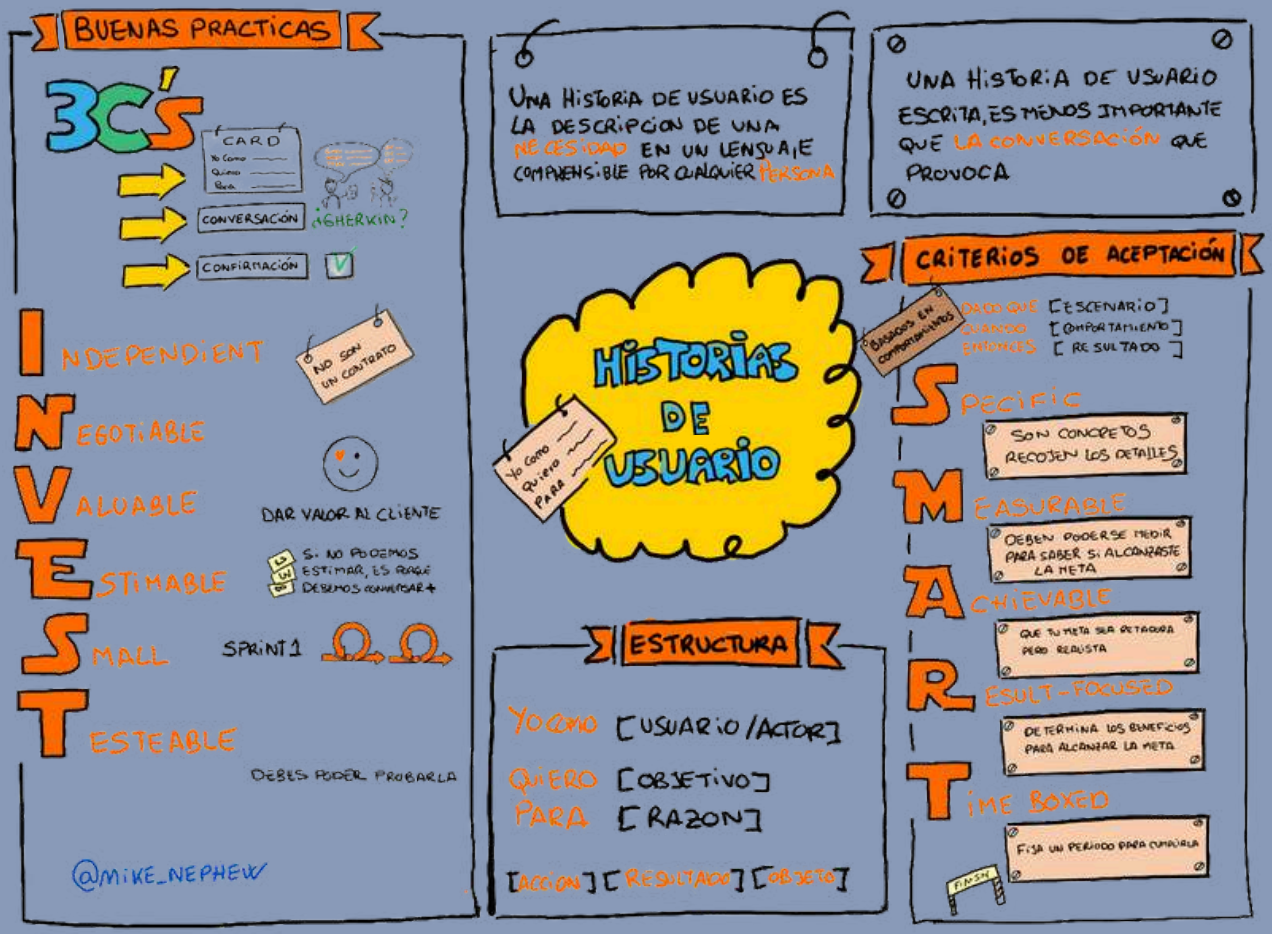
v.5.0 (2026)



HISTORIAS DE USUARIO

Scrum Manager®
Uncovering better ways

Historias de usuario

Formación complementaria

Versión 5.0.

Scrum Manager ®

Fecha de la versión: agosto 2026.
Diseño de portada: Miguel Ángel Sobrino.
Autores en colaboración con Scrum Manager: Alexander Menzinsky, Gertrudis López, Juan Palacio, Miguel Ángel Sobrino, Rubén Álvarez y Verónica Rivas.
Uncovering Better Ways SLU es la editora y propietaria de los derechos de distribución, que libera en los términos de la licencia Creative Commons-by-nd-nc 4.0.
Derechos registrados en Safe Creative. Nº de registro: 2512134011732

Índice de contenidos

Ingeniería de requisitos ágil....4
Historias de usuario....4
De temas y épicas a tareas....6
Anatomía de una historia de usuario....9
Descripción....11
Valor de negocio....13
Estimación....14
Prioridad....16
Criterios de aceptación....19
Requerimientos no funcionales....22
Definición de hecho....24
INVEST y las historias de usuario de calidad....26
División de historias de usuario....29
Las 10 estrategias de Verwijs....31
Otras formas de toma de requisitos....50
Casos de uso....50
Requisitos funcionales....52
Estrategias de transición y coexistencia....53
Cuándo usar cada enfoque....54
Historias técnicas....55
Características de las historias técnicas....55
Tipos de historias técnicas....56
Deuda técnica....58
Priorización de historias técnicas....58
User Story Mapping....60
Elementos de User Story Mapping....61
Técnicas relacionadas....63
Inteligencia artificial e historias de usuario....64
Usos apropiados de la IA....64
Riesgos y limitaciones críticas de la IA....68
Apéndices....72
Historias de usuario en otros ámbitos....72
Herramientas digitales....73
Resumen historias de usuario....77
Mejora continua y control de calidad Scrum Manager....78
Bibliografía....79

Herramientas y plataformas de IA..... 81
Información, recursos y comunidad profesional..... 81
Redes sociales y profesionales.....81
Tabla de ilustraciones..... 82

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 (Mike Cohn 2004). El foco está puesto en qué necesidades o problemas soluciona lo que se va a construir.

Su origen viene de la metodología *eXtreme Programming* (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.

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 las tres Cs, definida por Ron Jeffries en 2001, para capturar funcionalidades:

- ***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 los tres aspectos 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 los 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 de 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.

En el contexto actual

Las historias de usuario han evolucionado desde su origen en *eXtreme Programming* para adaptarse a contextos contemporáneos de desarrollo de software:

- **Equipos distribuidos y remotos:** las historias facilitan la colaboración asíncrona mediante herramientas digitales (Jira, Azure DevOps, Trello), manteniendo el equilibrio entre documentación escrita y conversación.
- **DevOps y entrega continua:** los criterios de aceptación se automatizan como tests, permitiendo integración y despliegue continuos del software.
- **Escalado ágil:** algunos marcos de escalado introducen niveles adicionales (p. ej., *features* y *capabilities* en SAFe); mientras que otros mantienen los artefactos de scrum y coordinan múltiples equipos sobre un *backlog* compartido (p. ej., LeSS, Nexus).

A pesar de estos avances tecnológicos, el principio fundamental permanece: **las historias de usuario son herramientas de comunicación** que facilitan conversaciones y construyen entendimiento compartido entre negocio y tecnología.

Es decir, no son especificaciones completas: son recordatorios para conversar. Por coherencia con ese propósito, la estimación se apoya en escalas relativas y la IA se puede usar como asistente para acelerar tareas mecánicas, manteniendo siempre validación humana y conversación.

De temas y épicas a tareas

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

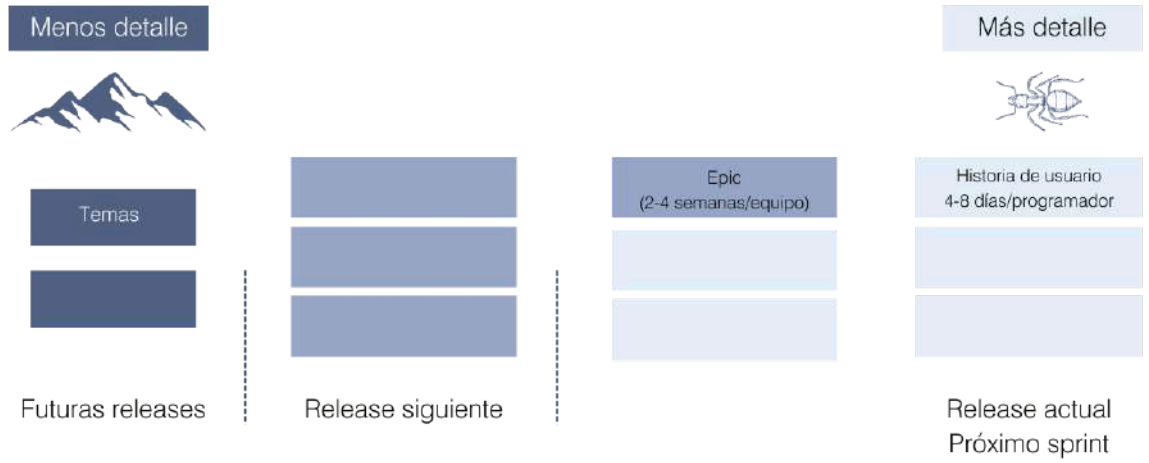


Ilustración 2: granularidad de la pila de producto.

Épicas (*epic*)

Es una historia de usuario de gran tamaño, con un mayor grado de incertidumbre[1]. Es como una etiqueta que asignamos a una historia cuyo esfuerzo impide completarla de una sola vez o en un solo sprint.

[1] La granularidad indica el tamaño de grano con que se divide el trabajo de la pila de producto: las épicas son de grano grueso (piezas grandes, poco detalladas) y las tareas de grano fino (piezas pequeñas y detalladas). Cuanto mayor es la prioridad de un elemento, más fino conviene que sea su grano.

