PLANEANDO SER ÁGIL

# Destino desconocido, la diferencia entre planear todo desde el principio a realizar planeamiento iterativo

*Douglas Adams dijo una vez: «Me encantan los plazos. Me encanta el silbido que*

*hacen al pasar volando. ¡Whoosh! Ahí va otra fecha límite».*

Esto nos pasa todo el tiempo, ponemos una fecha límite y no la cumplimos. La pregunta es, ¿por qué sucede esto? Pero lo que es más importante, ¿qué podemos hacer para evitarlo?

A esto me gusta llamarlo navegar por lo desconocido, por ejemplo, si te dijera que tienes que navegar a través de este campo de pingüinos, ya sabes, podrías mirar hacia abajo y decir: «Bueno, si puedo, puedo poner mi pie aquí y tal vez entrar y dar una vuelta».

Pero cuando llegas al centro, ¿cómo trazas un rumbo hacia el otro lado? Ah, y por cierto, los pingüinos van a seguir moviéndose a medida que tú avanzas por ellos, como ocurre con el

desarrollo de software, en el que el sistema operativo se va parcheando y remendando. Y los

paquetes se están parcheando. Y ya sabes, las cosas se mueven, se mueven, pero sabes que si te metes de puntillas en medio de esos pingüinos y miras hacia abajo, se parecerá mucho a

lo que era al principio, cuando tú, ya sabes más, estés en un punto de vista diferente.

Y esa es la idea, claro, desde este punto de vista, probablemente puedas seguir conspirando.

Y a medida que os vayáis acercando, tendréis el siguiente punto de vista en el que podréis trazar el mapa.

**Así que el mensaje aquí es que no lo decidas todo en el punto que menos sabes.** Hacemos esto todo el tiempo. Al principio del proyecto, no sabemos prácticamente nada, sabemos muy poco sobre el proyecto. Y ahí es cuando planificamos todo, ¿no?, como si pudiéramos imaginarnos lo que va a pasar hacia el final. De eso se trata la metodología ágil, de la planificación iterativa. **No decidas todo al principio, cuando menos sabes.** Lo que quieres hacer es planificar lo que sabes y, a medida que avanzas, decides que sabes más, ajustas el plan.

Por lo tanto, sus estimaciones serán más precisas de esta manera. Porque ya sabes, si alguien te pregunta qué vas a hacer dentro de tres meses. Dígales: «Bueno, puedo decirles que podemos tener una precisión del 50%, ¿verdad? Pero puedo decirles con una precisión de casi el 100% lo que voy a hacer dentro de dos semanas, bien y dos semanas después».

**Así que la idea aquí es que no intentes ser omnipotente, porque no tienes razón. No intentes planificar todo por adelantado.** Desea planificar sobre la marcha y, a medida que vaya aprendiendo más de lo que podría, añadir más al plan y obtener estimaciones cada vez mayores del lugar en el que se encuentra y del tiempo que tardará en llegar hasta allí.

# Roles ágiles y la necesidad de entrenamiento

Algunas cosas organizativas que las personas hacen cuando intentan ser ágiles, que conducen al fracaso, y una de ellas es **colocar a las personas en un nuevo puesto sin capacitarlas.**

Por lo tanto, **el gerente de producto se convierte en el propietario del producto.** Aquí hay una gran diferencia, el gerente de producto es un título de trabajo. El propietario del producto es un rol de scrum. Puede que sean la misma persona, puede que no sean la misma persona.

Luego veo que **el gerente del proyecto se convierte en el maestro de scrum** sin capacitación sobre cómo ser un maestro de scrum.

Y, por último, **el equipo de desarrollo se convierte en un equipo de scrum.** Pero solo son ingenieros de software. No hay evaluadores en el equipo. No hay personal de operaciones en el equipo. No hay analistas de negocios ni nada en el equipo. Así que analicemos un poco más por qué hacer estas cosas conduce al fracaso.

