implementar la historia de usuario. Recordar que en scrum el equipo se autogestiona y es quien distribuye y asigna las tareas.
- **Sprint:** puede ser útil para organización del propietario del producto incluir el número de sprint en el que se prevé construir la historia.
- **Riesgo:** riesgo técnico o funcional asociado a la implementación de la historia de usuario.
- **Módulo:** módulo del sistema o producto al que pertenece.
- **Observaciones:** para enriquecer o aclarar la información, o cualquier otro uso necesario.

---

<sup>3</sup>Los puntos de historia son unidades institucionalizadas al principio del proyecto que representan el tiempo teórico de desarrollo por persona.

Historia de Usuario	
Número: 1	Usuario: Cliente
Nombre historia: Cambiar dirección de envío	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 2	Iteración asignada: 1
Programador responsable: José Pérez	
<b>Descripción:</b> Como cliente quiero cambiar la dirección de envío de un pedido para que me pueda llegar a casa o a la oficina	
<b>Validación:</b> El cliente puede cambiar la dirección de entrega de cualquiera de los pedidos que tiene pendiente de envío	



Ilustración 5: ejemplos de una historia de usuario.

Mike Cohn comenta que, si bien las historias de usuario son lo bastante flexibles como para describir la funcionalidad de la mayoría de los sistemas, no son apropiadas para todo. Si por cualquier razón se necesita expresar algún requisito de manera diferente, es recomendable hacerlo.

Por ejemplo: la maquetación de una interfaz se suele describir con capturas de pantalla. Sería la mejor manera de transmitir el diseño que queremos darle a una aplicación.

## Descripción

El pensamiento de las personas se estructura siguiendo una narrativa, una historia; así es como entendemos el mundo. Es también una manera eficaz de retener conocimiento y tomar decisiones. Al usar personajes y empatizar con ellos podemos entender su punto de vista y priorizar de manera más acertada.

Una técnica útil en este sentido es la creación de *user personas*, muy utilizadas en el ámbito de la *user experience* (UX, «experiencia de usuario» en español). El objetivo es comprender el punto de vista de los usuarios: construir una serie de personajes de ficción que representan, mediante arquetipos, una muestra realista de la audiencia clave de nuestra web o aplicación.

Estas *user personas* servirán como «rol del usuario» en el patrón que vimos antes para la descripción de la historia:

**Como [rol del usuario], quiero [objetivo], para poder [beneficio].**

- **Como [rol del usuario]:** ¿para quién estamos construyendo? No nos referimos a un cargo o un rol profesional, sino a la persona tras la necesidad descrita, según esté representada en su *user persona*. El equipo debe tener un entendimiento compartido de quién es, cómo trabaja, cómo piensa y siente; en definitiva, sentir empatía.
- **Quiero [objetivo]:** ¿qué es lo que está intentando conseguir realmente? Es importante recordar que en este campo se explica su intención, no la funcionalidad que usará.
- **Para poder [beneficio]:** ¿cómo encaja su necesidad inmediata en lo que le rodea? ¿cuál es el beneficio general que está tratando de lograr? ¿cuál es el gran problema que necesita resolver? Se trata de un beneficio que va relacionado con la visión del producto.

Estos personajes facilitan mantener las conversaciones centradas en los usuarios, dándoles un nombre y una personalidad para empatizar con el grupo de personas al que representan. Si el público objetivo de nuestra web tiene unas características determinadas, hacer su user persona y referirnos a ella por su mote nos permite imaginar mejor cómo interactuará con el sistema, ayudando al equipo a tomar decisiones más acertadas y ofrecer soluciones más adecuadas a sus necesidades reales.

Se suele representar a los usuarios en una ficha tipo tríptico incluyendo, por ejemplo:

- **Nombre y apodo:** una manera rápida de identificar y reconocer al personaje.
- **Datos demográficos:** ayudan a clasificar al personaje dentro de un segmento de mercado.
- **Descripción:** un párrafo breve que nos ayude a conocer al personaje.
- **Objetivos:** necesidades del usuario que distinguen su comportamiento del resto.



Ilustración 6: ejemplo de user persona.

## Valor de negocio

El campo «valor» de una historia de usuario representa el valor de negocio que aportará una vez realizada. Aporta información más profunda que la razón de la historia, pues está ligado a la lógica del negocio. Una manera de estimarlo es considerar cuánto estaría dispuesto a pagar el cliente por la funcionalidad.

Tal como funcionan scrum y otras metodologías ágiles, entregar valor al cliente significa centrarnos en resolver los problemas de alguien o en dar beneficios a alguien. Es muy importante conocer el valor de negocio para priorizar correctamente y focalizar al equipo. También existen otros factores a tener en cuenta, por lo que valor y prioridad son dos informaciones independientes.

El valor se mide con una escala arbitraria y normalmente numérica. Ha de ser una con la que el propietario del producto y los usuarios se sientan cómodos, por ejemplo la serie de Fibonacci, o series de números naturales como del 1 al 10, del 1 al 100 o del 1 a 1.000.000.

Una forma muy estimulante es repartir billetes de Monopoly por el valor total del proyecto a cada uno de los usuarios de negocio involucrados, y que éstos repartan el dinero en función del valor que le atribuyen a cada historia de usuario. El valor final es la suma de los billetes de todos. También se puede dividir el resultado por 100 o 1000 para que sea un número más manejable.

Benoît Pointet y Thomas Botton proponen estimar el valor de la misma manera que el equipo estima el esfuerzo, mediante cartas de *planning poker* con la serie de Fibonacci. El valor, igual que el esfuerzo, es sumativo y relativo entre historias. La idea es que el propietario del producto y los expertos de negocio se reúnan para estimar el valor de las historias mediante *planning poker*, creando una base de conocimiento compartida para usuarios y gente del negocio. De la misma manera que se crea una base común en las reuniones del equipo.

Esta propuesta tiene dos efectos secundarios muy interesantes: fomentar el respeto y comprensión del equipo hacia el propietario del producto y que éste comprenda mejor por qué el equipo necesita a veces reestimar ciertas historias.

Conviene recordar que una historia, como requisito ágil, es sólo una hipótesis a validar que estimamos dará un beneficio al usuario. La valoramos primero desde el punto de vista de negocio, con la información de la que disponemos. Pero no sabremos si la hipótesis era acertada hasta ver la respuesta real del cliente. Por esto no se recomienda usar el valor de negocio entregado por un equipo en cada sprint o *release* como métrica. Hemos de entender el valor de negocio como un dato para priorizar la pila de producto.

## Estimación

Como hemos visto, es parte de la información esencial en una historia de usuario. Estimar el esfuerzo ayuda al propietario de producto a priorizar y al equipo a decidir qué historias de la pila caben en el sprint.

Un hecho a considerar es que la incertidumbre para estimar historias de usuario pequeñas es mucho menor que para historias grandes. Además, aunque el tamaño de las historias crece linealmente, la incertidumbre lo hace exponencialmente.

Una serie numérica que se presta bien a esta tarea es la de Fibonacci, compuesta por números que son la suma de los dos anteriores. La distancia entre los números de la serie refleja el hecho de que la incertidumbre inherente a la estimación crece con el tamaño de la historia. Así, las diferencias entre 1, 2 y 3 puntos de historia son probablemente mejor comprendidas que las diferencias entre 13, 21 y 42.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

También hay razones que derivan de la naturaleza humana para utilizar esta serie. Nos es fácil estimar determinando cuánto menor o mayor es algo respecto a otra cosa, por lo que en scrum y otros métodos ágiles se emplean técnicas de estimación relativa. Pero tenemos la tendencia de pensar en múltiplos de dos. Por ejemplo, comparamos una historia de usuario con una de 2 puntos, vemos que es mayor y automáticamente pensamos en 4, 8 o 16. La serie de Fibonacci nos obliga a romper con este hábito y buscar la estimación más adecuada.

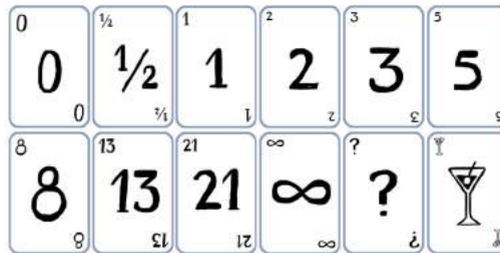


Ilustración 7: cartas planning poker con la serie de Fibonacci.

Las barajas que suelen usarse en estimaciones basadas en la serie de Fibonacci pueden contener variaciones. Algunas incluyen el 0 para historias que requieren un esfuerzo casi nulo y 1/2 para historias muy pequeñas. Algunas barajas sustituyen el 21 por un 20, ya que decir que una historia tiene un tamaño de 21 da una falsa sensación de precisión. A partir del 21 las barajas suelen pasar a incluir un símbolo de infinito ( $\infty$ ) y, si siguen, incluyen valores redondos como 40 y 100.

El objetivo del *planning poker* no es hacer predicciones que puedan confirmarse más adelante, sino ofrecer un proceso de estimación rápido, que dé al equipo valores útiles para estimar la pila de producto o planificar el sprint. Más importante aún es lograr que todos comprendan el elemento estimado: si los participantes estiman con valores muy dispares es que alguien sabe algo que otros no. Recordemos que en los equipos hay especialistas de diferentes áreas, optimistas y pesimistas, y eso hace que algunos vean soluciones que otros no ven. Estas discusiones servirán para alinear el conocimiento de todo el equipo.



Ilustración 8: cartas de estimación con tallas de camisa.

Para la estimación de elementos grandes como los epics y temas, la serie de Fibonacci no es adecuada: sus números darían una idea de precisión inexistente. Una solución es emplear una técnica basada en tallas de camisa. Las tallas sirven para intuir más que estimar, con un alto grado de incertidumbre, tal y como requieren elementos demasiado grandes y poco definidos. Esta técnica tiene la ventaja de ser puramente relativa, al no poder traducir sus valores a tiempo, jornadas u horas.

Con ambas técnicas podemos estimar la pila de producto al completo de forma coherente y ágil. Los elementos de la parte prioritaria de la pila están detallados, incluida la estimación, y los elementos más lejanos poco granulados, con una estimación más vaga y amplia.

El propietario del producto se basa en estos resultados para priorizar la pila de producto. Puede ver con más precisión lo más inmediato y disponer a la vez de detalle suficiente para los elementos lejanos y poco prioritarios.