Marcar una historia como épica 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.

Toda épica 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 una épica 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 épicas 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?

> ***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.*

Tema

Una colección de épicas 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 épicas «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.

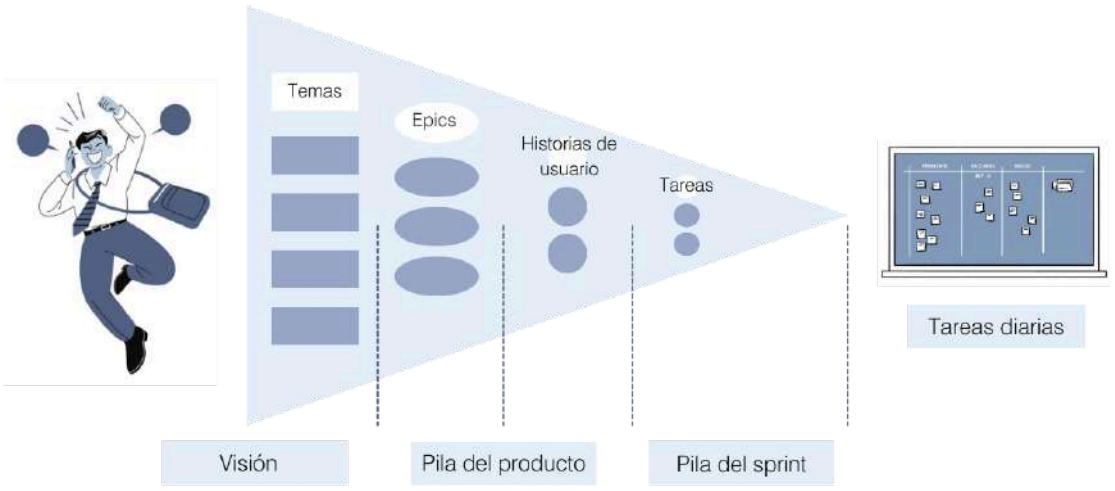


Ilustración 3: gráfico con los cuatro niveles de tamaño con que trata los requisitos la gestión ágil.

***

En scrum y en los métodos ágiles en general, una pila de producto puede contener tanto historias de usuario como temas y épicas.

Al final de la pila de producto está lo menos prioritario: las épicas, 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.

Anatomía de una historia 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. Sin embargo, mantener una estructura consistente dentro del equipo y organización facilita la comunicación y la comprensión compartida. La flexibilidad no significa arbitrariedad.

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.**

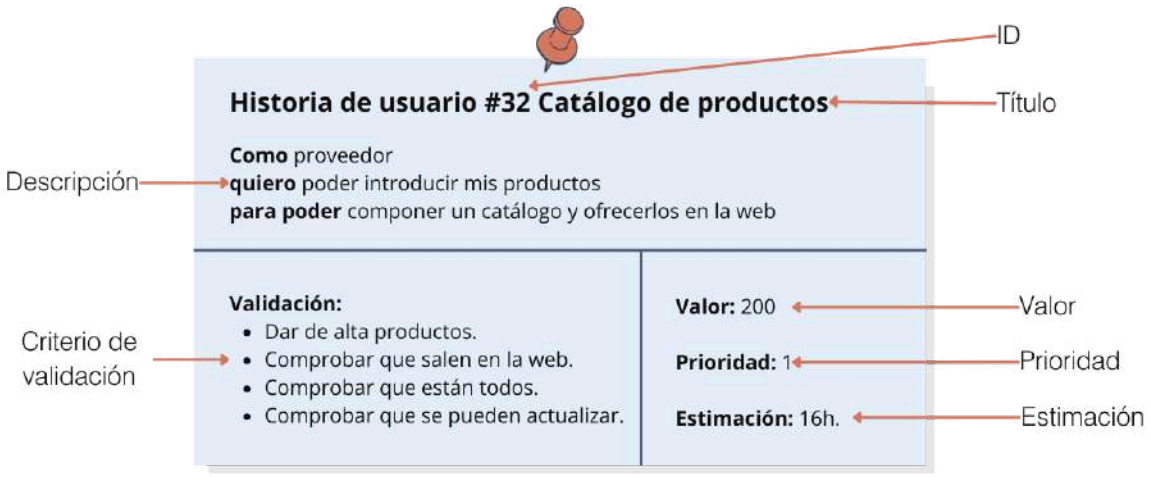


Ilustración 4: ejemplo de una tarjeta de historia de usuario

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 tamaño relativo del trabajo necesario para implementar la historia. Puede estimarse en escala relativa (puntos de historia[2]), 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.

[2]Los puntos de historia son unidades que expresan el tamaño relativo del trabajo (complejidad, incertidumbre, riesgo y esfuerzo) comparándolo con una referencia acordada por el equipo.

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	
Descripción: Como cliente quiero cambiar la dirección de envío de un pedido para que me pueda llegar a casa o a la oficina.	
Validación: El cliente puede cambiar la dirección de entrega de cualquiera de los pedidos que tiene pendiente de envío.	

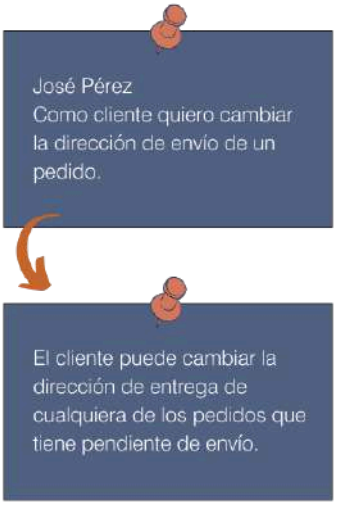


Ilustración 5: ejemplos de una historia de usuario

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 utilizada 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 interactúa 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.

Otra técnica, conocida como ***value poker***, consiste en estimar el valor igual 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 34.

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.

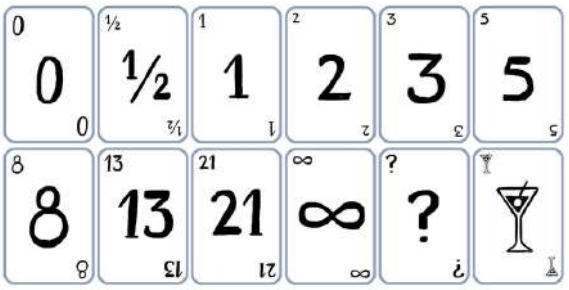


Ilustración 7: cartas planning poker con la serie de Fibonacci

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 (∞) 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.

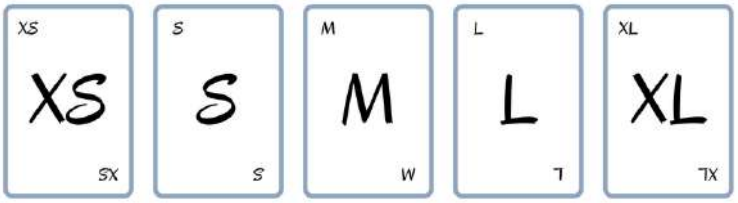


Ilustración 8: cartas de estimación con tallas de camisa

Para la estimación de elementos grandes como las épicas 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 1994 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 on Investment* (ROI, retorno de la inversión).

ROI = Valor de negocio/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.

WSJF = Coste de demora/Tamaño

Una técnica de priorización **lean** muy recomendable y completa es WSJF (*Weighted Shortest Job First*) de 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:

Coste de demora = Valor de negocio + Criticidad en el tiempo + Reducción del riesgo y valor de oportunidad

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

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 toma de decisiones. Durante la revisión del sprint, el propietario de producto comprobará, a través de estos criterios de aceptación y la definición de hecho (DoD), 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] [titulo 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.

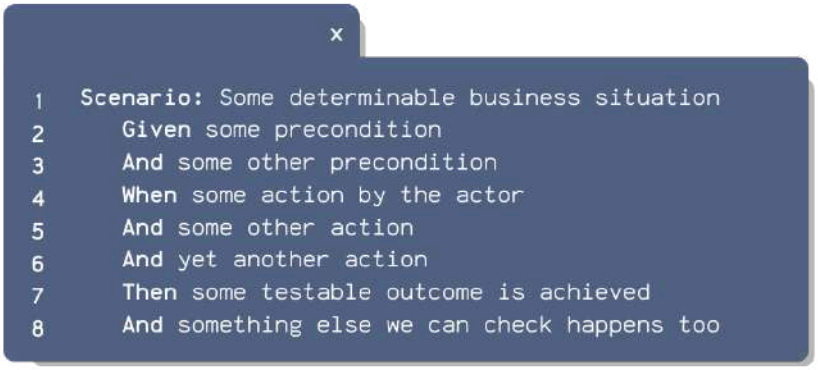


Ilustración 9: ejemplo gherkin

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 se carga el importe en la cuenta
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 las épicas e historias de usuario, cuyos criterios de aceptación se centran en comportamientos y funciones específicas. Sin embargo, una historia o épica 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).

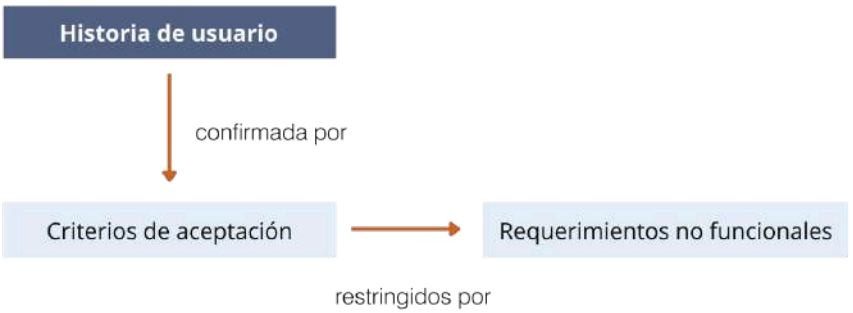


Ilustración 10: los requerimientos no funcionales como restricciones

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 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 acuerdo explícito de calidad 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 comentarios relevantes cuando aporten claridad.
- 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 el despliegue resueltas.
- Comunicación de release y formación a usuarios hechas.
- Aprobada por el propietario del producto.

