Formulación de requisitos Agile: Mejoras (o Épicas), Historias de Usuario y Spikes

? Archivo adjunto desconocido

Contenido

Resumen

- **Versión:** v01r01
- **Fecha publicación:** 29 de Octubre de 2020

Histórico de cambios

Los cambios en la documentación de apoyo vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución.

Versión	v01r01	**Fecha publicación**	29 de Octubre de 2020
Alcance			
• Presentación del paradigma de evolución de producto en proyectos ágiles, del concepto de Backlog del Producto y sus características y finalmente de los principales artefactos de los cuales nos valdremos para el desarrollo de productos ágiles: Mejoras (o épicas), Historias de Usuario y Spikes.			

Introducción

En este documento se analiza cómo trabajar con los requisitos del producto en un entorno ágil. Se describen las principales herramientas que se usan en la STIC para este fin: mejoras, historias de usuario y spikes, así como buenas y malas prácticas en la elaboración de las historias de usuario.

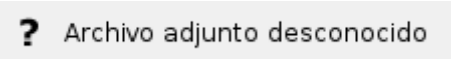
El Backlog del Producto

El *Product Backlog* (o Backlog del Producto) es una lista ordenada de todo lo que se sabe que se necesita en el producto (requisitos). Es la única fuente de requisitos para cualquier cambio que se realice en el producto. El Product Owner es responsable del Backlog del Producto, incluido su contenido, disponibilidad y priorización.

El refinamiento del Product Backlog es un proceso continuo en el que el Product Owner y los desarrollares agregan detalles y estimaciones de los elementos que vayan a ser implementados en el corto plazo (más info en Sprint Refinement).

Un correcto mantenimiento del Product Backlog implica:

- **Ordenación vertical de acuerdo a la prioridad:** Elementos de la parte superior del backlog más prioritarios que los de los niveles inferiores.
- **Definición "Justo a tiempo"**: Los elementos más prioritarios del Product Backlog que vayan a ser desarrollados en el corto plazo serán lo suficientemente detallados (y de menor tamaño), mientras que los menos prioritarios tendrán una descripción más vaga para no tener que retrabajarlos si las necesidades cambian de aquí a que se acometan.
- **Varios niveles:** Con vistas a mejorar la interpretación del backlog, este se dividirá en dos niveles fundamentales: *Épicas (o Mejoras)* e *Historias de Usuario*.



Mejoras (o Épicas): Requisitos de alto nivel

Una Épica es una gran cantidad de trabajo que no puede entregarse en una sola iteración y que por tanto puede dividirse en elementos de menor tamaño (Historias de Usuario). En el marco de trabajo ágil de la STIC las épicas toman el nombre de "Mejoras", en alusión a la entidad que las representa en JIRA.*

La definición de una Épica (o Mejora) se compone de cuatro elementos:

- ***Descripción***: una simple frase expresada en terminología de negocio que da nombre y un breve contexto de la Épica (<5 palabras).
- ***Beneficio esperado***: expresado en términos de negocio, el beneficio (medible) propuesto para el usuario final o la empresa (<15 palabras).
- ***Criterio de aceptación***: descripción que determina si la implementación es correcta y entrega el valor esperado.
- ***Estimación:*** estimación de alto nivel (carente de detalles). Para evitar confusiones con los estimados asociados a las Historias de Usuario se utilizarán los tamaños: XS, S, M, L, XL.

Ejemplo de descripción de épicas:

Épica	Beneficio esperado	Criterio de Aceptación	Estimación
Añadir métricas de uso	Conocer las costumbres de nuestros usuarios para mejorar la aplicación en consecuencia.	Todas las mediciones relevantes de cada sesión se almacenan de forma persistente.	L
...	...	...	...

**Nota: Una épica puede incluso ser un concepto de mayor nivel que la Mejora. Sería el ejemplo de una nueva funcionalidad de un aplicativo de varios módulos, cuyo impacto se extiende a todos los módulos de dicho aplicativo.*

Historias de Usuario: Requisitos de bajo nivel

Las Historias de Usuario son descripciones de una pequeña pieza de funcionalidad deseada, escritas en el lenguaje del usuario que pueden ser implementadas (desarrolladas y testeadas) en un sólo Sprint.