## Prioridad

Aunque todas las historias de usuario puedan ser importantes, para focalizarnos de forma eficiente hay que destacar las que den más valor al sistema. El propietario del producto debe asignar un valor a cada historia que intervenga en el sistema de priorización, teniendo en cuenta las siguientes variables:

- Beneficios de implementar la funcionalidad.
- Pérdida o coste que derive de posponer su implementación.
- Riesgos de implementarla.
- Coherencia con los intereses del negocio.
- Valor diferencial con respecto a productos de la competencia.

Un aspecto a tener en cuenta es que la definición de valor puede ser distinta para cada cliente. Es recomendable utilizar una escala cualitativa, con un significado intrínseco, que aporte más información que si la prioridad es alta, media, o baja.

Es el caso de la técnica *MoSCoW*, en la que el usuario responsable de asignar la prioridad es consciente del efecto real que producirá su elección. Esta técnica fue definida por primera vez en el año 2004 por Dai Clegg de Oracle UK Consulting en el libro *Case Method Fast-Track: A RAD Approach*. Su finalidad es alcanzar un entendimiento común del proyecto entre cliente y equipo, en concreto sobre la importancia de cada historia de usuario. La clasificación es:

- *Must have*, es necesario: la funcionalidad debe estar implementada en la solución, si no ésta fallará o no podrá considerarse un éxito.
- *Should have*, es recomendable: se debería implementar, ya que es una funcionalidad de alta prioridad. La solución no fallará si no existe, pero debería haber causas justificadas.
- *Could have*, podría implementarse: es deseable, sería conveniente tener esta funcionalidad en la solución, pero dependerá de los tiempos y el presupuesto del proyecto.
- *Won't have*, no lo queremos por ahora: se trata de una funcionalidad de muy baja prioridad o descartada, pero que en el futuro puede ser relevante. En tal caso pasaría a alguno de los otros estados.

Es importante distinguir entre prioridad y valor. Puede que una historia de usuario no tenga ningún valor para el cliente o usuario, pero que esta sea imprescindible y de alta prioridad. Un caso sería la infraestructura necesaria para la implementación de un software, que no aporta valor al cliente en sí, pero sin la cual no se puede desarrollar ni ejecutar la solución.

Otra herramienta que ayuda al propietario del producto a priorizar la pila de producto de forma adecuada es el cálculo del *Return of Investment* (ROI, retorno de la inversión).

$$\text{ROI} = \text{Valor de negocio} / \text{Tamaño}$$

- **Valor de negocio:** valor relativo del elemento para el negocio o el cliente.
- **Tamaño:** estimación en puntos de historia del esfuerzo necesario según complejidad y tamaño.

El ROI aplicado de forma descendente resulta en una excelente guía para priorizar la pila: cuanto más alto el retorno, mayor es la prioridad del elemento.

$$\text{WSJF} = \text{Coste de demora} / \text{Tamaño}$$

Una técnica de priorización *lean* muy recomendable y completa es *WSJF* (*Weighted Shortest Job First*) de Don Reinersten, descrita en 2009 en su libro *The Principles of Product Development Flow: Second Generation Lean Product Development*. Está basada en la economía de flujo de desarrollo de productos, que calcula dividiendo el coste de demora por la duración.

Lo interesante del coste de demora es que lo tienen incluso las historias técnicas, que no tienen valor de negocio intrínseco. El coste de demora representa el valor que no obtenemos si no construimos determinada historia. En el caso de una historia técnica puede que dejemos de obtener el valor de negocio de otras historias de usuario ligadas a ella.

Dado que la duración es imposible de estimar, calcularemos el coste de demora utilizando el tamaño del elemento:

$$\text{Coste de demora} = \text{Valor de negocio} + \text{Críticidad en el tiempo} + \text{Reducción del riesgo y valor de oportunidad}$$

- **Valor de negocio.**
- **Críticidad en el tiempo:** esta medida se ocupa de la necesidad de la entrega del elemento en una escala de tiempo y viene asociada con la caída progresiva de valor. A más necesidad de entrega más alto será este factor.
- **Reducción del riesgo y valor de oportunidad:** esta medida da un valor relativo a la eliminación de uno o varios riesgos, o un valor por las oportunidades de negocio potenciales que aporte el elemento.

La técnica de priorización WSJF consiste en aplicar el siguiente proceso sobre la pila de producto:

- Se estiman las funcionalidades/historias de usuario/epics entre sí, de forma relativa.
- La escala de unidades es la serie de Fibonacci: 1, 2, 3, 5, 8, 13, 21.
- Se rellena la lista columna por columna.
- Se pone un 1 en el elemento con el valor más pequeño. Tiene que haber un 1 en cada columna.
- Se rellena el resto de elementos de la columna con estimación relativa respecto al 1.
- Hecha la última columna se calcula el WSJF y se ordena de mayor a menor.
- El elemento con mayor prioridad es el de WSJF más alto.

Funcionalidad / Historia de usuario / Epic	Valor de negocio	Criticidad en el tiempo	Reducción del riesgo y valor de oportunidad	Tamaño	WSJF
Como miembro del departamento de administración <b>quiero</b> una pantalla <b>para poder</b> gestionar el catálogo de productos de la empresa y que enlace con la foto correspondiente de Google+.	1	5	1	1	7
Como comercial <b>quiero</b> un catálogo con un sistema de pedidos online <b>para poder</b> formar a los clientes y hacer pedidos para aquellos que lo precisen.	5	13	8	8	3,25
Como miembro del departamento de administración <b>quiero</b> una pantalla <b>para poder</b> gestionar los datos de los clientes.	8	2	3	5	2,6
Como gerente <b>quiero</b> tener información mensual de productos vendidos y facturas emitidas <b>para</b> tener reportes con información fiable y a demanda.	3	1	1	3	1,67

Ilustración 9: ejemplo de priorización WSJF.

## Criterios de aceptación

Después de 50 años de historia de ingeniería de software se ha llegado a la conclusión de que los criterios de aceptación, que a veces se traducen en pruebas, son un excelente lenguaje para detallar requerimientos funcionales, y es por ello que toman una gran importancia en las historias de usuario.

Para medir la calidad de un criterio de aceptación se utiliza el método SMART, por el que se han de ser, en la medida de lo posible:

- *Specific*, específicos.
- *Measurable*, medibles.
- *Achievable*, alcanzables.
- *Relevant*, relevantes.
- *Time-boxed*, limitados en el tiempo.

Los criterios se suelen expresar en forma de *checklist* o como un flujo en cuanto se obtienen historias de usuario susceptibles de implementarse, y se refinan en la planificación de sprint. Ayudan al equipo de desarrollo a entender el funcionamiento del producto, de manera que estimarán mejor el tamaño de la historia subyacente. Durante la fase de desarrollo servirán además de guía para la tomar de decisiones.

Durante la revisión del sprint, el propietario de producto comprobará, a través de estos criterios de aceptación y la definición de hecho (DoR), si cada una de las historias de usuario se puede aceptar y dar por finalizada.

Los criterios de aceptación pueden escribirse en lenguaje natural, tal como el propietario del producto se expresa, y pueden adoptar diferentes formatos. Por ejemplo:

**Comprobar** [criterios].

**Demostrar** [comportamiento esperado].

**Verificar que cuando** [rol] **hace** [acción] **consigue** [resultado / comportamiento esperado].

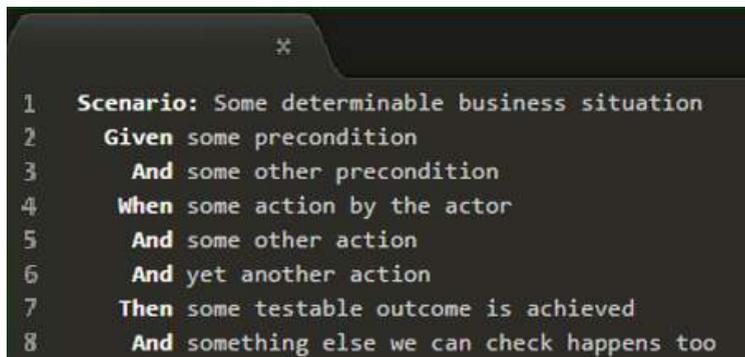
**Dado que** [contexto] **y adicionalmente** [contexto] **cuando** [evento] **entonces** [resultado / comportamiento esperado].

Una opción excelente es escribirlos con la técnica del comportamiento por escenarios propia de *BDD (Behavior Driven Development)* y con *gherkin*, un lenguaje creado para las descripciones de comportamiento de software. La sintaxis de *gherkin* es la siguiente:

**(Scenario) Escenario** [número de escenario] [título del escenario]:  
**(Given) Dado que** [contexto] **y adicionalmente** [contexto],  
**(When) Cuando** [evento],  
**(Then) Entonces** [resultado / comportamiento esperado].

Los elementos de los criterios de aceptación derivados de esta sintaxis son:

- **Número de escenario:** un identificador para el escenario asociado a la historia.
- **Título del escenario:** describe el escenario que define un comportamiento.
- **Contexto:** detalla las condiciones que desencadenan el escenario.
- **Evento:** representa la acción que el usuario ejecuta, en el contexto definido para el escenario.
- **Resultado / comportamiento esperado:** el comportamiento del sistema en esa situación.



```
1 Scenario: Some determinable business situation
2   Given some precondition
3     And some other precondition
4   When some action by the actor
5     And some other action
6     And yet another action
7   Then some testable outcome is achieved
8     And something else we can check happens too
```

Ilustración 10: ejemplo gherkin.