INVEST y las historias de usuario de calidad

El método INVEST fue formulado por Bill Wake en su artículo «INVEST in Good Stories, and SMART Tasks» publicado en 2003, y posteriormente popularizado por Mike Cohn en su libro *User Stories Applied* (2004). 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 dividirlas 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 que la historia sea suficientemente abierta para permitir conversación y negociación (evitando imponer el 'cómo'), pero no ambigua: debe expresar con claridad quién, qué necesita y para qué.

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 estimado 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:



Ilustración 11: 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 de 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 redunda 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.

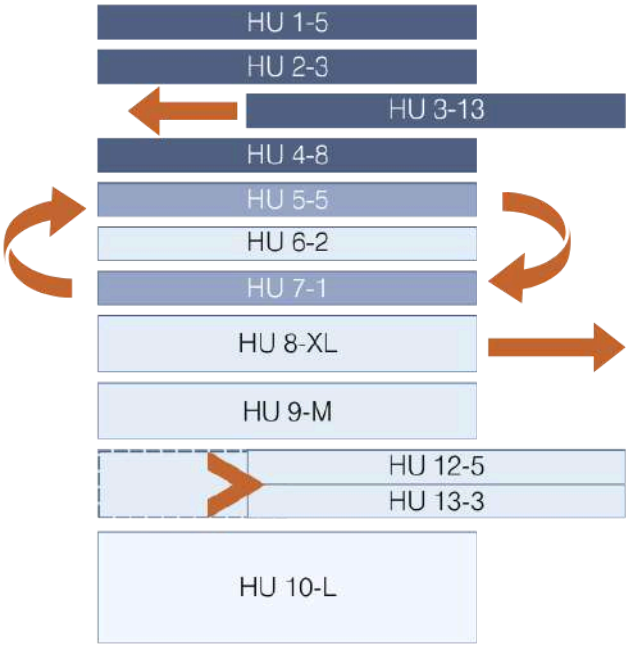


Ilustración 12: refinamiento de la pila de producto

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.

Las 10 estrategias de Verwijs

En 2009 Richard Lawrence documentó patrones recurrentes que observó al trabajar con múltiples equipos de desarrollo. Su trabajo *Patterns for Splitting User Stories* identifica técnicas específicas aplicables a diferentes contextos.

En 2015 Christiaan Verwijs las popularizó y las presentó como **10 estrategias para dividir historias de usuario de forma vertical**, para obtener historias funcionales y de menor tamaño:



Ilustración 13: 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: construir 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. Se puede tratar 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 (producto mínimo 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.

> *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 dividirlas 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 movimientos bancarios de forma segura.*

División considerando varios flujos posibles:

> ***Happy flow***
> *Como usuario*
> *quiero loguearme con mi cuenta 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.

Modelo SPIDR

Mike Cohn propone que aproximadamente el 80% de las historias de usuario pueden dividirse usando solo cinco técnicas, agrupadas en el acrónimo SPIDR:

- **S - Spike** (Extraer investigación).
- **P - Path** (Dividir por caminos alternativos).
- **I - Interface** (Dividir por complejidad de interfaz o plataformas).
- **D - Data** (Dividir por tipos de datos o parámetros).
- **R - Rules** (Dividir por reglas de negocio).

S - SPIKE

Un spike es una actividad de investigación limitada en el tiempo que genera conocimiento necesario para entender mejor una historia o reducir incertidumbre técnica o funcional. Puede incluir prototipos, pruebas de concepto, o investigación de viabilidad.

¿Cuándo aplicar esta técnica?

Cuando la historia contiene incertidumbre técnica o funcional significativa que impide estimarla con confianza o implementarla eficazmente. Los spikes son especialmente útiles cuando:

- No sabemos si una tecnología o integración es viable.
- Necesitamos evaluar diferentes enfoques técnicos.
- La complejidad técnica es desconocida.
- Requerimos investigación de usuario o validación de concepto.

Ejemplo

Historia original

Como ciclista
quiero pagar mi compra con tarjeta de crédito
para poder adquirir accesorios para mi bicicleta sin usar efectivo.

División extrayendo spike

Spike (historia técnica):

Investigar la integración con la pasarela de pagos de nuestro banco para determinar si cumple nuestros requisitos de seguridad, tiempos de respuesta y costes de transacción.

Criterios de aceptación del spike (preguntas a responder):

- *¿La pasarela soporta los tipos de tarjeta que necesitamos?*
- *¿Los tiempos de respuesta son menores a 3 segundos?*
- *¿Cumple con PCI-DSS?*

- *¿Cuál es el coste por transacción?*

Historia resultante (tras completar el spike):

Como ciclista
quiero pagar con tarjeta de crédito usando la pasarela [nombre]
para poder completar mi compra de forma segura y rápida.

P - PATH

Si un usuario puede lograr un objetivo de múltiples maneras (por ejemplo, pagar con tarjeta, PayPal y/o transferencia; compartir en Facebook, Twitter y/o email), cada camino alternativo puede convertirse en una historia separada.

¿Cuándo aplicar esta técnica?

Cuando la historia involucra:

- Múltiples flujos de trabajo diferentes.
- Diferentes formas de lograr el mismo objetivo.
- Varios métodos de entrada o interacción.
- Caminos *happy path* y *unhappy path*.
- Diferentes operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- Múltiples roles que realizan la misma funcionalidad de forma distinta.

Ejemplo

Historia original

Como usuario de la plataforma
quiero compartir un video
para poder recomendarlo a mis contactos.

División por caminos

- *Como usuario de la plataforma*
  quiero copiar el enlace del video al portapapeles
  para poder compartirlo por cualquier medio que elija (WhatsApp, email, etc.).
- *Como usuario de la plataforma*
  quiero compartir el video directamente en Facebook
  para poder llegar rápidamente a mis amigos en esa red social.
- *Como usuario de la plataforma*
  quiero compartir el video directamente en Twitter
  para poder difundirlo entre mis seguidores de forma inmediata.

- *Como usuario de la plataforma*
  quiero enviar el video por email desde la propia plataforma
  para poder compartirlo con contactos que no usan redes sociales.

I - INTERFACE

Esta técnica consiste en implementar primero una versión simple de la interfaz y posteriormente añadir complejidad, o bien soportar diferentes plataformas de forma incremental.

¿Cuándo aplicar esta técnica?

Cuando la historia:

- Debe funcionar en múltiples dispositivos o plataformas (web, móvil, tablet).
- Tiene una interfaz que puede implementarse en versiones de complejidad creciente.
- Requiere soporte para diferentes navegadores.
- Incluye características avanzadas de UI que no son esenciales inicialmente.

Ejemplo

Historia original

Como comprador
quiero buscar productos por categoría
para poder encontrar rápidamente lo que necesito.

División por complejidad (simple → compleja):

Como comprador
quiero buscar productos tecleando la categoría en un campo de texto
para poder encontrar productos de forma rápida y directa.

Como comprador
quiero seleccionar categorías desde un menú desplegable con iconos
para poder explorar las opciones de manera más intuitiva.

Como comprador
quiero navegar por un catálogo interactivo con imágenes de cada categoría
para poder descubrir productos de forma más atractiva y exploratoria.

D - DATA

Esta técnica consiste en implementar la funcionalidad para un subconjunto de datos, categorías o parámetros primero, y luego expandir gradualmente a otros tipos de datos.

¿Cuándo aplicar esta técnica?

Cuando la historia maneja:

- Múltiples tipos de datos o categorías de información.
- Diferentes parámetros de búsqueda o filtrado.
- Varios formatos de entrada.
- Distintas fuentes de datos.
- Diferentes atributos o propiedades de una entidad.

Ejemplo

Historia original:

Como comensal
quiero buscar restaurantes
para poder encontrar opciones que se ajusten a mis preferencias y situación.

División por tipos de datos de búsqueda:

Como comensal
quiero buscar restaurantes por tipo de cocina (italiana, japonesa, mexicana...)
para poder encontrar el estilo de comida que me apetece hoy.

Como comensal
quiero buscar restaurantes por rango de precios (€, €€, €€€)
para poder ajustarme a mi presupuesto disponible.

Como comensal
quiero buscar restaurantes por distancia desde mi ubicación
para poder encontrar opciones cercanas y convenientes.

Como comensal
quiero buscar restaurantes por valoración de usuarios (estrellas)
para poder elegir lugares bien puntuados por otros comensales.

Como comensal
quiero buscar restaurantes por disponibilidad de mesa
para poder reservar inmediatamente sin esperas.

R - RULES

Las reglas de negocio son restricciones, condiciones o lógica especial que determina cómo debe comportarse el sistema en diferentes situaciones. Incluyen regulaciones legales, políticas de la empresa, cálculos complejos, o condiciones especiales.

¿Cuándo aplicar?

Cuando la historia involucra:

- Múltiples reglas de negocio que pueden implementarse incrementalmente.
- Restricciones legales o regulatorias.
- Lógica condicional compleja.
- Casos especiales o excepciones.
- Diferentes políticas según tipo de cliente, región, etc.

El enfoque recomendado es implementar primero una versión simple de la funcionalidad con reglas básicas o incluso simplificadas (por ejemplo, un mensaje manual), y luego añadir reglas más complejas progresivamente.

Ejemplo

Historia original

Como comprador
quiero finalizar mi pedido
para poder recibir los productos que he seleccionado en mi domicilio.

División por reglas de negocio

Como propietario de la tienda
quiero establecer un pedido mínimo de 10€
para poder mantener la rentabilidad del negocio evitando pedidos muy pequeños.

Como propietario de la tienda
quiero aplicar gastos de envío de 5€ para pedidos entre 10€ y 50€
para poder cubrir los costes logísticos de envíos pequeños.