En primer lugar, el gerente de producto se convierte en propietario del producto. El gerente de producto suele ser una persona de negocios que administra el presupuesto, puede o no tener una visión de cómo van las cosas. Pero el **propietario del producto es el visionario que dirige al equipo en una serie de experimentos diseñados para lograr cualquiera que sea el objetivo del sprint.** Y estos no suelen ser los mismos conjuntos de habilidades, **hay que centrarse más en dónde llevar el producto que en cómo gestionar el presupuesto**. Además, los gerentes de producto suelen ser la persona, se centran en los aspectos operativos del negocio, ¿verdad? En lugar de ser un conducto entre las partes interesadas y el equipo, traduzca esos requisitos empresariales en los objetivos técnicos que el equipo debe alcanzar. Así que ten cuidado. Algunos gerentes de productos de proyectos son buenos para ser propietarios de productos, otros no solo entienden por qué uno de ellos es un puesto de trabajo. El otro es un rol en scrum, y puede que no sea la misma persona.

Lo siguiente es que el director del proyecto se convierta en el scrum master. Es clave entender cómo trabajan los gerentes de proyectos. Nuestro director de proyectos es un administrador de tareas, hace que todos sigan un plan. Sabes, cuando enseño Agile, muy a menudo tengo a alguien al final de la sala que me está viendo planificar nuestros sprints y crear nuestro tablero Kanban y dice: «Disculpe, ¿cómo puedo saber si le he asignado demasiado trabajo a alguien?» Y la respuesta es que no se asigna trabajo a nadie. Por eso no hay forma de saberlo, ¿verdad? Son un equipo autogestionado que se asignan el trabajo a sí mismos. Por lo tanto, ser **un maestro de scrum es una función muy diferente, en la que eres un entrenador que mantiene al equipo concentrado en el sprint actual y en la autogestión.**

La otra gran diferencia entre un gestor de proyectos y un scrum master es que cuando el gestor de proyectos siente que tienes un impedimento, documenta el riesgo en su pequeña hoja de cálculo, ¿verdad? Esto es lo que hacen, documentan los riesgos. Así que si tengo un problema, tengo algún bloqueo, el director del proyecto va a decir: «¿Qué vas a hacer para desbloquearte?» A diferencia de un **scrum master que elimina los impedimentos y protege al equipo de las interrupciones.** Por lo tanto, **si el equipo dice: «Tengo un impedimento», el maestro de scrum debería decir: «Deja que yo me encargue de eso , tú ponte a trabajar en algo más productivo. Y déjame quitarte eso del camino».** Los directores de proyecto casi nunca dirán eso. Por lo tanto, realmente se necesita un tipo de persona diferente para asumir el papel de scrum master que el que se les enseñó a los gerentes de proyectos a gestionar los proyectos para planificar y documentar los riesgos.

Y, por último, es el equipo de desarrollo contra el equipo de scrum. Quiero decir, un equipo de desarrollo normalmente está formado por ingenieros de software, ¿verdad? Solo desarrolladores. Pero **un equipo de scrum es un equipo multifuncional. Incluye a desarrolladores y evaluadores, personal de seguridad y analistas y operaciones empresariales, a cualquier persona que necesitemos para crear un incremento.** Por eso es muy importante que se reorganice y cree equipos de desarrollo multifuncionales, no solo un grupo de ingenieros de software.

*cita de Bill Cantor: «Hasta que los líderes empresariales no acepten la idea de que ya no administran proyectos con funciones, plazos y costos fijos como lo hicieron con Waterfall, tendrán dificultades para usar Agile tal como fue diseñado para usarse».*

Por lo tanto, los roles han cambiado. No se puede poner a las personas en un nuevo rol sin capacitarlas, de lo contrario, un gerente de proyecto se esforzará por convertir un tablero Kanban en un diagrama de Gantt, porque eso es todo lo que sabe, es administrar proyectos con diagramas de Gantt. Es muy importante que esta mentalidad provenga de la alta dirección, ya que **la alta dirección no puede pedirte que predigas cosas en un futuro lejano. En lugar de preguntarte qué vas a hacer antes de fin de año, deberían decir qué vas a hacer en las próximas dos semanas. ¿Cómo vas a deleitar a mis clientes al final del próximo sprint?**

# ​Kanban y herramientas de planificación ágil

Existen muchas herramientas de planificación ágil. Pero una cosa que debes entender es

que una herramienta no te hará ágil, realmente tienes que tener una mentalidad ágil para ser ágil. **Las herramientas respaldarán su proceso ágil, pero primero debe tener el proceso**. He visto a muchas personas que no han recibido capacitación tratar de hacer que un tablero Kanban parezca un diagrama de Gantt. Son dos cosas diferentes, dos maneras diferentes de gestionar proyectos. Así que es muy importante que entiendas esto. Y hay muchísimas herramientas de planificación ágil. La mayoría hace lo mismo, pero algunos son un poco más complicados que otros.