Ejemplo de criterios de aceptación escritos en gherkin:

### **Historia de usuario**

**Como** cliente  
**quiero** retirar dinero del cajero  
**para poder** evitar ir al banco a hacer cola.

### **Criterios de aceptación**

**Escenario 1:** cuenta tiene crédito.

**Dado que** la cuenta tiene crédito  
**Y** que la tarjeta es válida  
**Y** que el cajero tiene dinero disponible  
**Cuando** el cliente pide dinero  
**Entonces** la cuenta es debitada  
**Y** el dinero es entregado al cliente  
**Y** el cliente recupera su tarjeta

**Escenario 2:** la cuenta excede el límite negativo acordado con el banco.

**Dado que** la cuenta excede el límite negativo acordado con el banco  
**Y** que la tarjeta es válida  
**Cuando** el cliente pide dinero  
**Entonces** el cajero muestra un mensaje negando el pedido  
**Y** el dinero no es entregado al cliente  
**Y** el cliente recupera su tarjeta

## Requerimientos no funcionales

Los requerimientos no funcionales (*nonfunctional requirements* o *NFR* en inglés) representan cualidades generales y restricciones que afectan a aplicaciones y sistemas enteros. A diferencia de los epics e historias de usuario, cuyos criterios de aceptación se centran en comportamientos y funciones específicas. Sin embargo, una historia o epic no se puede considerar hecha si no cumple con los requisitos no funcionales asociados. Éstos garantizan que el sistema cumple con cualidades de gran importancia como la usabilidad, seguridad, y otras como:

Accesibilidad	Confiabilidad	Disponibilidad	Interoperabilidad	Operabilidad
Extensibilidad	Trazabilidad	Rendimiento	Recuperabilidad	Mantenibilidad
Compatibilidad	Conformidad	Escalabilidad	Usabilidad	Configurabilidad
Fiabilidad	Certificación	Eficiencia	Seguridad	Robustez

La implementación de nuevos NFR suele requerir de nuevos elementos en la pila de producto en forma de historias técnicas. Una vez incluidas en un sprint, éstas evolucionan en nuevas restricciones para el sistema.

También se suelen reflejar los NFR en los criterios de aceptación de las historias de usuario. Y si se trata de restricciones persistentes se añaden como elementos nuevos en la definición de hecho (*DoD*).



Ilustración 11: los requerimientos funcionales como restricciones.

Como todo requerimiento, los NFR se han de describir y cuantificar para entenderlos mejor. Vamos a ver un ejemplo siguiendo este patrón:

**Paso 1**

**Nombre (*name*):** [en forma de cualidad.subcualidad].

**Escala (*scale*):** [qué medir (unidad)].

**Métrica (*meter*):** [cómo medir (método)].

**Paso 2**

**Objetivo (*target*):** [nivel de éxito a alcanzar].

**Restricción (*constraint*):** [nivel de fallo a evitar].

**Línea base (*baseline*):** [nivel actual].

Partimos de la siguiente historia de usuario relacionada con una librería online:

**Como** cliente  
**quiero** poder explorar libros  
**para poder** escoger el que voy comprar.

Para la que hemos determinado el tiempo de respuesta como requerimiento no funcional, ya que si la página tarda en cargar más de 3 segundos el comprador buscará otras opciones:

**Paso 1**

**Nombre:** usabilidad.rendimiento.

**Escala:** milisegundos entre que el comprador clica «buscar» y se le presenta la página con el resultado.

**Métrica:** promedio de los tiempos de búsqueda de libros.

**Paso 2**

**Objetivo:** <200 milisegundos (lo que Google considera como tiempo de respuesta máximo).

**Restricción:** >3000 milisegundos (cuando el comprador empieza a sentir ansiedad).

**Línea base:** 1800 milisegundos.

## Definición de hecho

La definición de hecho o DoD es un artefacto de scrum que responde la pregunta ¿qué tiene que cumplir una historia de usuario para que esté hecha? Es una herramienta relacionada con la calidad del software y que tiene que ser puesta en común por el equipo y el propietario del producto. Consiste en una serie de criterios y acciones que agregan valor verificable y demostrable al producto.

La definición de hecho es un documento que explica qué es necesario para considerar una historia como hecha, y permite que todos los implicados compartan esta definición. Conviene diferenciarla de los criterios de aceptación, que son específicos para cada historia.

Sin una definición de hecho se puede producir una nube de trabajo no terminado que complica las planificaciones, provoca retrasos y riesgos ocultos para la release.

Como vamos a ver a continuación, la definición es aplicable a nivel de tarea, historia de usuario, sprint, y release. A nivel de sprint se suele incluir que la historia de usuario cumpla con los criterios de aceptación.

### Tarea:

- Implementada.
- El código escrito cumple estilos y directrices.
- Hechas las pruebas unitarias.
- Integrada en el repositorio.
- Integra con el resto en el *build*.
- Gestor de tareas actualizado.
- Satisfechas ciertas métricas: cobertura, análisis estático, etc.
- Aprobada por el *tester*.

### Sprint:

- Todas las historias planificadas están hechas.
- Hecha la revisión de sprint y obtenido feedback de los interesados.
- Hecha la retrospectiva e identificada una acción de mejora.
- Aprobado por el propietario del producto.

### **Historia de usuario:**

- Todas las tareas asociadas están hechas.
- Incidencias detectadas en fase de desarrollo resueltas.
- Código documentado y/o comentado.
- Documentación y otros requisitos del proyecto/producto hechos.
- Satisface los criterios de aceptación.
- Pruebas de integración hechas.
- Superadas las pruebas unitarias acumuladas y de aceptación de forma manual o automática.
- Sigue los estándares de ingeniería/arquitectura.
- Cumple con los requerimientos no funcionales.
- Está instalada en un entorno tipo *test*, *staging* o preproducción y ha pasado las pruebas de humo.
- Aprobada por el propietario del producto.

### **Release:**

- Todas las historias planificadas están hechas.
- Medios de distribución producidos.
- Comprobada la documentación de usuario.
- Cumple con los requerimientos no funcionales y estándares.
- Instrucciones de instalación/despliegue de la release hechas.
- Pruebas de seguridad, despliegue y regresión hechas.
- Incidencias críticas y de resolución necesaria para despliegue resueltas.
- Comunicación de release y formación a usuarios hechas.
- Aprobada por el propietario del producto.

# Historias de usuario de calidad

---

En 2003 Bill Wake desarrolló un método llamado *INVEST* para asegurar la calidad en la escritura de historias de usuario. El método sirve para comprobar la calidad de una historia de usuario revisando que cumpla una serie de características:

- *Independent*, independiente.
- *Negotiable*, negociable.
- *Valuable*, valiosa.
- *Estimable*, estimable.
- *Small*, pequeña.
- *Testable*, comprobable.

**Independiente:** poder planificar e implementar las historias de usuario en cualquier orden tiene muchas ventajas y facilita el trabajo posterior del equipo. Para que sea posible, cada historia debería ser independiente. Una forma de reducir las dependencias es combinar historias o dividir las de manera diferente.

**Negociable:** una historia de usuario es una descripción corta de una necesidad, sin detalles. Deben ser negociables ya que los pormenores serán acordados con el cliente o el usuario más adelante. Conviene mantener las historias en términos generales o vagos para no limitar estas conversaciones.

**Valiosa:** una historia de usuario tiene que aportar valor al cliente o al usuario. Una manera de conseguirlo es que sean ellos quienes la escriban.

**Estimable:** una buena historia de usuario debe poder estimarse con precisión suficiente para que el propietario del producto pueda priorizar y planificar su implementación. El equipo es quien suele realizar la estimación, y ésta se verá afectada por el tamaño de la historia, pues a mayor tamaño mayor incertidumbre, y por el conocimiento del equipo sobre la necesidad expresada. En caso de falta de conocimiento, serán necesarias más fases de conversación.

**Pequeña:** las historias de usuario deberían englobar como mucho unas pocas semanas de trabajo por persona. Incluso hay equipos que las restringen a días por persona. Una descripción corta ayuda a disminuir el tamaño de una historia de usuario, facilitando así su estimación.

**Comprobable:** la historia de usuario debería poder probarse en la fase de confirmación. Si el cliente o usuario no sabe cómo, significa que la funcionalidad no es del todo clara o que no es valiosa, y si el equipo no puede probarla es imposible que sepa si está terminada o no.

Thomas Wallet ha ideado una serie de preguntas que ayudan a evaluar los criterios INVEST:

### **Independiente**

- ¿Las dependencias internas están resueltas?
- ¿Las dependencias están visualizadas?
- ¿Las dependencias externas están acordadas?
- ¿Las dependencias externas están resueltas?

### **Negociable**

- ¿Se entiende la necesidad a resolver del usuario?
- ¿La descripción está abierta a futuros refinamientos?
- ¿El equipo puede definir el cómo?
- ¿Sólo el equipo formula el cómo?

### **Valiosa**

- ¿Se identificaron los usuarios clave?
- ¿El valor de negocio está definido?
- ¿La participación en la definición de valor de negocio es la adecuada?
- ¿El valor de negocio permite negociar?

### **Estimable**

- ¿El ítem se definió y registró claramente?
- ¿El equipo está listo para estimar?
- ¿Se estimó en una escala relativa de referencia?
- ¿Todo el mundo participó en la estimación?

### **Pequeña**

- ¿Tiene menos de <N> criterios de aceptación?
- ¿El tamaño estimando es menor a <N>?
- ¿Ya no tiene sentido subdividir más?
- ¿Ya no hay riesgos de aumento de tamaño?

### **Comprobable**

- ¿Los criterios de aceptación están definidos y claros?
- ¿Los criterios de aceptación están acordados con los usuarios?
- ¿Se generó la documentación necesaria?
- ¿Se acordó la provisión de recursos críticos de prueba?
- ¿La provisión de recursos críticos de prueba está resuelta?

Hay empresas que han diseñado sus propias tarjetas, como éstas de Braintrust. Aparte de dar un *look* profesional, son completas en información y coherentes en espacio para los campos:

USER STORY	
As a	
I want	
So that	
<b>I N V E S T</b>	Size:
<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Business Value:

ACCEPTANCE CRITERIA	
Meets team's definition of ready? <input type="checkbox"/>	
The braintrust consulting group www.braintrustgroup.com	

Ilustración 12: ejemplo de tarjetas diseñadas por Braintrust.

Consejos de buenas prácticas:

- Siempre escribir las historias con el qué, evitar el cómo.
- No escribir una descripción exhaustiva, sólo lo justo.
- Escribir el criterio de aceptación de forma suficientemente explícita.
- Estimar todas las historias. No hacerlo puede crear falsas expectativas.
- No fiar toda la información a las tarjetas. Se puede usar documentación externa, como una wiki.
- Nunca dar una historia por finalizada antes de cumplir con todos los requerimientos.