Como propietario de la tienda
quiero ofrecer envío gratuito para pedidos superiores a 50€
para poder incentivar compras de mayor volumen.

Como propietario de la tienda
quiero aplicar un recargo de 15€ para envíos fuera de la UE
para poder cubrir los costes adicionales de envío internacional.

Como propietario de la tienda
quiero reservar automáticamente el stock durante 48 horas tras el pedido
para poder asegurar disponibilidad mientras el cliente completa el pago.

Como propietario de la tienda
quiero cancelar automáticamente pedidos no pagados tras 48 horas
para poder liberar el stock reservado y ofrecerlo a otros clientes.

Aplicación práctica del modelo SPIDR

Cuando te enfrentes a una historia grande durante el refinamiento del backlog, hazte estas preguntas en orden:

1. ¿Hay incertidumbre técnica o funcional significativa? → Considera extraer un **SPIKE**.
2. ¿El usuario puede lograr el objetivo de múltiples formas? → Divide por **PATH** (caminos alternativos).
3. ¿La interfaz tiene múltiples niveles de complejidad o plataformas? → Divide por **INTERFACE**.
4. ¿La historia maneja diferentes tipos de datos o parámetros? → Divide por **DATA**.
5. ¿Involucra múltiples reglas de negocio o restricciones? → Divide por **RULES**.

Una misma historia puede dividirse usando múltiples técnicas SPIDR de forma sucesiva. Por ejemplo, primero divides por PATH (operaciones CRUD), y luego divides cada operación por DATA (tipos de productos). La división es un proceso iterativo.

Otras formas de toma de requisitos

Las historias de usuario no son la única técnica para capturar requisitos de software. A lo largo de la historia de la ingeniería de software se han desarrollado diversos enfoques, cada uno con sus fortalezas y contextos de aplicación. Dos de las técnicas más comunes son los casos de uso y requisitos funcionales tradicionales.

Casos de uso

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

¿Qué son los casos de uso?

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. Los casos de uso incluyen elementos formales como:

- **Actores:** los roles que interactúan con el sistema.
- **Precondiciones:** estado del sistema antes de ejecutar el caso de uso.
- **Flujo principal:** secuencia de pasos en el escenario exitoso.
- **Flujos alternativos:** variaciones del flujo principal.
- **Flujos de excepción:** qué ocurre cuando algo falla.
- **Postcondiciones:** estado del sistema después de ejecutar el caso de uso.

Diferencias con las historias de usuario

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. Mientras que los casos de uso se centran 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.**

Aspecto	Casos de uso	Historias de usuario
Lenguaje	UML (técnico, formal)	Lenguaje natural del usuario
Enfoque	Especificación de requisitos (el cómo)	Captura de requisitos (el qué)
Nivel de detalle	Alto (flujos, extensiones, precondiciones)	Bajo (placeholder para conversación)
Documentación	Diagramas UML + descripciones extensas	Tarjeta + criterios de aceptación simples
Momento de elaboración	Análisis previo extenso	*Just-in-time*, poco antes del desarrollo
Criterios de aceptación	Matrices de seguimiento con porcentajes	Binario (cumple o no cumple)
Mantenimiento	Requiere actualización de diagramas UML	Mínimo (se descartan tras implementar)
Autor principal	Analista funcional o arquitecto	Product owner + equipo + usuario
Flexibilidad ante cambios	Requiere reelaboración formal	Alta, cambios incorporados fácilmente

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 adopta métodos ágiles, generalmente conviene priorizar las historias de usuario sobre los casos de uso tradicionales, dado su enfoque en la conversación y la entrega iterativa. Sin embargo, ciertos contextos regulados o sistemas complejos pueden beneficiarse de un enfoque híbrido documentado.

¿Cuándo pueden seguir siendo útiles los casos de uso?

Aunque las historias de usuario son generalmente más efectivas en contextos ágiles por su simplicidad y enfoque en la conversación, los casos de uso pueden seguir aportando valor en situaciones específicas:

- **Sistemas con lógica compleja e interdependiente** donde es crucial documentar formalmente todos los flujos posibles antes de comenzar el desarrollo.
- **Proyectos altamente regulados** (sector salud, aeronáutico, financiero) donde se requiere documentación formal para auditorías o certificaciones.
- **Equipos muy distribuidos** donde la comunicación síncrona es difícil y se necesita documentación más exhaustiva.

- **Proyectos de integración con sistemas *legacy*** donde mapear casos de uso existentes ayuda a entender el sistema actual.
- **Contratos de alcance cerrado** donde se requiere especificación detallada previa al desarrollo.

En contextos ágiles puros, donde los requisitos son volátiles, el equipo es multifuncional y la colaboración es continua, las historias de usuario suelen ser más apropiadas por su ligereza y adaptabilidad.

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.

¿Qué son los requisitos funcionales tradicionales?

Un requisito funcional tradicional es una declaración formal y detallada de una capacidad que el sistema debe proporcionar. Suelen seguir estándares como IEEE 830 (ahora IEEE 29148) y se documentan en especificaciones de requisitos de software (SRS: *Software Requirements Specification*).

Un requisito funcional típico tiene esta estructura:

ID: REQ-FUN-001
Título: autenticación de usuario
Prioridad: alta
Descripción: el sistema deberá permitir a los usuarios autenticarse mediante nombre de usuario y contraseña. El sistema validará las credenciales contra la base de datos de usuarios. Si las credenciales son correctas, el sistema creará una sesión válida por 24 horas. Si las credenciales son incorrectas, el sistema mostrará el mensaje «Usuario o contraseña incorrectos» y bloqueará la cuenta tras 5 intentos fallidos consecutivos.
Precondiciones: el usuario debe estar registrado en el sistema.
Postcondiciones: el usuario tiene una sesión activa.
Dependencias: REQ-FUN-015 (Gestión de sesiones).

Diferencias clave con las historias de usuario

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 post-its, 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.

Aspecto	Requisitos funcionales	Historias de usuario
Formato	Especificación formal detallada	Frase simple en lenguaje natural
Nivel de detalle inicial	Muy alto (comportamiento completo)	Bajo (solo lo esencial)
Elaboración	Análisis previo extenso (fase de requisitos)	Conversación continua (refinamiento progresivo)
Autor	Analista de requisitos	Product owner con input del usuario final
Audiencia	Técnica (desarrolladores, testers, arquitectos)	Mixta (negocio y técnica)
Documentación	Documento formal extenso (SRS)	Tarjetas físicas o digitales simples
Trazabilidad	IDs únicos, matrices de trazabilidad	Backlog priorizado
Gestión de cambios	Proceso formal de change request	Cambios bienvenidos en cualquier momento
Criterios de éxito	Cumplimiento de especificación	Conversación → Criterios de aceptación
Filosofía	«Documentación comprensiva»	«Conversación sobre documentación»

Estrategias de transición y coexistencia

Adoptar enfoques ágiles no exige abandonar por completo técnicas previas; lo recomendable es **seleccionar y combinar prácticas según el contexto**, manteniendo trazabilidad o documentación formal cuando aporte valor o sea obligatoria.

Enfoque híbrido por capas

- Requisitos de alto nivel o épicas documentadas formalmente (para cumplimiento normativo o contractual).
- Historias de usuario para implementación iterativa de esos requisitos de alto nivel.
- Trazabilidad entre documentos tradicionales e historias de usuario.

Transición gradual

1. Comenzar con historias de usuario en proyectos piloto de bajo riesgo.
2. Mantener requisitos formales en sistemas críticos o regulados.
3. Ir aumentando progresivamente el uso de historias según la madurez del equipo.

Documentación *just-enough*

- Mantener documentación formal donde es legalmente necesario o aporta valor real.
- Usar historias de usuario donde la flexibilidad y rapidez son más importantes.
- Generar documentación formal automáticamente desde historias cuando sea posible (herramientas de trazabilidad).

Cuándo usar cada enfoque

La decisión sobre qué técnica emplear debería basarse en una evaluación pragmática del contexto:

Contexto del proyecto	Técnica más apropiada
Producto digital iterativo, startup	Historias de usuario
Sistema médico con certificación FDA	Requisitos formales + casos de uso
App móvil con feedback continuo de usuarios	Historias de usuario
Sistema de control aeronáutico	Requisitos formales + casos de uso
Plataforma web en constante evolución	Historias de usuario
Integración con múltiples sistemas legacy	Casos de uso para mapeo + historias para nuevas features
Proyecto con contrato de alcance cerrado	Requisitos formales (con posibilidad de historias internamente)
Desarrollo interno de herramientas corporativas	Historias de usuario (más ágil y colaborativo)

No hay enfoque único correcto

El debate entre casos de uso, requisitos funcionales e historias de usuario no debería plantearse en términos de «correcto vs incorrecto» o «moderno vs obsoleto». Cada técnica tiene su lugar y su valor según el contexto.

Las historias de usuario destacan en entornos donde la adaptabilidad, la colaboración continua y la entrega incremental de valor son prioritarias. Los casos de uso y requisitos funcionales tradicionales siguen siendo valiosos en contextos donde la documentación formal, la trazabilidad exhaustiva y la especificación previa detallada son necesarias o regulatoriamente requeridas.

La madurez profesional consiste en conocer bien todas las herramientas disponibles y saber cuándo aplicar cada una. Como afirma el *Manifiesto Ágil*: valoramos «colaboración con el cliente sobre negociación contractual» y «respuesta ante el cambio sobre seguir un plan», pero eso no significa que los contratos y los planes carezcan de valor, simplemente que valoramos más lo primero.

Historias técnicas

La pila de producto contiene todos los elementos que se van a construir: historias de usuario, épicas y temas. Pero para poder realizar planificaciones tanto de sprint como de release, debe contener además todo el trabajo pendiente, incluyendo necesidades de carácter técnico. Las historias técnicas suelen aportar **valor indirecto** (sostenibilidad, reducción de riesgo, rendimiento, seguridad), aunque no siempre son valor visible para el usuario final.