Vamos a usar ZenHub para este curso. Entonces, ¿qué es ZenHub? Bueno, es un complemento para GitHub, esto permite que los desarrolladores permanezcan en GitHub y usan este complemento de ZenHub. Lo que hace ZenHub es que te permite agregar herramientas de administración de proyectos, un tablero Kanban, para administrar tus proyectos mientras permaneces en GitHub. La otra es que ZenHub es personalizable. comienza con un conjunto de columnas, llamadas canalizaciones, en tu tablero Kanban, pero puedes personalizarlas para que se adapten a lo que quieras. Puedes hacerlo tan complejo como simple. Preferiblemente simples.

Entonces, ¿por qué lo usamos? Bueno, una de las principales razones es porque usa problemas de GitHub. No es otra herramienta con otra cosa que tenga que editar. Uno de los

problemas que encuentro es que si utilizas demasiadas herramientas y un desarrollador tiene que ir a otro lugar para actualizar el estado, ese estado estará desactualizado el 100% del tiempo. Porque tan pronto como se actualicen, pasarán a hacer otra cosa, y eso no se reflejará en el estado. En cambio ZenHub Proporciona una manera fácil de entender lo que está pasando, mantiene una versión actualizada de la verdad, los desarrolladores trabajan en historias y en GitHub todo el día. Los problemas en GitHub, que son historias. Esos problemas son las cosas que abren y cierran, y no tienen que ir a otro lugar para actualizar el estado. El estado se mantiene directamente en su herramienta favorita, GitHub. Por lo tanto, permite a los desarrolladores ir a un solo lugar para hacer las cosas y no tener que actualizar varias herramientas.

¿Qué es un tablero Kanban? Hemos estado hablando de los tableros Kanban aquí. En pocas palabras, **hay cosas que debes hacer, cosas que estás haciendo y cosas que**

**ya has hecho.** No tiene por qué ser más complicado que eso. Además, **mueves las cosas por el tablero Kanban para mostrar el progreso que estás haciendo**, por lo que es una forma muy visual de ver exactamente dónde te encuentras en cualquier momento.

Así que hablemos de las canalizaciones o columnas predeterminadas que están en ZenHub.

Empieza con nuevos temas. **Los nuevos números son como tu bandeja de entrada. Cuando alguien abre un tema. El valor predeterminado es**

**ir a la columna de nuevos problemas.** Por eso no me gusta dejarlo ahí demasiado tiempo. Cuando refino mi cartera de pedidos y trabajo en mi tablero Kanban, normalmente hago primero una clasificación de las nuevas ediciones y digo **que es algo que acaba de llegar, ¿a dónde debería ir? Y lo paso a otro canal, o lo rechazo, pero no lo guardo en el nuevo número para que cuando llegue otro, sepa que debe ser muy nuevo porque lo acabo de limpiar como tu bandeja de entrada de correos electrónicos.**

**Luego está la nevera.** Esto es exclusivo de ZenHub, pero me gusta un poco. La nevera es de almacenamiento en frío. **Es donde pones las cosas que van a estar, en las que vas a trabajar a largo plazo.** Así que si no voy a conseguir algo durante un tiempo, lo tiraré a la

nevera. Así que no lo olvido. Pero tampoco estoy trabajando activamente en mis otros proyectos para que no se acumulen demasiado.

Después de la nevera está la **cartera de productos. La**

**cartera de productos es todo lo que siempre deseas hacer para siempre en tu producto y que aún no has hecho.**Correcto. No contiene las cosas que estás haciendo. **Contiene todas las cosas que no has incluido en un sprint y que quieres resolver en el futuro.** Y una vez más, para que la cartera de productos no sea demasiado desordenada, cosas que tienen un futuro a largo plazo, me gusta pasarme a la nevera.

Esto nos lleva al atraso de los sprints.

**La cartera de sprints es lo que vamos a hacer en las próximas dos semanas.** Así que he tomado cosas de la cartera de productos y las he trasladado a la cartera de sprints para

hacer un plan de sprints. Ahora los desarrolladores no tienen que preocuparse por ninguno de estos otros procesos, solo se concentran en la acumulación de sprints, porque ese es el trabajo que haremos en el próximo sprint.