# División de historias de usuario

---

La reunión de refinamiento es parte del flujo continuo de toma de requisitos a través de historias de usuario. Es una reunión para mantener actualizada la pila del producto en la que el propietario, con ayuda del equipo, añade, reprioriza, elimina y divide elementos de la pila. Su objetivo es garantizar que las historias de usuario susceptibles de ser desarrolladas pronto tengan suficiente nivel de detalle y estimación para que el equipo pueda comprometerse.

La experiencia nos muestra que las historias de usuario más pequeñas mejoran el flujo y que las historias de usuario grandes implican una mayor incertidumbre funcional y una mayor dificultad para ser estimadas.

Si tenemos historias de usuario pequeñas, éstas serán más simples, comprensibles, y su desarrollo será más claro. Esto redundará en la aparición de menos incidencias por historia, de modo que el trabajo del equipo se podrá centrar en desarrollar nuevas historias en lugar de resolver contratiempos.

Cuando surgen imprevistos o bloqueos, el impacto sobre la entrega al final del sprint es directamente proporcional al tamaño de las historias. Historias pequeñas tienen alcances pequeños, por lo que si no se pueden terminar el alcance de lo no entregado será pequeño.

Imaginemos que tenemos la siguiente historia:

**Como** creador de contenido de *marketing*  
**quiero** gestionar las noticias de la web  
**para poder** anunciar al cliente nuestras novedades.

La historia podría dividirse en las siguientes sub-funcionalidades:

- Dar de alta una noticia.
- Editar una noticia.
- Eliminar una noticia.
- Publicar una noticia.
- Despublicar una noticia.

No tiene el mismo impacto dejar fuera la historia inicial que una o dos de sus sub-funcionalidades.

Además, disponer de historias de usuario pequeñas nos ofrece una ventaja competitiva en el mercado. La historia del ejemplo abarca 5 posibles historias menores. ¿Necesitamos implementar todas las sub-funcionalidades para sacar al mercado la funcionalidad de noticias? No. Seguramente con dar de alta las noticias, poder publicarlas y despublicarlas (por si cometemos algún error) será suficiente. De esta manera dejamos hueco en el sprint para poder sacar al mercado otros requisitos que aporten valor al usuario.

Dividir historias mejora su comprensión, hace las estimaciones más precisas y la priorización más fácil, reduce el número de incidencias, da mayor estabilidad en los tiempos de entrega, aumenta la productividad y la competitividad en el mercado.

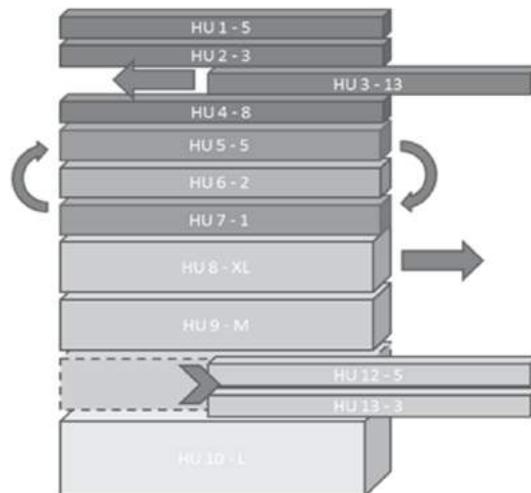


Ilustración 13: refinamiento de la pila del producto

Podemos dividir las historias de usuario de forma horizontal y vertical.

Dividir de forma horizontal significa hacerlo según el tipo de trabajo; por ejemplo, por tecnologías. Esto es típico de las metodologías de gestión tradicionales. Genera historias que no tienen valor de negocio de forma individual, sino en conjunto. Esto hace que no se puedan priorizar y además propicia que cada miembro del equipo se centre sólo en historias de su especialidad, lo cual tiende a generar cuellos de botella. La ingeniería de requisitos ágil evita este tipo de problemas con la multifuncionalidad de sus miembros: todos participan en mayor o menor medida en los diferentes tipos de tarea.

La división vertical es más útil. Una historia dividida en vertical genera historias que a su vez tienen valor de negocio. No está dividida a lo largo de capas

técnicas, sino de capas funcionales. A semejanza de los incrementos resultantes de un sprint, estas historias son como una porción de una tarta que incluye todas las capas técnicas necesarias.

Christiaan Verwijs describe 10 estrategias para dividir historias de usuario de forma vertical, para obtener historias funcionales y de menor tamaño:



Fuente: Christiaan Verwijs, Agilistic.

Ilustración 14: esquema con las 10 estrategias de división de historias de usuario.

A las 10 estrategias hay que añadir una para el caso en que la implementación de la historia sea compleja y difícil de entender y requiera una actividad exploratoria (*spike*) previa.

- Estrategia 0: extraer un spike.
- Estrategia 1: división por pasos de flujo de trabajo.
- Estrategia 2: división por reglas de negocio.
- Estrategia 3: división por *happy/unhappy flow*.
- Estrategia 4: división por opciones/plataformas de entrada.
- Estrategia 5: división por tipos de datos o parámetros.
- Estrategia 6: división por operaciones.
- Estrategia 7: división por casos/escenarios de test.
- Estrategia 8: división por roles.
- Estrategia 9: división por optimizar ahora o más tarde.
- Estrategia 10: división por compatibilidad de navegador.

En todas estas estrategias, que se explican a continuación, la división reduce la incertidumbre, nos permite centrarnos en las historias de mayor importancia y dejar las demás para desarrollos futuros.

## Estrategia 0: construir un spike

Un spike viene a ser una actividad de desarrollo en forma de historia técnica que se incluye en la pila de producto. Puede tratarse de reunir información para una toma de decisión posterior o del diseño de una solución.

### ¿Cuándo aplicar esta estrategia?

Construir un spike sirve para adquirir el conocimiento previo necesario para entender mejor la solución o la necesidad asociada a una historia de usuario, reduciendo incertidumbres respecto a su implementación. Esta técnica se materializa en pruebas de concepto o prototipos que permiten evaluar la viabilidad de la historia. Los resultados de estas actividades exploratorias permiten tomar decisiones adecuadas para refinar o definir el alcance de una funcionalidad.

Los spikes se escriben en forma de historia técnica. Supongamos la siguiente historia de usuario:

**Como** ciclista

**quiero** pagar mi compra con tarjeta de crédito

**para poder** comprar accesorios para mi bicicleta.

El spike correspondiente podría ser:

Explorar la pasarela de pago de tarjetas de crédito de nuestro banco para conocer la viabilidad de su inclusión en nuestro producto.

Los criterios de aceptación hacen referencia a las cuestiones que necesitan ser respondidas. Aparte de generar conocimiento y despejar dudas, los spikes facilitan a los equipos dar estimaciones sólidas a historias de usuario posteriores.

## Estrategia 1: división por pasos de flujo de trabajo

Al refinar una historia de usuario con un flujo de trabajo asociado, solemos descubrir que éste la hace más compleja de lo esperado. Cuando esto sucede, esta estrategia identifica el MPV (Mínimo Producto Viable) y las historias incrementales necesarias para obtener la historia original.

A veces el mayor valor está en los pasos inicial y final del flujo; los intermedios agregan valor, pero pueden posponerse. El MPV se compondría de esos primer y último paso, dividiendo el resto en pasos o grupos de pasos que puedan agregarse de forma independiente a posteriori.

En otras historias de usuario todo el flujo de trabajo es necesario, y el MPV ha de ser un corte fino de todo el flujo. La estrategia de división se basaría en agregar complejidad a través de historias que añadan capas al flujo de trabajo.

También encontraremos historias con varias alternativas o caminos posibles. Podemos enfocar el MPV empezando por un único camino, el de máximo valor, y basar la estrategia de división en los diferentes caminos posibles.

La división por flujos de trabajo aporta el gran beneficio de mejorar el entendimiento de la funcionalidad, lo que a su vez facilita definir el MPV.

### ¿Cuándo aplicar esta estrategia?

Cuando la historia de usuario involucra un flujo de trabajo de algún tipo. Veamos un ejemplo con esta historia de una tienda online:

**Como** comprador  
**quiero** pagar los artículos de mi carrito de la compra  
**para poder** recibirlos en mi casa.

División considerando los pasos del flujo de trabajo:

#### MPV

**Como** comprador  
**quiero** confirmar y revisar mi carrito de la compra  
**para poder** confirmar antes de pagar.

**Como** comprador  
**quiero** utilizar PayPal  
**para poder** informar de forma automática mis datos personales y la dirección de envío y efectuar el pago.

## **Incremento 1**

**Como** comprador

**quiero** identificarme a través de mi cuenta de Facebook

**para poder** introducir de forma automática mis datos y la dirección de envío.

**Como** comprador

**quiero** utilizar una tarjeta de crédito

**para poder** efectuar el pago.

## **Incremento 2**

**Como** comprador

**quiero** utilizar una transferencia bancaria

**para poder** efectuar el pago.

## Estrategia 2: división por reglas de negocio

Esta estrategia de división propone incorporar reglas de negocio especiales paulatinamente, como ampliaciones incrementales de la funcionalidad inicial. Estas reglas pueden ser estándares de la industria, regulaciones legales, o cualquier tipo de restricciones complejas.

Así el propietario del producto puede obtener historias de usuario iniciales que cubran las reglas imprescindibles, implementando las demás más adelante o de forma simplificada. Una regla simplificada como un mensaje en una tienda online que diga «no enviamos pedidos fuera de UE» puede ser suficiente para empezar.

### ¿Cuándo aplicar esta estrategia?

Cuando la historia de usuario involucre un conjunto de reglas de negocio, implícitas o explícitas, y algunas se puedan incorporar paulatinamente a posteriori.

Ejemplo con una historia de usuario de una tienda online:

**Como** comprador  
**quiero** pagar los artículos de mi carrito de la compra  
**para poder** recibirlos en mi casa.

Su división genera otras historias igual de complejas usando diferentes reglas de negocio:

**Como** propietario de la tienda  
**quiero** rechazar pedidos menores a 10 euros  
**para poder** evitar compras no rentables.

**Como** propietario de la tienda  
**quiero** rechazar clientes de fuera de UE con pedidos inferiores a 100 euros  
**para poder** evitar gastos de envío que hacen la compra no rentable.

**Como** propietario de la tienda  
**quiero** reservar stock de productos pedidos durante 48 horas  
**para poder** ofrecer a los clientes un stock real.