Por poner ejemplos, las historias técnicas incluyen acciones como preparar un servidor web, 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.

Características de las historias técnicas

Escritura y formato

Las historias técnicas se escriben directamente en **texto técnico claro y preciso**, sin un patrón como ocurre con las historias de usuario.

Criterios de aceptación técnicos

También tienen criterios de aceptación asociados que se comprueban en la revisión de sprint, pero estos son **verificados por la audiencia técnica correspondiente** (equipo, arquitecto, otros desarrolladores), no por el propietario del producto o usuarios finales. Los criterios suelen ser técnicos y mensurables:

- «La migración se completa sin pérdida de datos (verificado con checksums)».
- «El tiempo de respuesta de consultas se reduce de 2s a <200ms».
- «La cobertura de tests se mantiene por encima del 80%».
- «La solución es compatible con versiones anteriores de la API».

Propiedad y responsabilidad

El propietario del producto es responsable de la pila de producto incluyendo las historias técnicas. Sin embargo, dada la naturaleza especializada de estas historias, el equipo técnico tiene un rol protagónico en su definición, estimación y priorización relativa.

Es recomendable que el PO confíe en el criterio técnico del equipo para entender el por qué y cuándo de estas historias, pero manteniendo la responsabilidad final de priorizarlas en el contexto de toda la pila.

Importancia de la visibilidad

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. Sin esta visibilidad es frecuente que las historias técnicas sean percibidas como «pérdida de tiempo» o «perfeccionismo innecesario» por *stakeholders* no técnicos, lo cual genera fricción y deuda técnica acumulada.

Tipos de historias técnicas

Se pueden identificar diferentes tipos de historias técnicas:

Arquitectura

Construyen elementos como las APIs, que crean la estructura, funcionamiento e interacción entre distintas partes del software. Estas historias definen decisiones arquitectónicas fundamentales que impactan múltiples funcionalidades.

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.

Deuda técnica

El término «deuda técnica» fue acuñado por Ward Cunningham en 1992 con la siguiente metáfora: acumular deuda técnica es como pedir un préstamo financiero. Obtienes velocidad ahora (puedes lanzar antes), pero pagas intereses después en forma de mayor dificultad, lentitud y coste para hacer cambios.

Tipos de deuda técnica

- **Deuda deliberada:** en situaciones excepcionales y críticas, el equipo puede decidir conscientemente tomar atajos muy específicos y acotados para cumplir una ventana de oportunidad de negocio estratégica. Esta práctica debe ser excepcional, no habitual. Si un equipo incurre en deuda deliberada frecuentemente, indica problemas de planificación o estimación que deben abordarse.
- **Deuda accidental:** surge por falta de conocimiento o comprensión del problema, descubierta cuando el código dificulta cambios futuros.
- **Deuda por negligencia:** ignorar buenas prácticas conocidas por presión o falta de disciplina (no escribir tests, código duplicado sin justificación).

Gestión de la deuda técnica

Como en finanzas, la deuda técnica acumula «intereses»: velocidad decreciente, bugs crecientes, menor moral del equipo, y mayor coste de cambio. El interés se paga en cada sprint, no solo cuando decidamos abordar la deuda.

La deuda técnica puede ser aceptable cuando permite validar hipótesis rápidamente, cumplir ventanas de oportunidad críticas, o aprender del dominio. Lo importante es que sea consciente, documentada, y con plan claro de pago.

Estrategias de gestión

- Refactorizar progresivamente cada vez que se toca código legacy.
- Reservar un porcentaje fijo de capacidad del sprint para historias técnicas.
- Priorizar según impacto en velocidad futura y riesgo.

Priorización de historias técnicas

Uno de los mayores desafíos en equipos ágiles es equilibrar historias funcionales (valor directo al usuario) e historias técnicas (sostenibilidad del sistema). Como referencia general, equipos maduros en productos establecidos suelen dedicar entre 20-30% de la capacidad del sprint a historias técnicas. Sin embargo, este ratio es altamente contextual. Por ejemplo, en proyectos nuevos con arquitectura sólida puede ser de un 10-15%; en productos con alta deuda técnica acumulada puede ser de un 40-50% temporalmente...

Técnicas de priorización

- **Coste de demora técnico:** ¿cuánto nos cuesta NO hacer esta historia técnica? Deuda en módulos críticos que generan bugs constantes tiene prioridad alta; mejoras «nice to have» tienen prioridad baja.
- **Matriz impacto-esfuerzo:** clasificar en alto/bajo impacto y alto/bajo esfuerzo. Priorizar historias de alto impacto y bajo esfuerzo primero (*quick wins*), planificar las de alto impacto y alto esfuerzo (proyectos estratégicos).
- **Tiempo fijo reservado:** establecer política de equipo como «25% de cada sprint se dedica a historias técnicas», dando predictibilidad a negocio y sostenibilidad al sistema.
- **Señales de alerta de insuficiente inversión técnica:** velocidad decreciente sprint tras sprint, bugs crecientes, tiempo de desarrollo aumentando, equipo frustrado, y miedo a cambiar código frágil.

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 épicas.

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 épicas que se prevén como parte del producto a construir.



Ilustración 14: 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 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.

Elementos de User story mapping

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 15: 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 16: 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 17: 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.

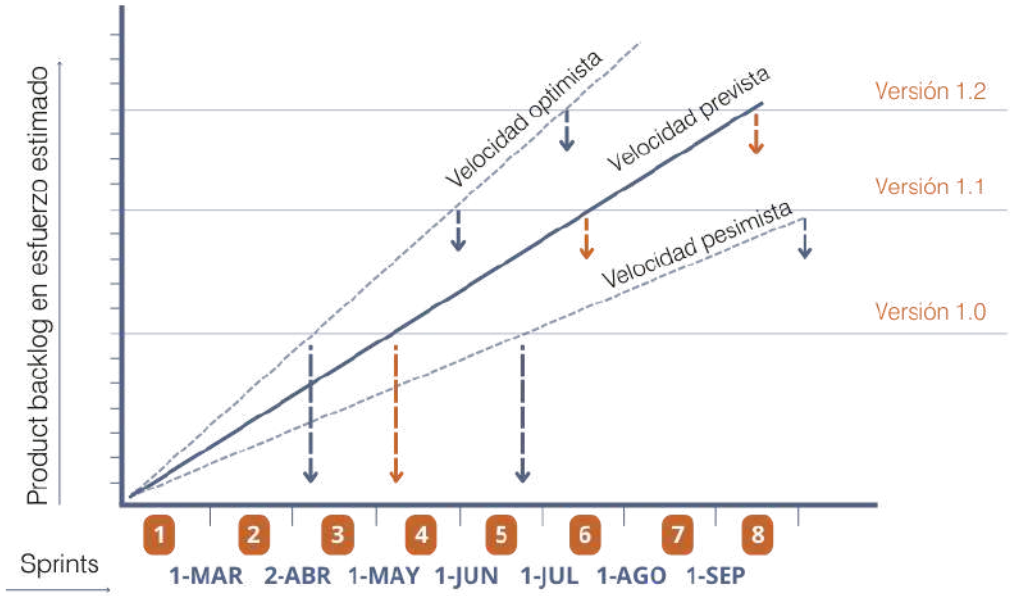


Ilustración 18: previsión de lanzamiento de versiones sobre gráfico de producto

Técnicas relacionadas

El User Story Mapping no es la única técnica para visualizar y priorizar trabajo en entornos ágiles. Otras técnicas complementarias o alternativas incluyen:

Impact mapping

Desarrollado por Gojko Adzic, el Impact mapping ayuda a conectar objetivos de negocio con entregas de producto. Se estructura en cuatro niveles:

- **¿Por qué? -** Objetivo de negocio.
- **¿Quién? -** Actores que pueden producir el impacto deseado.
- **¿Cómo? -** Impactos en el comportamiento de esos actores.
- **¿Qué? -** Entregas (features) que pueden producir esos impactos.

Impact mapping es especialmente útil cuando necesitas asegurar que cada feature del backlog está justificada por un objetivo de negocio claro, evitando construir funcionalidades «porque sí». Se complementa bien con story mapping: impact mapping ayuda a decidir QUÉ construir, story mapping ayuda a organizar CÓMO construirlo secuencialmente.

Opportunity Solution Trees

Popularizados por Teresa Torres, los Opportunity Solution Trees ayudan a equipos de producto a conectar outcomes deseados con soluciones específicas a través de oportunidades identificadas:

- Outcome deseado (raíz del árbol).
- Oportunidades descubiertas mediante investigación de usuarios.
- Soluciones posibles para cada oportunidad (pueden ser experimentos, features, etc.).

Esta técnica es valiosa cuando trabajas en descubrimiento continuo de producto (continuous discovery) y quieres asegurar que estás resolviendo problemas reales de usuarios, no solo implementando features. Al igual que impact mapping, puede usarse antes de o en combinación con story mapping.

Inteligencia artificial e historias de usuario

La llegada de herramientas de inteligencia artificial generativa (ChatGPT, Claude, Gemini...) ha planteado nuevas posibilidades y riesgos en la gestión de historias de usuario. Estas tecnologías prometen acelerar procesos, generar contenido y analizar grandes volúmenes de información en segundos. Sin embargo, es fundamental establecer una postura clara sobre su rol en el desarrollo ágil.

La inteligencia artificial es una herramienta de apoyo, no un sustituto de la colaboración humana ni un miembro del equipo. Las historias de usuario existen para facilitar conversaciones entre personas. La IA puede optimizar ciertos procesos repetitivos y proporcionar sugerencias útiles, pero nunca debe desplazar el diálogo genuino con usuarios reales, ni la creatividad estratégica y el juicio crítico del equipo.