**Luego, una vez que empezamos a trabajar en las cosas, las pasamos a estar en progreso.** Así que sé que cuando veo esas cartas, esas historias en la columna en progreso, sé que alguien está trabajando en ellas mientras se las asigna a sí mismo, aparece su pequeño avatar. Así puedo ver exactamente quién está trabajando en qué y en qué se está trabajando mientras hablamos.

Una vez que se completa el trabajo, **el desarrollador suele crear una solicitud de extracción para devolver su trabajo a la rama base.** Y para que se incluyan en el control de calidad de las revisiones, también puedes configurar GitHub y ZenHub para que creen automáticamente solicitudes de cambios en esa columna de control de calidad de las reseñas. Y para que los desarrolladores sepan, cuando vean que algo aparece allí, que tienen que ir a verlo y ayudar a otro desarrollador a revisar su historia y asegurarse de que lo entienden, cumple con los criterios necesarios para fusionarlo con el resto del código.

**Y, por último, está la columna «Listo». Cuando hayamos terminado, volvemos a fusionar nuestro código y, a continuación, movemos la historia a la columna Listo. Hecho significa que el desarrollador ha terminado. No significa que el**

**propietario del producto lo haya aceptado.** Eso es algo que ocurre durante la revisión del sprint. Pero sí significa que el desarrollador ha terminado con esa historia. Vuelven al trabajo pendiente del sprint, ponen la historia en marcha, se la asignan a sí mismos y siguen trabajando. Así que el flujo va de izquierda a derecha. Las nuevas historias aparecen por la izquierda y un incremento hecho termina saliendo por la derecha.

# CREANDO BUENAS HISTORIAS DE USUARIO

¿Qué es una historia de usuario?

En pocas palabras, **una historia de usuario representa una parte del valor empresarial que el equipo puede ofrecer en poco tiempo.** Solíamos llamarlos requisitos, pero las historias de usuario son mucho más. Por lo general, el requisito es: necesito esto, necesito aquello, pero **una historia de usuario incluye: ¿Para quién es? ¿Qué es lo que necesitan? Pero lo que es más importante, ¿por qué lo necesitan?** ¿Cuál es el valor empresarial que obtienen al obtener esta función o función?

Por lo tanto, las historias deben contener una buena descripción del valor empresarial: qué es. ¿Quién lo necesita? ¿Cuál es el valor que obtenemos de esa historia? Luego, me gusta **tener cualquier suposición y cualquier detalle documentado en la historia.** A veces, sabes que hay suposiciones, estás haciendo una historia y necesita algún tipo de persistencia.Y sabes que vas a usar una base de datos relacional, ponla ahí para darle al desarrollador una pista: «Oye, no busques bases de datos NoSQL, hemos decidido que vamos a usar una base de datos relacional. Así que cualquier cosa que sepas, cualquier suposición, aprovisionaremos algo en la nube, lo pondremos en la historia del usuario para que el desarrollador sepa cuáles son tus intenciones. Y lo más importante, **necesitas la definición de terminado, algunos criterios de aceptación. ¿Cómo puedo saber si esta historia está completa? ¿Cuándo esté hecho?** No querrás ver en la reseña del sprint que el propietario del producto dice: «Oye, eso no es lo que quería». Quieres documentar la definición de hecho para decir: «Bueno, esta es la definición en la que estuvimos de acuerdo. Esto es lo que hace la historia, y si no es lo que quieres, podemos escribir otra historia, pero hicimos lo que dijeron que debía hacer».

Entonces, la descripción de la historia, ¿qué pones en esa descripción? Una vez más, quieres documentar a una persona, solicitar alguna funcionalidad para conseguir algún objetivo o lograr algún objetivo, obtener algún valor empresarial. Por lo tanto, la forma en que lo hacemos es con este problema de sintaxis como rol. **¿Cuál es el papel? ¿Es el director de marketing? ¿Son los clientes? ¿El administrador del sistema?** ¿Para quién es esto? Y luego necesito algo de funcionalidad. ¿Qué es lo que necesitan? Este es el meollo de la historia del usuario, pero es muy importante, para obtener algún beneficio empresarial. ¿Cuál es el beneficio de ofrecer esa función? Cuando volvamos y prioricemos el trabajo atrasado, consideraremos el valor empresarial como uno de los criterios para preguntarnos: «¿Debería tener un volumen elevado de atrasos o uno más bajo?» **Así que es muy, muy importante, no solo tener lo que es, sino ¿cuál es el valor que obtenemos de ello?**