**Como** propietario de la tienda  
**quiero** cancelar automáticamente pedidos no pagados en 48 horas  
**para poder** vender esos productos a otros clientes.

Las reglas de negocio a menudo son implícitas, por lo que para descubrirlas puede ser útil dividir las primero por casos/escenarios de test.

## Estrategia 3: división por happy/unhappy flow

Las buenas historias de usuario y sus criterios de aceptación implican flujos de éxito (happy flows) y de fallo (unhappy flows). El flujo de éxito o feliz describe el escenario ideal, con pocas o ninguna opción; el comportamiento de la funcionalidad cuando todo va bien. Los flujos de fallo o infelices tratan cualquier flujo alternativo: desviaciones, excepciones, o problemas que puedan darse.

Identificar flujos inherentes a una funcionalidad permite entenderla en profundidad y facilita la priorización al propietario del producto. Por ejemplo: puede que bloquear a un usuario después de tres intentos fallidos (un unhappy flow) no sea importante mientras tengamos pocos usuarios.

### ¿Cuándo aplicar esta estrategia?

Cuando la funcionalidad implica un flujo ideal en el que todo sale bien y uno o más flujos alternativos.

Ejemplo de una historia de usuario de *login* en un banco:

**Como** usuario  
**quiero** acceder a mi *home-banking*  
**para poder** acceder a mis movimiento bancarios de forma segura.

División considerando varios flujos posibles:

#### Happy flow

**Como** usuario  
**quiero** loguearme con mi cuenta de de correo electrónico  
**para poder** acceder a mi home-banking.

#### Unhappy flow

**Como** usuario  
**quiero** resetear mi contraseña cuando mi login falla  
**para poder** intentar acceder de nuevo.

**Como** usuario  
**quiero** tener la opción de registrar una cuenta de correo electrónico nueva si no puedo acceder con la actual  
**para poder** acceder a mi home-banking.

**Como** responsable de seguridad del banco  
**quiero** bloquear usuarios que se loguen con credenciales inválidas tres veces seguidas  
**para poder** proteger el sistema de *hackers*.

## Estrategia 4: división por opciones/plataformas de entrada

Esta estrategia propone dividir las historias de usuario por interfaces o medios por los cuales un usuario puede interactuar con el sistema. Por ejemplo diferentes dispositivos, como ordenadores de sobremesa, portátiles, teléfonos móviles... y sus sistemas operativos, que pueden requerir personalizaciones.

Esta división facilita al propietario del producto priorizar las opciones de entrada o plataformas más importantes en cada momento.

A veces la interfaz de usuario presenta más complejidad que la funcionalidad en sí. En ese caso la estrategia de división pasa por una historia de usuario con una interfaz lo más simple posible que luego se enriquece con historias incrementales.

### ¿Cuándo aplicar esta estrategia?

Cuando la historia tenga que soportar varias formas de entrada y/o plataformas y nos podamos preguntar ¿hay una versión más simple que podamos construir antes?

Ejemplo de una historia de usuario de una agencia de viajes online:

**Como** viajero

**quiero** comprar vuelos entre dos destinos

**para poder** disfrutar de mis vacaciones conociendo mundo.

Historias resultantes de la división por esta estrategia:

#### Por dispositivo

**Como** viajero

**quiero** comprar vuelos entre dos destinos a través de una *app* móvil

**para poder** disfrutar de mis vacaciones conociendo mundo.

**Como** viajero

**quiero** comprar vuelos entre dos destinos a través de un ordenador

**para poder** disfrutar de mis vacaciones conociendo mundo.

#### Por complejidad

**Como** viajero

**quiero** poder elegir vuelos tecleando la fecha que me interesa

**para poder** encontrar el billete.

**Como** viajero

**quiero** poder elegir vuelos seleccionando la fecha

a través de una elegante interfaz de usuario tipo calendario

**para poder** encontrar el billete.

Ahora el viajero tiene que imprimir del navegador, más adelante añadiremos:

**Como** viajero

**quiero** imprimir el vuelo comprado

**para poder** embarcar.

## Estrategia 5: división por tipos de datos o parámetros

Esta estrategia divide historias de usuario en elementos más pequeños basándose en la complejidad de sus datos o los parámetros. Lo que plantea es incorporarlos de manera incremental a través de historias más pequeñas, permitiendo a su vez entenderlos mejor.

Imaginemos una funcionalidad de búsqueda de libros que dividimos en historias por tipos de búsqueda; esto nos permite estimar y priorizar la funcionalidad con mayor precisión. Quizás las búsquedas relevantes sean por título y por ISBN, si se acerca septiembre y es época de comprar libros de texto; otros tipos de búsqueda pueden implementarse por ahora de manera simplificada o dejarlas para más adelante.

También se puede dividir según los tipos de datos que devuelve la funcionalidad. Aunque la idea sea presentarlos visualmente con gráficos, por ahora podemos decidir mostrarlos en un formato tabular y crear los gráficos a mano con Excel.

Otra opción es emplear DER (diagramas de entidad-relación), que sugieren varios grupos diferentes de datos sobre lo que basarnos. Si en un DER tenemos entidades como libros, categorías y autores, podemos esperar historias de usuario relacionadas.

### ¿Cuándo aplicar esta estrategia?

Cuando la historia es divisible por los tipos de datos que retorna o por los parámetros que se supone debe manejar.

Ejemplo de una historia de usuario para una web de reservas en restaurantes:

**Como** comensal  
**quiero** encontrar un restaurante  
**para poder** llevar a mi pareja a una cena romántica.

La división genera tantas historias como criterios de búsqueda con valor:

**Como** comensal  
**quiero** buscar restaurantes por tipo de cocina  
**para poder** encontrar rápidamente un restaurante que se adapte a mis deseos.

**Como** comensal  
**quiero** buscar restaurantes por un rango de precios  
**para poder** obtener resultados de búsqueda más relevantes.

**Como** comensal  
**quiero** buscar restaurantes por un grupo de ingredientes principales  
**para poder** obtener resultados de búsqueda más relevantes.

## Estrategia 6: división por operaciones

Esta estrategia ofrece una forma natural de dividir las operaciones de una historia de usuario en implementaciones independientes.

Nos referimos a historias de usuario sobre entidades como artículos, pedidos, tarifas, usuarios, etc., todas ellas asociadas a operaciones de alta, lectura, modificación y baja (*CRUD* por sus siglas en inglés). El propietario del producto puede llevar al equipo las funcionalidades más prioritarias o complejas, como el alta de un artículo, dejando otras menos frecuentes para más adelante, como eliminar o actualizar un artículo, que puede efectuarse directamente sobre la base de datos mientras tanto.

### ¿Cuándo aplicar esta estrategia?

Cuando una historia de usuario involucre operaciones. La palabra “gestionar” suele ser un indicador de que cubre múltiples operaciones, como las típicas *CRUD* u otras, de configuración por ejemplo.

Veamos el caso de una historia de usuario para un carrito de la compra:

**Como** comprador online de libros  
**quiero** gestionar los libros que voy eligiendo en un carrito de la compra  
**para poder** realizar la compra con mi decisión final.

División considerando las operaciones *CRUD* individuales:

**Como** comprador online de libros  
**quiero** añadir libros nuevos a mi carrito de la compra  
**para poder** ver los libros elegidos  
y una vez agotado mi presupuesto comprar esos libros.

**Como** comprador online de libros  
**quiero** poder consultar mi carrito de la compra  
**para poder** saber lo que he elegido hasta el momento.

**Como** comprador online de libros  
**quiero** poder actualizar las unidades de cada libro  
**para poder** comprar varios ejemplares de un mismo libro.

**Como** comprador online de libros  
**quiero** poder eliminar libros de mi carrito de la compra  
**para poder** comprar los libros más importantes para mí  
y que la compra se ajuste a mi presupuesto.

Desde la perspectiva del valor de negocio, añadir libros al carrito de la compra es la funcionalidad más prioritaria.

## Estrategia 7: división por casos/escenarios de test

Esta división se hace en base a todos los casos y escenarios de prueba derivados de los criterios de aceptación de la historia. Es útil sobre todo cuando resulta difícil dividir la historia en base sólo a su funcionalidad. Los escenarios de prueba relevantes pueden traducirse fácilmente a historias de usuario y pueden implementarse de forma incremental.

Los escenarios de prueba en entornos automatizados suelen estar escritos en gherkin, un lenguaje creado a tal efecto para que las pruebas puedan ser entendidas tanto por los equipos de desarrollo y los técnicos como por el propietario del producto, interesados del negocio, analistas e incluso el cliente.

Si un escenario de prueba no es muy común o no presenta un riesgo lo suficientemente alto, el propietario del producto puede decidir omitir la funcionalidad de momento para centrarse en otras que aporten más valor. También puede decidir simplificar algún escenario de prueba para cubrir los problemas más urgentes.

### ¿Cuándo aplicar esta estrategia?

Cuando podamos responder a la pregunta ¿qué escenarios tienen que ser comprobados para saber que la funcionalidad está bien implementada?

Dividiremos así una historia de usuario de un banco:

**Como** cliente

**quiero** retirar dinero del cajero automático

**para poder** evitar ir al banco a hacer una cola.

#### Escenario 1

Cuenta tiene crédito

**dado que** la cuenta tiene crédito

y que la tarjeta es válida

y que el cajero tiene dinero disponible

**cuando** el cliente pide dinero

**entonces** la cuenta es debitada

y el dinero es entregado al cliente

y el cliente recupera su tarjeta.

## Escenario 2

La cuenta excede el límite negativo acordado con el banco  
**dado que** la cuenta excede el límite negativo acordado con el banco  
y que la tarjeta es válida  
**cuando** el cliente pide dinero  
**entonces** el cajero muestra un mensaje negando el pedido  
y el dinero no es entregado al cliente  
y el cliente recupera su tarjeta.

## Escenario 3

El cliente ha iniciado la operativa en su móvil  
**dado que** el cliente ha introducido el importe a retirar en la app móvil  
**cuando** coloca el móvil con el código QR frente al escáner del cajero  
**entonces** el cajero entrega el dinero directamente al cliente.

La división por escenarios de prueba de esta historia genera nuevas historias:

## Escenario 1

**Como** responsable del banco  
**quiero** comprobar que la cuenta del cliente tiene crédito  
**para poder** no entregar dinero por error.