Las historias de usuario se fundamentan en la fórmula de las **tres Cs de Ron Jeffries**. La conversación es el elemento central: el momento donde el equipo, el propietario del producto y los usuarios construyen un entendimiento compartido. Una IA puede generar tarjetas y sugerir criterios, pero no puede reemplazar la conversación humana que da vida a las historias.

Usos apropiados de la IA

La inteligencia artificial puede aportar valor en fases específicas del trabajo con historias de usuario, siempre que se utilice como complemento, nunca como reemplazo, del trabajo colaborativo del equipo.

Además es fundamental tener en cuenta la información que se introduce en herramientas de IA. Es mejor evitar introducir datos sensibles: datos personales, secretos comerciales, incidencias con datos de clientes, credenciales, detalles contractuales. Cuando sea posible, utiliza entornos corporativos aprobados y anonimiza o minimiza los datos antes de analizarlos.

Generación de borradores iniciales

Una de las aplicaciones más evidentes de la IA es la generación rápida de borradores de historias de usuario a partir de descripciones breves o necesidades identificadas.

Herramientas (ejemplos)

Existen modelos generalistas, asistentes integrados en suites de gestión y extensiones específicas para backlog. La disponibilidad concreta cambia con rapidez; lo importante es validar *outputs* y gobernar el uso.

Aplicación práctica

Imaginemos que durante una sesión de descubrimiento con usuarios identificamos la siguiente necesidad: «Los clientes quieren poder recuperar su contraseña cuando la olvidan». Podemos usar la IA para generar rápidamente variantes de historias de usuario:

Prompt a la IA

Basándote en esta necesidad: 'Los clientes quieren recuperar su contraseña', genera 3 variantes de historia de usuario siguiendo el formato 'Como [rol], quiero [objetivo], para [beneficio]'.

Posible respuesta de la IA

Variante 1:
Como cliente registrado,
quiero poder solicitar el reseteo de mi contraseña mediante mi email,
para poder recuperar el acceso a mi cuenta si la olvido.

Variante 2:
Como usuario de la plataforma,
quiero recibir un enlace temporal de recuperación de contraseña,
para poder crear una nueva contraseña de forma segura.

Variante 3:
Como cliente,
quiero poder verificar mi identidad mediante preguntas de seguridad,
para poder restablecer mi contraseña sin depender del email.

Estas variantes pueden servir como **punto de partida para la conversación** con el equipo y el propietario del producto. Ahora bien, requieren **validación humana obligatoria.** No se deberían implementar historias creadas exclusivamente por IA sin este paso de validación.

Refinamiento y mejora de historias existentes

La IA puede actuar como asistente de calidad, ayudando a identificar problemas en historias ya escritas y sugiriendo mejoras.

La IA puede ayudar a:

- identificar ambigüedades en la redacción que podrían generar malentendidos,
- sugerir criterios de aceptación complementarios basándose en la descripción de la historia,
- detectar inconsistencias entre historias relacionadas del backlog,
- proponer divisiones de historias grandes aplicando estrategias,

- analizar conformidad con INVEST (el modelo de calidad de historias de usuario).

Ejemplo práctico

> **Como** usuario **quiero** gestionar productos.

Esta historia es demasiado grande, ambigua y no especifica el beneficio. Podemos pedir ayuda a la IA:

> **Prompt**
>
> *Esta historia de usuario parece demasiado grande y ambigua: 'Como usuario, quiero gestionar productos'. Sugiere cómo dividirla siguiendo el patrón CRUD (crear, leer, actualizar, eliminar) y añade beneficios específicos.*
>
> **Posible respuesta de la IA**
>
> Historia 1:
>
> Como administrador del catálogo,
> quiero poder dar de alta nuevos productos,
> para poder ampliar la oferta disponible para los clientes.
>
> Historia 2:
>
> Como administrador del catálogo,
> quiero poder consultar el listado completo de productos,
> para poder revisar qué artículos están activos en el sistema.
>
> Historia 3:
>
> Como administrador del catálogo,
> quiero poder actualizar datos de productos existentes,
> para poder mantener la información actualizada (precios, descripciones, stock).
>
> Historia 4:
>
> Como administrador del catálogo,
> quiero poder dar de baja productos descontinuados,
> para poder mantener el catálogo limpio y evitar confusiones.

El equipo puede entonces revisar estas sugerencias, ajustar el rol («administrador del catálogo» en lugar de «usuario» genérico), priorizar cuáles son más valiosas para el negocio, y refinar los beneficios según el contexto específico de su organización.

Análisis y organización del backlog

Cuando un backlog crece considerablemente, la IA puede asistir en tareas de análisis y detección de patrones que serían laboriosas manualmente.

La IA puede:

- identificar historias duplicadas o muy similares que podrían consolidarse,
- detectar *gaps* en la cobertura funcional («tenemos historias para alta y consulta, pero no para actualización»),
- analizar dependencias implícitas entre historias (ej. una historia requiere que otra esté implementada primero),
- sugerir agrupaciones lógicas por tema, módulo o flujo de usuario,
- proponer priorización inicial basándose en patrones de valor de negocio.

Ejemplo

En un backlog con 150 historias, podemos solicitar a la IA:

> *Analiza este backlog y agrupa las historias por módulos funcionales:*
> *autenticación, catálogo de productos, carrito de compra, pagos, y administración.*

La IA puede procesar el listado completo y proporcionar una categorización inicial que el equipo luego revisa y ajusta. Esto ahorra tiempo en sesiones de backlog refinement.

Extracción de información de múltiples fuentes

Una de las capacidades de la IA es el procesamiento de lenguaje natural (NLP) aplicado a grandes volúmenes de texto no estructurado.

La IA puede procesar:

- **Transcripciones de entrevistas con usuarios:** identificar patrones, necesidades recurrentes y *pain points* mencionados.
- ***Tickets* de soporte al cliente:** analizar quejas y solicitudes frecuentes que pueden convertirse en historias.
- ***Feedback* en encuestas o NPS:** extraer *insights* cualitativos sobre qué funcionalidades valoran los usuarios.
- ***Reviews* de productos en *marketplaces*:** entender qué aspectos del producto son más apreciados o criticados.
- **Documentación técnica dispersa:** consolidar información sobre sistemas legacy o integraciones necesarias.

Ejemplo

Después de realizar 20 entrevistas con usuarios, tenemos 20 documentos de transcripción. Podemos usar IA para:

> *Analiza estas 20 transcripciones de entrevistas con usuarios y extrae las 10 necesidades o problemas mencionados con mayor frecuencia. Para cada uno, sugiere una historia de usuario preliminar.*

Este análisis proporciona un punto de partida basado en datos reales, que luego el propietario del producto valida y prioriza según la estrategia del negocio.

Riesgos y limitaciones críticas de la IA

A pesar de las ventajas aparentes, el uso de IA en historias de usuario conlleva riesgos serios que pueden comprometer la calidad del producto y la efectividad del proceso ágil. Es esencial que los equipos sean conscientes de estas limitaciones para evitar errores costosos.

La IA no comprende usuarios reales

La inteligencia artificial genera texto basándose en patrones estadísticos aprendidos de vastos corpus de datos durante su entrenamiento. No tiene experiencia directa con los usuarios de tu producto, no comprende su contexto específico, ni puede empatizar genuinamente con sus frustraciones y necesidades.

Cuando la IA genera una historia de usuario, está produciendo texto que suena bien porque ha visto miles de ejemplos similares en su entrenamiento, pero no tiene comprensión real del problema que intenta resolver ni del valor que aportaría a tu usuario específico.

Puedes terminar con un backlog lleno de historias que suenan profesionales, siguen perfectamente el formato «Como... quiero... para...», y cumplen con INVEST, pero que no resuelven problemas reales de tus usuarios. Esto lleva a **construir funcionalidades que nadie usa o valora.**

La IA no puede reemplazar «salir del edificio» y hablar con usuarios reales. La IA puede ayudar a procesar y organizar información, pero nunca a generarla desde cero sin un *input* humano real.

Historias «perfectas» que inhiben la conversación

Las historias de usuario generadas por IA pueden parecer demasiado pulidas y completas, creando una falsa sensación de que ya no hay nada que discutir o refinar.

Esto contradice uno de los principios fundamentales que vimos al inicio: las historias de usuario son *placeholders* para conversaciones, no documentos finales cerrados.

Cuando una historia llega al equipo con redacción impecable, criterios de aceptación detallados y estructura perfecta (todo generado por IA), el equipo puede caer en la trampa de asumir que ya está «terminada» y simplemente proceder a implementarla sin el diálogo necesario.

Recordemos las tres Cs:

- **Card (tarjeta):** un recordatorio breve.
- **Conversation (conversación):** el elemento central donde se construye el entendimiento.
- **Confirmation (confirmación):** validación de que se entendió correctamente.

Si la IA genera una «card» tan completa que parece no necesitar «conversation», estamos pervirtiendo el modelo.

Pérdida de contexto organizacional

La IA opera sin conocimiento de las particularidades de tu organización, lo cual limita severamente su utilidad para decisiones estratégicas.

La IA no conoce:

- la estrategia específica de tu empresa y sus objetivos a corto/medio plazo,
- las limitaciones técnicas de tu sistema *legacy* o arquitectura actual,
- el conocimiento tácito acumulado por tu equipo a través de iteraciones previas,
- las restricciones regulatorias específicas de tu industria (GDPR, PCI-DSS, normativas sectoriales),
- la cultura organizacional y dinámicas internas que influyen en la viabilidad de ciertas soluciones,
- el historial de decisiones pasadas y por qué se tomaron ciertos caminos.

Ejemplo

La IA puede sugerir: «Como usuario, quiero integración con redes sociales para compartir mi perfil». Esto suena razonable y es una funcionalidad común. Sin embargo, tu equipo sabe que:

- la estrategia de privacidad de tu empresa prohíbe integraciones con redes sociales de terceros,
- ya se intentó en el pasado y generó problemas de soporte,
- vuestros usuarios objetivo (sector B2B corporativo) no valoran esta funcionalidad.