Luego están las suposiciones y los detalles. Así que cualquier cosa que sepas, claro, no tiene que ser en gran detalle. Pero cualquier cosa que sepas, como dije, ya sabes, si sabes que vas a usar una base de datos, algo de persistencia, ya sabes, simplemente ponla ahí. Cualquier cosa de la que hayas hablado mientras escribías la historia, ponla ahí. **Lo que estás intentando hacer es ayudar al desarrollador a entender lo que hay que hacer;** cualquier cosa que no fuera obvia para completar esta historia. Y luego están los **criterios de aceptación**, la definición de terminado. Esto es de una importancia crítica y crítica. Es muy importante entender qué hace que esta historia de usuario esté terminada, completa o terminada.

**Y para eso utilizamos una sintaxis llamada Gherkin,** que lleva el nombre del pepinillo. Además, Gherkin es una sintaxis que facilita tanto a las partes interesadas como a los clientes y a los desarrolladores la descripción de la definición de terminado. Describir el comportamiento del sistema. Y es más o menos así.

*Dadas algunas condiciones previas,* esto prepara la historia, dado que tengo cinco artículos en mi carrito de compras, Dado que ya he iniciado sesión en el sistema, sea lo que sea, se establece la historia. *¿Y entonces cuál es la acción que ocurre?* ¿Qué es lo que lo desencadena cuando esto ocurre, cuando agrego otro artículo a mi carrito o elimino un artículo de mi carrito? Y finalmente, *¿cuál es el resultado comprobable de eso?*Entonces debería ver esto, no debería ver aquello, debería tener seis artículos en mi carrito o debería ver cuatro artículos en mi carrito, ¿verdad? Hace que sea muy fácil de entender para todos. Dado que este es el caso base, cuando esto suceda, debería tener un resultado medible. Y si mido ese resultado, la historia está acabada. Así que veamos un ejemplo de historia.

***Como gerente de marketing,*** ahora sabemos cuál es su función. Esto no es para el cliente. Esto no es para el administrador del sistema. Esto es para el director de marketing, así que sabemos quién se va a beneficiar de ello. ***Necesito una lista de los nombres y correos electrónicos de los clientes.*** Ahora que sabemos cuál es la función, tienen que enumerar a

los clientes con sus correos electrónicos y, luego, el valor empresarial, ***para que pueda notificarles las promociones de marketing.***ahora puedo decir: «¿Qué importancia tiene notificar a los clientes sobre las promociones de marketing?» Y tal vez anteponga eso a otra historia que es menos importante. Así que ahora he capturado: ¿para quién es? ¿Qué es lo que necesito? ¿Y por qué lo necesito? ¿Qué valor obtengo de ello? Entonces me gusta hacer cualquier suposición. Estas suposiciones ayudan al desarrollador a crear el código que acompaña a la historia. ***Por lo tanto, una suposición es que mantenemos los***

***correos electrónicos de los clientes, tal vez no.*** Tal vez esta historia dependa de otra historia que realmente crea los correos electrónicos de los clientes. Quizás haya otra suposición: los clientes han optado por las promociones. Eso es bastante importante. Puede que alguien no haya pensado en eso. Oye, no podemos simplemente enviar correos electrónicos a todos nuestros clientes, tenemos que permitirles optar por recibir promociones.

Y luego tenemos los criterios de aceptación. ¿Cuál es la definición de terminado? ¿Cuál es el comportamiento que queremos que tenga esta historia? Y usaremos la sintaxis de Gherkin, ***dado que hay 100 clientes en la base de datos.*** Así que eso prepara el escenario. ***Y 90 han optado por las promociones por correo electrónico.*** Por lo tanto, esto es de vital importancia no solo para los 100, sino para los 100 que han optado por la promoción por correo electrónico, por lo que ahora los desarrolladores saben que tienen que filtrar el número de clientes de la base de datos según quién lo haya hecho.