**Como** responsable del banco  
**quiero** comprobar que la tarjeta del cliente es válida  
**para poder** no entregar dinero por error.

**Como** responsable del banco  
**quiero** comprobar que el cajero tiene dinero disponible  
**para poder** asegurar la entrega del dinero.

## Escenario 2

**Como** responsable del banco  
**quiero** que el cajero deniegue la entrega del dinero a través de un mensaje  
**para poder** evitar que la cuenta exceda el límite negativo acordado.

## Escenario 3

**Como** cliente  
**quiero** iniciar la operativa de retirar dinero del cajero automático en la app  
**para poder** evitar hacer una cola y obtener el dinero directamente.

Como podemos ver, esta estrategia ayuda de manera implícita a aplicar otras estrategias. Los escenarios 1 y 2 nos llevan a usar la estrategia de división por reglas de negocio, mientras que el escenario 3 se presta a dividir por opciones/plataformas de entrada.

## Estrategia 8: división por roles

Una de las primeras consideraciones al escribir una historia de usuario es el rol para quien la escribimos. Los roles se suelen definir a través de user personas, y es probable que la historia beneficie a cada uno de maneras distintas. La división por roles se puede aplicar cuando hay más de un rol receptor beneficiado.

Imaginemos una web de reserva de restaurantes que va dirigida a dos públicos o roles: clientes comunes y clientes VIP. Es probable que el proceso de reserva sea distinto. Un propietario del producto encuentra con esta estrategia una manera de dividir, priorizar y entregar valor antes.

### ¿Cuándo aplicar esta estrategia?

Cuando la historia involucra a varios roles o grupos que realizan ciertas partes de la funcionalidad.

Ejemplo de una historia de usuario de un periódico online:

**Como** área responsable de noticias  
**quiero** publicar nuevos artículos en nuestra página web  
**para poder** atraer a nuestros clientes y tengan un aliciente para regresar.

División considerando los distintos roles involucrados:

**Como** cliente  
**quiero** leer nuevos artículos  
**para poder** estar informado de eventos importantes.

**Como** periodista  
**quiero** escribir nuevos artículos  
**para poder** atraer a nuestros clientes.

**Como** diseñador  
**quiero** aplicar los estilos corporativos a los artículos nuevos  
**para poder** integrarlos en el look&feel de nuestra página web.

**Como** editor  
**quiero** revisar artículos nuevos antes de publicarlos en nuestra página web  
**para poder** prevenir errores.

**Como** responsable audiovisuales  
**quiero** incluir fotografías y vídeos en los artículos nuevos  
**para poder** registrar las escenas de la noticia.

## Estrategia 9: división por optimizar ahora o más tarde

En algunas historias la complejidad no está en su implementación funcional, sino en los requerimientos no funcionales como el rendimiento. Esta estrategia de división trata de implementar la funcionalidad en diversos grados de perfección y optimización, que podemos resumir en «hazlo funcionar» y más adelante «haz que sea rápido».

Se puede aprender mucho de una solución inicial lenta, e incluso puede tener algo de valor para el usuario. El propietario del producto puede priorizar las historias resultantes en base a lo que la mayoría de clientes necesite antes.

Es importante resaltar que esta estrategia se refiere a la optimización funcional, no a una optimización de código que pueda servir de excusa para incurrir en deuda técnica.

### ¿Cuándo aplicar esta estrategia?

Cuando la funcionalidad pueda implementarse a diferentes niveles de perfeccionamiento y optimización, hacerla funcionar primero y después mejorarla para cumplir con los requerimientos no funcionales.

Ejemplo de una historia de búsqueda de hoteles:

**Como** visitante  
**quiero** buscar hoteles en un vecindario  
**para poder** restringir mi búsqueda a la zona donde esté ubicado.

División considerando los diferentes grados de optimización:

**Como** visitante  
**quiero** buscar hoteles considerando un radio a partir de una dirección base  
**para poder** acotar mis búsquedas.

**Como** visitante  
**quiero** introducir el código postal y buscar en el vecindario  
**para poder** no tener que completar la dirección base.

**Como** visitante  
**quiero** usar la ubicación de mi *GPS* y buscar en el vecindario  
**para poder** no tener que completar la dirección base.

**Como** visitante  
**quiero** obtener los hoteles más buscados en el vecindario de forma inmediata mientras que otros hoteles se cargan en *background*  
**para poder** obtener resultados más rápidamente.

## Estrategia 10: división por compatibilidad de navegador

Esta estrategia propone dividir las historias de usuario por navegadores de Internet que puedan interactuar con la página. Los navegadores más modernos, como Chrome, Firefox o Edge, se mantienen para ser compatibles con los estándares, mientras que los más antiguos suelen necesitar personalizaciones para que todo funcione.

Esta estrategia facilita la priorización al propietario del producto y ayuda al equipo a centrarse en el navegador de más valor en cada momento. Puede que un alto porcentaje de usuarios de una tienda online acceda desde Chrome, pero que los usuarios de una intranet de una gran multinacional cargada de *legacy* accedan mediante Internet Explorer. Focalizándonos en el navegador más utilizado podemos entregar valor antes.

### ¿Cuándo aplicar esta estrategia?

Cuando tengamos historias para aplicaciones web que deban funcionar en distintos navegadores.

Ejemplo de una web de venta online de artículos de belleza:

**Como** comprador  
**quiero** ver los detalles del producto  
**para poder** decidir si es lo que quiero comprar.

División considerando los diferentes navegadores donde se debe poder ver la funcionalidad:

**Como** comprador  
**quiero** ver los detalles del producto en Edge  
**para poder** decidir si es lo que quiero comprar.

**Como** comprador con un ordenador *vintage*  
**quiero** ver los detalles del producto en Internet Explorer 7  
**para poder** decidir si es lo que quiero comprar.

**Como** comprador con un móvil iPhone  
**quiero** ver los detalles del producto en Safari  
**para poder** decidir si es lo que quiero comprar.

# Otras formas de toma de requisitos

---

## Casos de uso

Los casos de uso son otra técnica más que a veces se incorpora en los procesos ágiles, pero no son equivalente a las historias de usuario. Conviene diferenciar los unos de las otras para evitar confusiones.

Un caso de uso captura la funcionalidad deseada desde la perspectiva de los usuarios (actores) y cómo interactúan con el sistema. Se escriben usando *UML* (*unified modeling language*) en los diagramas de casos de uso. UML es un lenguaje de modelado para describir, de forma sencilla, un sistema desde sus perspectivas estática y dinámica. A diferencia de las historias de usuario, que se escriben en un lenguaje coloquial para recordar la conversación con el cliente, el UML no pretende sonar natural.

Como comenta Alistair Cockburn en 2001 en su libro *Agile Software Development*, las historias de usuario están más cerca de la captura de requisitos, la fase que sirve para extraer las necesidades del usuario. Los casos de uso se centran en cambio en la especificación de requisitos. Podríamos decir que una historia de usuario dice *qué* quiere el cliente o usuario, y un caso de uso entra en *cómo* lo quiere.

También difieren en los criterios de aceptación. Los casos de uso requieren matrices de seguimiento de requisitos con porcentajes que marquen su avance; el criterio de aceptación de una historia de usuario no usa porcentajes sino que es binario, o vale o no vale.

Las historias de usuario son documentos vivos. El análisis funcional y técnico se hace poco antes del desarrollo, en la reunión de inicio de sprint en caso de scrum, y el desglose en tareas lo hace el equipo. El nivel de detalle y previsión supera en mucho al que pueda hacer un único arquitecto o analista funcional en los casos de uso.

Cuando un proyecto comienza a seguir un método ágil se deberían de olvidar los casos de uso y el equipo debería de centrarse en la realización de historias de usuario.

## **Requisitos funcionales**

Las historias de usuario se confunden también a veces con los requisitos funcionales. Se entiende que las primeras son un artefacto de los métodos ágiles, y que los otros cumplen el mismo papel en metodologías tradicionales. Sin embargo, hay diferencias significativas.

Hemos de ser conscientes de que, aunque las historias de usuario describen funcionalidades que serán útiles para el cliente o usuario, y que se suelen escribir en tarjetas o pósts, son mucho más que eso. Implican una conversación posterior en la que el equipo detalla, junto con el usuario o cliente, la funcionalidad a desarrollar.

Igual que en los requisitos funcionales, las historias de usuario dicen el qué pero no el cómo se desarrollará la funcionalidad. Las historias no deben tener el nivel de detalle que tiene la especificación de un requisito funcional.

# Historias técnicas

---

La pila de producto contiene todos los elementos que se van a construir: historias de usuario, epics y temas. Pero para poder realizar planificaciones tanto de sprint como de release, debe de contener además todo el trabajo pendiente, incluyendo necesidades de carácter técnico, que no aportan valor de negocio pero son consumidas por los elementos funcionales.

Por poner ejemplos, las historias técnicas incluyen acciones como preparar un *webserver*, implementar un conjunto de tablas en una base de datos que va a ser consumida por varias funcionalidades, elementos de seguridad, escalabilidad, rendimiento, etc. Otros tipos de historias técnicas se centran en resolver deuda técnica y en refactorizaciones, otros en historias de exploración como un análisis técnico o uno funcional que sirve para despejar incertidumbre sobre alguna historia de usuario.

Es interesante que negocio y propietario del producto tengan contacto con los aspectos técnicos, para forjar un lenguaje común y que todos los implicados sean conscientes de su importancia.

Las historias técnicas se escriben directamente en texto técnico claro y preciso, sin un patrón como ocurre con las historias de usuario. También tienen criterios de aceptación asociados que se comprueban en la revisión de sprint por la audiencia técnica correspondiente.

Aunque el propietario del producto sea el responsable de la pila y por tanto de todos sus elementos, en el caso de las historias técnicas los verdaderos propietarios son perfiles de carácter técnico como el equipo o un arquitecto. Son responsables no sólo de definir la historia, sino que también de responder dudas asociadas en la planificación y en la entrega.

Se pueden identificar diferentes tipos de historias técnicas:

- **Arquitectura:** construyen elementos como las APIs, que crean la estructura, funcionamiento e interacción entre distintas las partes del software.  
Ejemplo: «Implementar un sistema de login seguro.»
- **Infraestructura de producto:** historias que son consumidas directamente por historias de usuario. Esto podría incluir infraestructura nueva y/o modificada, y oportunidades de refactorización originada por alguna necesidad funcional.  
Ejemplo: «Preparar los servidores de base de datos y web.»