Esta historia generada por IA es técnicamente correcta pero estratégicamente equivocada para tu contexto.

Historias «correctas» técnicamente pero irrelevantes estratégicamente

La IA puede generar historias que siguen todas las buenas prácticas (INVEST, formato correcto, criterios de aceptación), pero que no aportan valor real al negocio o que priorizan funcionalidades de baja importancia.

Ejemplo

Le pides a la IA «genera 20 historias de usuario para una aplicación de banca móvil». La IA generará historias que suenan lógicas basándose en aplicaciones bancarias que conoce de su entrenamiento. Puede incluir cosas como:

- «Como usuario, quiero poder cambiar el tema de color de la app».
- «Como usuario, quiero ver animaciones al realizar transacciones».
- «Como usuario, quiero poder poner apodos a mis cuentas».

Estas historias son válidas desde un punto de vista técnico, pero ¿son prioritarias para tu banco en este momento? Tal vez tu estrategia se centra en captar clientes jóvenes que valoran la inversión en criptomonedas, o en mejorar la experiencia de personas mayores que luchan con la complejidad de la app.

Las historias generadas por IA no reflejan estas prioridades estratégicas a menos que las especifiques muy detalladamente en el prompt (y aún así, la IA interpretará según sus sesgos de entrenamiento).

Por tanto, el equipo puede perder tiempo valioso implementando funcionalidades que, aunque correctas, no mueven la aguja en los objetivos de negocio.

Sobreconfianza en *outputs* de IA

Un riesgo psicológico importante es la sobreconfianza que pueden generar las respuestas de la IA. Como las respuestas suelen estar bien redactadas y presentadas con seguridad, es fácil asumir que son correctas o completas.

Es probable que se activen los siguientes sesgos cognitivos:

- ***Authority bias:*** tendemos a confiar en fuentes que parecen «de autoridad».
- ***Automation bias:*** confiamos excesivamente en sistemas automatizados, asumiendo que son más precisos que el juicio humano.

La realidad es que las IA generativas pueden generar información incorrecta con total confianza (el fenómeno conocido como «alucinaciones»). En el contexto de historias de usuario, esto puede traducirse en:

- sugerir integraciones con APIs que no existen,
- proponer funcionalidades técnicamente inviables,

- inventar «mejores prácticas» que no están respaldadas por la industria,
- crear dependencias entre historias que no tienen sentido.

Es crítico mantener una postura de escepticismo saludable y validar siempre los outputs de IA con conocimiento experto del equipo.

Atrofia de habilidades profesionales

El uso excesivo de IA como «primera línea» para generar historias puede atrofiar las habilidades del equipo. Los profesionales ágiles deben pensar críticamente primero sobre el problema, después pueden usar Inteligencia Artificial como herramienta de refinamiento.

Lo recomendable es que el equipo escriba primero un borrador sin utilizar Inteligencia Artificial. Después puede recurrir a ella para revisar, sugerir mejoras, identificar gaps; la decisión final sobre qué incorporar también debe ser del equipo.

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 el marketing. Las micro-campañas 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
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, con frecuencia se concretan antes en capacidades y criterios verificables; las de marketing se centran en outcomes (impacto en comportamiento/ percepción). 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, más que de funcionalidades.

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. El ecosistema de herramientas para gestionar historias de usuario ha crecido significativamente, ofreciendo opciones para diferentes necesidades y contextos de trabajo.

Principales categorías de herramientas

Suite Atlassian

- **JIRA:** herramienta robusta y altamente configurable, líder del mercado para gestión de proyectos ágiles.
- **Trello:** interfaz visual basada en tableros Kanban, más simple e intuitiva.
- **Confluence:** complementa JIRA/Trello con documentación y wikis.

Herramientas modernas especializadas

- **Linear:** diseñada específicamente para equipos de desarrollo de software, interfaz rápida y minimalista.
- **ClickUp:** plataforma todo-en-uno con alta personalización.
- **Notion:** combina gestión de proyectos con documentación colaborativa.
- **Monday.com:** enfocada en visualización y automatización de *workflows*.

Herramientas especializadas

- **Azure DevOps:** integración completa con ecosistema Microsoft.
- **Rally (Broadcom):** para organizaciones grandes con múltiples equipos.
- **Taiga:** plataforma *open-source* para gestión ágil.

Herramientas de colaboración visual

- **Miro:** pizarra digital colaborativa, ideal para story mapping y *workshops* remotos.
- **Mural:** similar a Miro, enfocada en facilitación visual.
- **Excalidraw:** pizarra digital colaborativa de código abierto.
- **FigJam:** de Figma, integrada con diseño de producto.

Funcionalidades con inteligencia artificial

Muchas de estas herramientas están incorporando capacidades de inteligencia artificial para asistir en la redacción, refinamiento y gestión de historias de usuario. Por ejemplo:

- **Atlassian Intelligence (Jira):** puede sugerir criterios de aceptación, dividir historias grandes automáticamente, generar resúmenes de progreso, y automatizar respuestas a preguntas sobre el proyecto.
- **Linear AI:** asiste en la redacción de descripciones, sugiere etiquetas y prioridades.
- **ClickUp AI:** genera borradores de historias, resume *threads* de comentarios.
- **Notion AI:** ayuda con documentación complementaria y resúmenes.

Estas funcionalidades deben usarse como herramientas de apoyo, no como reemplazo del juicio y colaboración humana. La IA puede acelerar tareas mecánicas (redacción inicial, formateo, sugerencias), pero la conversación y el refinamiento colaborativo siguen siendo esenciales.

Ejemplos de uso

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 19: 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):