***Cuando solicito una lista de correo electrónico de clientes,*** esa es la acción, eso es lo que ocurre, ***debería ver una lista de 90 correos electrónicos de clientes, no de 100 correos electrónicos de clientes,*** 90 correos electrónicos de clientes, porque solo 90 han optado por la promoción por correo electrónico. Como puedes ver que esto es algo que puedes regalar a una parte interesada, el gerente de marketing podría entender el comportamiento y decir: «Sí, ese es el comportamiento que quiero». El desarrollador puede leer esto y decir: «Sí, ese es el comportamiento que puedo ofrecer». Y que al final del sprint, siempre y cuando pidas clientes, solo recibas a los que se han suscrito. Y esa es la definición de hecho. Por lo tanto,

no hay discusiones sobre si la historia está hecha o no. Si tiene ese comportamiento, está hecho.

INVEST afirma ***«Las historias deben ser independientes, quiero poder clasificarlas en la lista de publicaciones pendientes, quiero poder moverlas de un lado a otro, quiero poder decir que esta es anterior a la otra».*** Ahora sé que no siempre pueden ser independientes, a veces hay dependencias, ¿verdad? En el ejemplo anterior, supusimos que había correos electrónicos de clientes en la base de datos y, si no los había, no se

trataba de una historia independiente, sino que dependía de la historia que pusiera los correos electrónicos en la base de datos. Pero, en su mayor parte, quieres intentar escribirlas para que sean independientes. ***También tienen que ser negociables***, los voy a poner en la lista de pedidos pendientes. Voy a clasificarlos más o menos. Tal vez voy a decir que deberíamos poner más valor, más funcionalidad o menos funcionalidad para negociar cuánto hay que hacer realmente. Por lo tanto, deberían ser negociables. No deberían estar demasiado

acoplados exactamente a lo que se necesita. ***Y luego tienen que ser valiosos,*** Necesito poder decir: «¿Qué tan valiosa es esta historia? ¿Qué valor obtiene el cliente de ello para

asegurarse de que realmente se trata de una historia de usuario?». Y no solo una historia de deuda técnica de que esto es algo que tengo que hacer y que el cliente nunca ve. ***Tiene que ser estimable, tengo que ser capaz de estimarlo.***¿Qué tan grande es? ¿Pequeña? ¿Medio? ¿Grande? Tiene que haber suficiente información ahí para poder decir: «Vaya, esta es una historia muy importante, teniendo en cuenta todo lo que hay aquí». Muy bien. Y no es solo una línea que parece simple, sino que se vuelve realmente difícil. Y debe ser pequeño. Quieres que sea algo en lo que alguien pueda trabajar en un sprint, ***por lo que tiene que ser más pequeño que un sprint.*** Y, por último, ***tiene que ser comprobable.*** Necesito poder comprobar si esa historia está lista. Quiero probar la definición de hecho para esa historia. Por lo tanto, las historias deben poder comprobarse.

¿Qué hay de las ideas realmente grandes? Bueno, las grandes ideas se llaman epopeyas ***(EPIC).*** Usamos epopeyas cada vez que tenemos una historia que es más grande que un solo sprint, porque la historia debe ser más pequeña que un sprint, por definición. Y cuando es más grande, se convierte en una gran idea que no podemos completar en un sprint, así que la convertimos en épica y luego añadimos historias más pequeñas que forman la epopeya a la épica. Así que una epopeya en la jerarquía es más alta que una historia. Las historias son consumidas por las epopeyas. Por lo general, cualquier historia que sea demasiado grande, la convertimos en épica, si no podemos estimarlo por sí solo, lo dividimos en cosas más pequeñas que podemos estimar. Entonces, ¿cuándo usamos una epopeya? Bueno, está claro que cuando una historia es demasiado grande para ponerla en un sprint, la convertimos en una epopeya. Por lo general, los artículos pendientes comienzan como grandes ideas, van a ser, van a ser muy grandes. y así comienzan como epopeyas. Y luego, a medida que refinamos el trabajo atrasado, hacemos que sean historias cada vez más pequeñas que podemos incluir en un sprint. Así que, para planificar los sprints, ***querrás dividir todas esas grandes historias en historias más pequeñas, dividir todas esas epopeyas en historias más pequeñas, cosas que normalmente vienen en los nuevos números, a veces son epopeyas,*** alguien ha pedido una función o una función. Suena bastante simple, pero va a llevar más de un sprint. Así que lo pones como una epopeya y, más adelante, lo desglosas en historias de usuario.