- **Infraestructura del equipo:** historias que respaldan al equipo en su capacidad para entregar software. Suelen ser historias para herramientas y marcos de pruebas, métricas, diseño, planificación... También pueden implicar que el equipo desarrolle o compre e instale algo.  
Ejemplo: «Preparar un sistema de integración continua.»
- **Refactorización:** historias que representan candidatos a refactorizar, como por ejemplo deuda técnica. Pero no sólo el código necesita de refactorización, también puede incluir diseños, automatización, herramientas y cualquier documentación de proceso.  
Ejemplo: «Homogeneizar el código de la función de cálculo de préstamos.»
- **Spikes:** historias de exploración limitadas en el tiempo que dan respuesta a una cuestión, o reúnen información para una toma de decisión posterior o el diseño de una solución.  
Ejemplo: «Evaluar Oracle versus SQL/Server.»
  - **Spike técnico:** si no estamos seguros de cómo desarrollar algo desde un punto de vista técnico creamos este tipo de spike, una breve actividad que se centra en encontrar un enfoque de desarrollo, en determinar la factibilidad y el impacto de las estrategias de diseño.
  - **Spike funcional:** sirve a los equipos para descubrir los detalles de las funcionalidades y los diseños a través de la creación de prototipos y llegar a entender exactamente lo necesita el cliente.

# User Story Mapping

---

Jeff Patton describe esta técnica en su libro *User Story Mapping: Discover the Whole Story, Build the Right Product* (2014). Resulta muy útil para construir una pila de producto que vaya más allá de una lista unidimensional de historias de usuario y epics.

Se trata de una técnica colaborativa de construcción de la pila de producto. Implica a todos los involucrados en la definición, uso y construcción del producto: interesados, clientes o usuarios finales, el propietario del producto y el equipo. También hay un moderador, que en la mayoría de los casos es el *scrum master* o el *agile coach*. El objetivo es que se definan, descubran, prioricen y estimen de forma conjunta las historias de usuarios y epics que se prevén como parte del producto a construir.

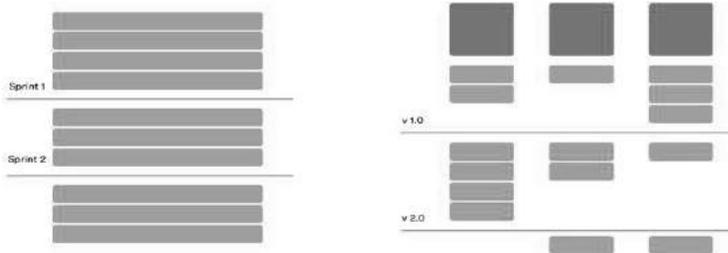


Ilustración 15: pila de producto plana y pila de producto construida con la técnica user story mapping.

Esta técnica nos permite obtener:

- **Visión compartida:** todos los implicados participan en la elaboración en la elaboración del user story map, construyendo una visión compartida del producto a obtener. Esto facilita mantener el foco en la solución a lo largo de su construcción.
- **Alineación:** los miembros del equipo y negocio (propietario del producto e interesados o clientes) podrán estar alineados sobre lo que se va a construir sabiendo el porqué o las razones subyacentes.
- **Mejor entendimiento de los deseos y necesidades de los clientes:** a través del modelado de los clientes y de lo que van a hacer con el producto.

- **Mejor entendimiento de los problemas que enfrentan los clientes:** participar en esta dinámica, modelando lo que el cliente hará con el producto y cómo, facilita empatizar, reflexionar y plantear mejores soluciones.
- **Un mapa de historias de usuario ordenado por versión:** al modelar un *backbone* del flujo del cliente a través del producto, podemos definir el conjunto de historias que conformarán el mínimo producto viable a sacar al mercado. Un mínimo que permita a los usuarios llevar a cabo todo el flujo de una manera básica (al menos una funcionalidad por cada actividad obligatoria en el flujo). También sirve para definir una idea inicial de las siguientes versiones que se prevén, que puede cambiar en función de los resultados obtenidos cuando salga a producción el MPV y cada nueva versión.

Para emplear user story mapping hay que definir los siguientes elementos:

1. **Backbone del user story map:** el backbone o espina dorsal del user story map captura las actividades de alto nivel que realizará un usuario cuando utilice el producto. Por ejemplo, el backbone de un proceso de compra de un libro en formato digital en una web sería:



Ilustración 16: ejemplo de backbone de un user story mapping.

2. **Historias de usuario asociadas con cada actividad del proceso ordenadas por valor:** se definen las historias de usuario que permitan realizar cada actividad. Luego se ordenan de arriba abajo, colocando las más valiosas o prioritarias más arriba. Continuando con el ejemplo, un conjunto de historias de usuario asociadas a cada actividad y priorizadas por valor sería:



Ilustración 17: backbone e historias de usuario de un user story mapping.

3. **Mínimo producto viable (v 1.0) y siguientes versiones previstas:** se refleja en el user story map cuáles son las historias de usuario que compondrán el MPV (v 1.0) y las siguientes versiones.

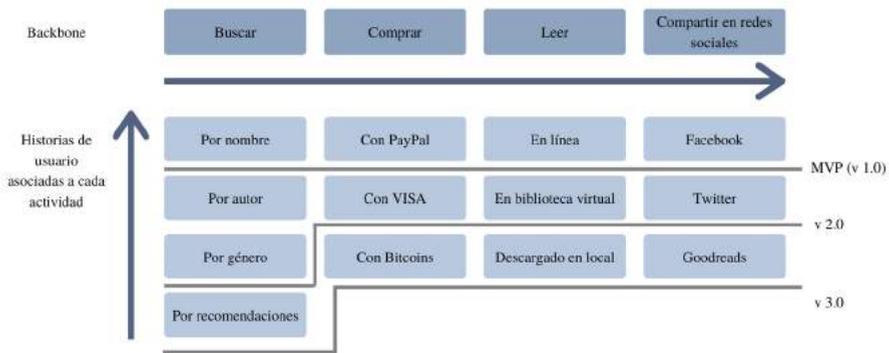


Ilustración 18: backbone, historias de usuario y versiones de un user story mapping.

Una vez que se ha llegado hasta aquí, el equipo puede hacer una estimación inicial del esfuerzo necesario para realizar cada historia de usuario.

Así el propietario del producto puede conocer el esfuerzo aproximado para desarrollar el MPV y cada una de las versiones definidas en el user story map. Si conoce la velocidad del equipo, es decir, cuántos puntos de historia puede hacer el equipo por sprint, y la duración de los sprints, puede hacer un gráfico de producto y obtener una estimación inicial de cuándo es probable que se entregue cada versión.

Esta estimación inicial se irá ajustando conforme se hagan entregas de software funcionando, que aporten feedback sobre el producto y sus funcionalidades.

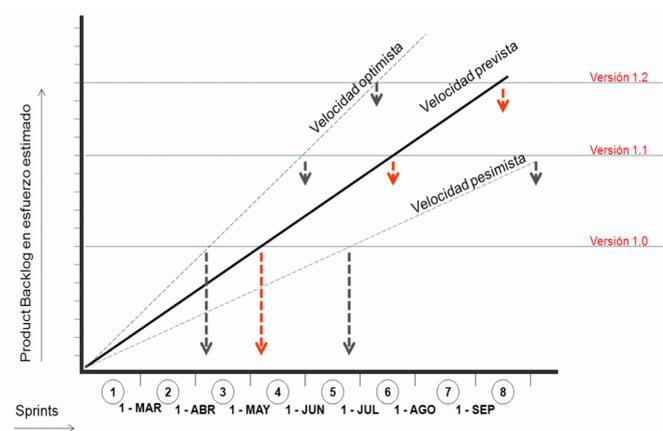


Ilustración 19: previsión de lanzamiento de versiones sobre gráfico de producto.

# Apéndices

---

## Historias de usuario en otros ámbitos

La agilidad se está incorporando a nuevas áreas de la compañía, y una de las más propicias para ello es marketing. Las microcampañas con feedback rápido en forma de sprints hacen que scrum sea una forma idónea de trabajo. Jim Ewel, experto en Agile Marketing, compara las historias de usuario de marketing con las de TI poniendo el foco en sus diferencias:

Historias de usuario de TI	Historias de usuario de marketing
Bajo nivel	Alto nivel
Número elevado de estas en la pila	Relativamente pocas en la pila
Hemisferio izquierdo del cerebro	Hemisferio derecho del cerebro
Focalizadas en funcionalidades	Focalizadas en resultados

Las historias de usuario de TI son detalladas y de una granularidad muy fina, mientras que las de marketing son más amplias y su pila de producto suele ser corta.

Pero lo más interesante es en lo que se focalizan: las historias de TI, al hablar de funcionalidades, están orientadas a la parte racional de las personas; las de marketing a la parte emocional. Por poner un caso, cuando la gente recuerda ciertos anuncios suelen hablar de cómo les han hecho sentir, de cómo cambió su día o cómo les hizo reír.

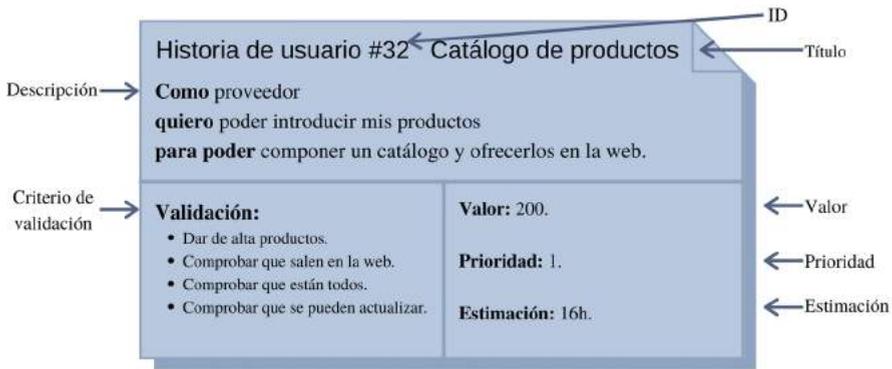
La solución a la que apunta una historia de marketing habla de lo que los clientes quieren lograr, ya sea resolver un problema, aliviar un dolor, o aspirar a algo mejor a través de nuestro producto. El apartado «para» de la historia debe responder a esta cuestión central en marketing: ¿en qué me beneficia? (*WIIFM* por sus siglas en inglés: *what's in it for me?*).