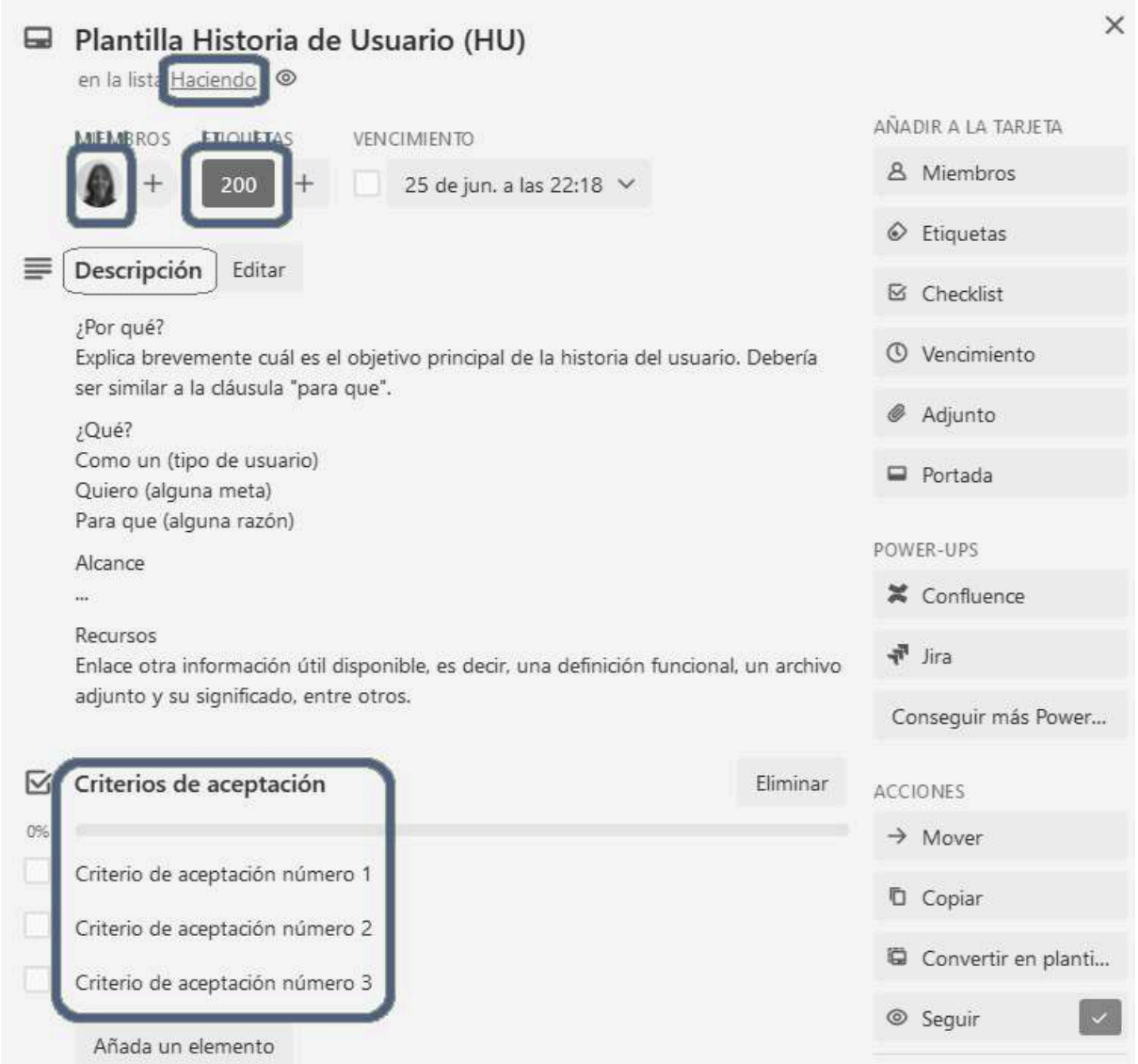


Ilustración 20: ejemplo de una historia de usuario en Trello

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

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

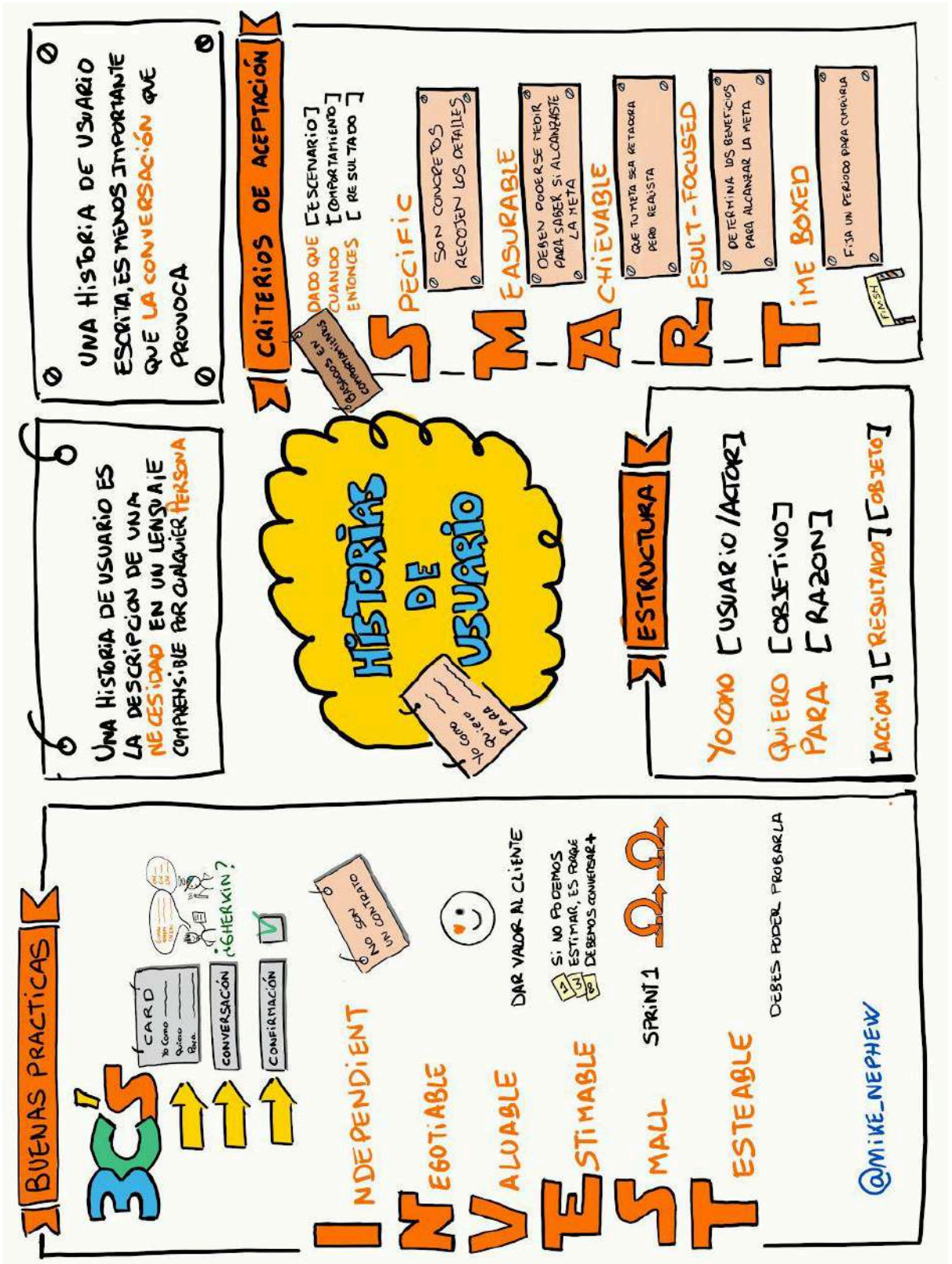
También se puede enlazar y complementar la historia de usuario con herramientas alternativas como JIRA o Confluence. Como toda herramienta digital, se pueden potenciar además con *plugins* o *addons*, que habilitan más funcionalidades para nuestros equipos.

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

Aportan informes y gráficos útiles para el seguimiento de épicas, historias y releases; elementos de larga duración que dan valor al negocio. Se pueden observar desde su entrada en el backlog hasta su despliegue, y analizar métricas de uso, calidad y resultados.

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.

Resumen historias de usuario



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.

Bibliografía

Adzic, Gojko

- (2009). *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*, Neuri Limited.
- (2011). *Specification by Example: How Successful Teams Deliver the Right Software*, Manning Publications.
- (2012). *Impact Mapping: Making a Big Impact with Software Products and Projects*, Provoking Thoughts.

Beck, Kent (1999). *Extreme Programming Explained: Embrace Change*, Addison-Wesley.

Bland, David J.; Osterwalder, Alexander (2019). *Testing Business Ideas: A Field Guide for Rapid Experimentation*, Wiley.

Clegg, Dai; Barker, Richard (1994). *Case Method Fast-Track: A RAD Approach*, Addison-Wesley.

Cockburn, Alistair (2001). *Agile Software Development*, Addison-Wesley.

Cohn, Mike

- (2004). *User Stories Applied: For Agile Software Development*, Addison-Wesley.
- (2009). *Succeeding with Agile: Software Development Using Scrum*, Addison-Wesley.
- (2024). «SPIDR: Five Simple but Powerful Ways to Split User Stories», Mountain Goat Software:

  https://www.mountaingoatsoftware.com/blog/five-simple-but-powerful-ways-to-split-user-stories

Cunningham, Ward (1992). «The WyCash Portfolio Management System», OOPSLA '92.

Ewel, Jim (2011). «Agile Marketing» [vídeo], YouTube: https://www.youtube.com/watch?v=tllAqvvvM_A

Fowler, Martin (2009). «TechnicalDebt», martinfowler.com: https://martinfowler.com/bliki/TechnicalDebt.html

Garzás, Javier (2014). *Agilidad y Lean: gestionando los proyectos y negocios del s. XXI*, Miriadax.

Humanizing Work (2024). «Guide to Splitting User Stories», humanizingwork.com: https://www.humanizingwork.com/the-humanizing-work-guide-to-splitting-user-stories/

Jeffries, Ron; Anderson, Ann; Hendrickson, Chet (2001). *Extreme Programming Installed*, Addison-Wesley.

Krebsbach, Heather (2016). «Know Thy Customer: Agile's Essential Guide to User Story Maps», Atlassian: https://www.atlassian.com/blog/2016/05/guide-to-agile-user-story-maps

Lawrence, Richard (2009). «Patterns for Splitting User Stories», Agile for All: https://www.agileforall.com/patterns-for-splitting-user-stories/

López, Gertrudis (2016). «Esquema con las 10 estrategias de división de historias de usuario», MindMeister: https://www.mindmeister.com/app/map/682181724?t=LjoGABkbaF

Mollick, Ethan (2024). *Co-Intelligence: Living and Working with AI*, Portfolio.

O'Reilly, Barry (2020). «How to Implement Hypothesis-Driven Development», barryoreilly.com: https://barryoreilly.com/explore/blog/how-to-implement-hypothesis-driven-development/

Patton, Jeff (2014). *User Story Mapping: Discover the Whole Story, Build the Right Product*, O'Reilly Media.

Pichler, Roman (2010). *Agile Product Management with Scrum: Creating Products That Customers Love*, Addison-Wesley.

Rehkopf, Max (s. f.). «User Stories with Examples and a Template», Atlassian Agile Coach: https://www.atlassian.com/agile/project-management/user-stories

Reinertsen, Donald (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*, Celeritas Publishing.

Rubin, Kenneth S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*, Addison-Wesley.

Savoia, Alberto (2011). *Pretotype It: Make Sure You Are Building the Right It Before You Build It Right*.

Torres, Teresa (2021). *Continuous Discovery Habits: Discover Products That Create Customer Value and Business Value*, Product Talk.

Verwijs, Christiaan (2015). «10 Useful Strategies for Breaking Down Large User Stories (and a Cheatsheet)», The Liberators.

Wake, Bill (2003). «INVEST in Good Stories, and SMART Tasks», XP123: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Wallet, Thomas (s. f.). «Checklist INVEST para evaluación de historias de usuario».

Herramientas y plataformas de IA

Atlassian (2024). *Atlassian Intelligence*: https://www.atlassian.com/software/artificial-intelligence

Anthropic (2024). *Claude*: https://www.anthropic.com/claude

ClickUp (2024). *ClickUp AI*: https://clickup.com/ai

GitHub (2024). *GitHub Copilot*: https://github.com/features/copilot

Linear (2024). *Linear AI*: https://linear.app/features/ai

Notion (2024). *Notion AI*: https://www.notion.so/product/ai

OpenAI (2024). *ChatGPT y GPT-4*: https://openai.com/chatgpt

Información, recursos y comunidad profesional

Club Agile →

Skill Arena →

Academy →

BoK/Wiki →

Comunidad →

Tiras de cómic →

Acerca de la certificación Scrum Manager® →

Preguntas frecuentes sobre cursos y exámenes Scrum Manager® →

Redes sociales y profesionales

LinkedIn

Youtube

Spotify

Ivoox

Tabla de ilustraciones

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

Ilustración 2: granularidad de la pila de producto.

Ilustración 3: gráfico con los cuatro niveles de tamaño.

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

Ilustración 5: ejemplos de una historia de usuario.

Ilustración 6: ejemplo de user persona.

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

Ilustración 8: cartas de estimación con tallas de camisa.

Ilustración 9: ejemplo gherkin.

Ilustración 10: los requerimientos funcionales como restricciones.

Ilustración 11: ejemplo de tarjetas diseñadas por Braintrust.

Ilustración 12: refinamiento de la pila de producto.

Ilustración 13: esquema con las 10 estrategias de división de historias de usuario.

Ilustración 14: pila de producto plana y pila de producto, técnica user story mapping.

Ilustración 15: ejemplo de backbone de un user story mapping.

Ilustración 16: backbone e historias de usuario de un user story mapping.

Ilustración 17: backbone, historias de usuario y versiones de un user story mapping.

Ilustración 18: previsión de lanzamiento de versiones sobre gráfico de producto.

Ilustración 19: ejemplo de una historia de usuario en JIRA.

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