**Como [cliente ideal], quiero [nuestra solución],  
para [que mis problemas desaparezcan].**

## Herramientas digitales

Muchas empresas prefieren utilizar herramientas digitales en lugar de tableros físicos. Algunas de las más conocidas para realizar historias de usuario son JIRA, Trello, Asana, Miro y Taiga, entre otras.

Veamos nuestro ejemplo de una tarjeta de historia de usuario:



Ahora diseñaremos la misma tarjeta en JIRA, una herramienta que pertenece a la suite de Atlassian. Tenemos disponible toda la información igual que en la tarjeta modelo: identificador, descripción, estimación, criterios de aceptación, etc.



Ilustración 20: ejemplo de una historia de usuario en JIRA.

Lo maravilloso de las herramientas digitales es que nos permiten ampliar o enriquecer la tarjeta tradicional. Podemos incluir imágenes, documentación y otros datos para que estén disponibles de un solo vistazo:

- Compañero que está desarrollando la historia de usuario.
- Sprint en el que se encuentra.
- Estado de la historia de usuario, que dependerá de los que se definan en la herramienta.
- Quién ha creado la historia de usuario.

Veamos otro ejemplo con otra herramienta digital, en este caso Trello (también de la suite de Atlassian):

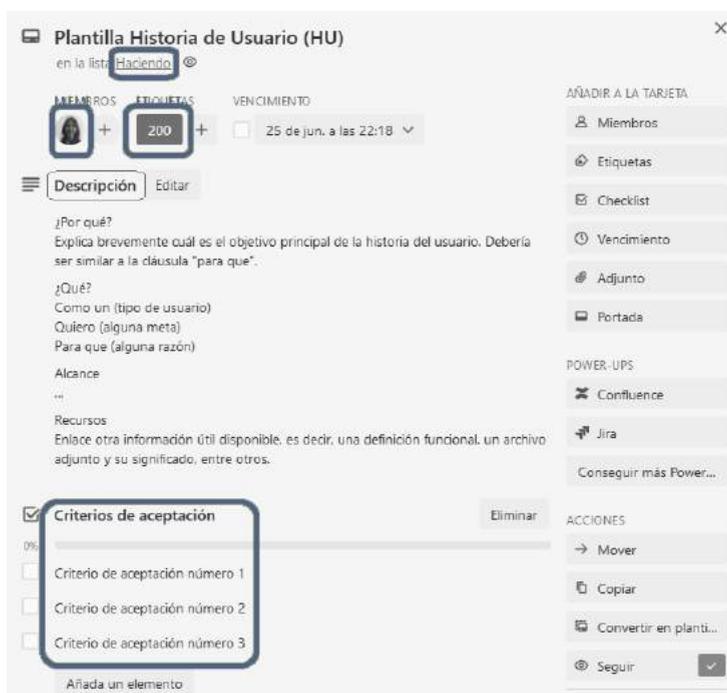


Ilustración 21: ejemplo de una historia de usuario en Trello.

Volvemos a disponer de toda la información de la historia de usuario en la tarjeta de la herramienta:

- Descripción.
- Estimación.
- Criterios de aceptación, etc.

También se puede enlazar y complementar la historia de usuario con herramientas alternativas como JIRA o Confluence. Como toda herramienta digital, se pueden potenciar además con *plugins* o *addons*, que habilitan más funcionalidades para nuestros equipos.

Una ventaja muy importante de este tipo de herramienta es que permite realizar todos los eventos, seguimientos, artefactos, etc., en torno a scrum con equipos remotos y proporciona gran libertad a la hora de distribuir el trabajo.

Aportan además informes y gráficos que resultan ideales para realizar el seguimiento epic, historias de usuario y releases; elementos de larga duración que dan valor al negocio. Se pueden observar y gestionar desde que nacen en la pila de producto hasta su fin, instalados en producción, tras tomar métricas como el ROI.

Entre las desventajas que pueden tener las herramientas digitales encontramos que no están preparadas para dar una visión completa del tablero virtual, con todas sus historias de usuario. Y que pueden llevar a las personas a focalizarse en aprender el proceso de la herramienta concreta, en lugar de fomentar el descubrimiento del proceso propio y su mejora continua.

## Epics

Un epic es una historia de usuario que se distingue por su gran tamaño. Es como una etiqueta que asignamos a una historia cuyo esfuerzo impide completarla de una sola vez o en un solo sprint. A diferencia de las historias de usuario, los epics tienen una alta granularidad y un alto grado de incertidumbre asociados.

Recordemos que todo epic e historia es, en el fondo, una hipótesis; hasta que el usuario la vea o la utilice y nos dé feedback no sabremos si es la solución adecuada. Desde esta perspectiva podemos entender un epic como una serie de experimentos o futuras historias de usuario, que construyen la funcionalidad de forma incremental hasta llegar al resultado deseado.

Esta forma de desarrollar epics se conoce como ciclo lean startup. Trata de desarrollar nuevas ideas, productos y servicios a través del desarrollo guiado por hipótesis (*HDD, hypothesis driven development*). Gracias al feedback que se va obteniendo conforme se validan las hipótesis, el propietario del producto gana comprensión sobre la solución más adecuada.

Barry O'Reilly propone el siguiente patrón en su artículo «*How to Implement Hypothesis-Driven Development*»:

**Creemos que** [esta capacidad]

**resultará en** [este resultado].

**Tendremos confianza para proceder cuando** [veamos una señal medible].

### 1. Creemos que [esta capacidad]

¿Qué funcionalidad desarrollaremos para probar nuestra hipótesis? Al definir la capacidad de prueba del producto o servicio que estamos intentando construir, identificamos la funcionalidad y la hipótesis que queremos probar.

### 2. Resultará en [este resultado]

¿Cuál es el resultado esperado de nuestro experimento? ¿Cuál es el resultado específico que esperamos lograr al desarrollar la capacidad de prueba?

### 3. Tendremos confianza para proceder cuando [veamos una señal medible]

¿Qué señales indicarán que la capacidad que hemos construido es efectiva? ¿Qué métricas predictivas clave (*leading indicators*), cualitativas y/o cuantitativas, mediremos para saber que el experimento ha tenido éxito y pasar a la siguiente etapa?

Ejemplo de un epic de negocio escrito con este patrón:

**Creemos que** aumentar las fotos de los restaurantes en la página de reserva **resultará en** una mejor participación y conversión del cliente.

**Tendremos confianza para proceder cuando** veamos un aumento del 5% en los clientes que revisan las imágenes de los restaurantes y luego proceden a reservar en los próximos 2 minutos.



## Bibliografía

- Alistair Cockburn (2001). *Agile software development*. Boston: Addison-Wesley Professional.
- Barry O'Reilly (2020) *How to Implement Hypothesis-Driven Development*.  
<https://baryoreilly.com/how-to-implement-hypothesis-driven-development/>
- Bill Wake (2003). *INVEST in good stories, and smart tasks*.  
<http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>
- Christiaan Verwijs (2015). *10 useful strategies for breaking down large User Stories (and a cheatsheet)*.  
<http://blog.agilistic.nl/10-useful-strategies-for-breaking-down-large-user-stories-and-a-cheatsheet/>
- Dai Clegg (2004). *Case Method Fast-Track: A RAD Approach*. Boston: Addison-Wesley Longman.
- Don Reinersten (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Pub.
- Gertrudis López (2016). *Esquema con las 10 estrategias de división de historias de usuario*.  
<https://mm.tt/682181724?t=LjoGABkbaF>
- Heather Krebsbach (2016). *Know thy customer: agile's essential guide to user story maps*.  
<https://www.atlassian.com/blog/2016/05/guide-to-agile-user-story-maps>
- Javier Garzás (2014). *Agilidad y Lean. Gestionando los proyectos y negocios del s. XXI*.  
<https://www.miriadax.net/>
- Jeff Patton (2014). *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly Media Inc.
- Jim Ewel (2011). *Agile Marketing*.  
<http://www.agilemarketing.net/>
- Kent Beck (1999). *eXtreme Programming Explained*. Boston: Addison-Wesley Professional.
- Mike Cohn (2004). *User Stories Applied for Agile Software Development*. Boston: Pearson Education, Inc.
- Ron Jeffries, Ann Anderson, Chet Hendrickson (2001). *Extreme Programming Installed*. Boston: Addison-Wesley.

## Tabla de ilustraciones

Ilustración 1: gráfico con las tres fases de la fórmula de las tres Cs.....	5
Ilustración 2: granularidad de la pila de producto.....	7
Ilustración 3: gráfico con los cuatro niveles de tamaño con que trata los requisitos la gestión ágil.....	8
Ilustración 4: ejemplo de una tarjeta de historia de usuario.....	9
Ilustración 5: ejemplos de una historia de usuario.....	11
Ilustración 6: ejemplo de user persona.....	13
Ilustración 7: cartas planning poker con la serie de Fibonacci.....	15
Ilustración 8: cartas de estimación con tallas de camisa.....	16
Ilustración 9: ejemplo de priorización WSJF.....	19
Ilustración 10: ejemplo gherkin.....	21
Ilustración 11: los requerimientos funcionales como restricciones.....	23
Ilustración 12: ejemplo de tarjetas diseñadas por Braintrust.....	29
Ilustración 13: refinamiento de la pila del producto.....	31
Ilustración 14: esquema con las 10 estrategias de división de historias de usuario. .....	32
Ilustración 15: pila de producto plana y pila de producto construida con la técnica user story mapping.....	50
Ilustración 16: ejemplo de backbone de un user story mapping.....	51
Ilustración 17: backbone e historias de usuario de un user story mapping.....	51
Ilustración 18: backbone, historias de usuario y versiones de un user story mapping.....	52
Ilustración 19: previsión de lanzamiento de versiones sobre gráfico de producto..	52
Ilustración 20: ejemplo de una historia de usuario en JIRA.....	55
Ilustración 21: ejemplo de una historia de usuario en Trello.....	55