Son el artefacto principal utilizado para definir el comportamiento del sistema en metodologías ágiles. Son descripciones sencillas, cortas, de una funcionalidad normalmente narradas desde el punto de vista del usuario y escritas en su lenguaje. Con cada Historia de Usuario se pretende facilitar la implementación de una pequeña rebanada vertical del comportamiento del sistema.

Estructura de una HU: Las 3 C's

- **Card (tarjeta):** Una tarjeta o post-it captura la declaración de intenciones de la historia del usuario. Las fichas proporcionan una relación física entre el equipo y la historia. El tamaño de la tarjeta limita físicamente la longitud de la historia y las sugerencias prematuras sobre la especificidad del comportamiento del sistema.
- **Conversation (conversación):** Una "conversación" que tiene lugar en diferentes momentos y lugares durante un proyecto entre las diversas personas interesadas por una característica determinada de un producto de software: clientes, usuarios, desarrolladores, testers; esta conversación es en gran parte verbal, pero la mayoría de las veces se complementa con documentación.
- **Confirmation (confirmación):** La “confirmación” final entre los interesados en torno a los que giraba la conversación, finalmente, cuanto más formal mejor, de que se han alcanzado los objetivos.

Narrativa de una HU

El formato en el que se redactará seguirá el siguiente patrón:

Como <usuario: ¿Para quién vamos a construir esto?>

- Lo más específico posible.
- Los desarrolladores NO son un usuario.

Quiero <acción: ¿Qué vamos a construir?>

- El comportamiento del sistema debe ser escrito como una acción.
- Voz activa, no pasiva.
- El "sistema" o pre-requisitos de partida están implícitos y, por tanto, no se describen en detalle en la HU.

Para <valor de negocio: ¿Por qué vamos a construirlo?>

- Varias HUs pueden compartir el mismo objeto o valor de negocio.
- El beneficio puede ser para otros usuarios o clientes, no solo para el usuario de la HU.

Criterios de aceptación de una HU

Las historias de usuario registradas en JIRA incluirán sus criterios de aceptación en la propia historia de usuario de JIRA. Estos son fundamentales para validar que la historia de usuario cumple con las necesidades de negocio, además de ayudar a los desarrolladores a entender el funcionamiento del producto y servir como guía durante la implementación de la funcionalidad.

Para el caso de historias funcionales, los criterios de aceptación se redactarán, además de en lenguaje natural si así lo hace el Product Owner, utilizando Gherkin, lenguaje que define la estructura y una sintaxis básica para la descripción de las pruebas.

Nº (de escenario)	Título (del CA)	Contexto	Evento	Resultado/Comportamiento
1	Título	Dado...	Cuando...	Entonces...

Criterios INVEST para escribir buenas HUs

- **Independiente:** tanto como se pueda, debe evitarse la dependencia entre HUs así como de otros resultados /elementos externos. Las dependencias entre historias llevan a problemas de priorización y planificación.
- **Negociable:** una HU no es un contrato, es una invitación a una conversación entre el negocio y los desarrolladores. En la historia inicialmente se captura el valor de negocio y el resultado final (la parte del Quiero...) se consigue a través de una negociación colaborativa entre el cliente (o product Owner / Proxy Product Owner), el desarrollador y el tester.
- **Valor:** en la mayor medida posible, toda HU debe aportar un valor de negocio al cliente o usuarios finales ya que, de otra manera, sería difícil para Product Owners planificar el trabajo a implementar en cada iteración.
- **Estimable:** Es importante que los desarrolladores puedan estimar (o al menos adivinar) el tamaño de una historia o la cantidad de tiempo que tomará convertir una historia en código funcional y testeado. Hay tres razones principales que impiden la estimación de una HU:

  1) La HU es demasiado grande.

  2) Falta de conocimiento del dominio; lo cual puede resolverse con una estrecha colaboración entre Product Owner y el equipo.

  3) Falta de conocimiento técnico; lo cual puede resolverse mediante un Spike (ver más abajo).

- **Pequeña (Small):** Lo suficiente pequeñas como para que puedan desarrollarse y testearse durante un Sprint, pero lo suficientemente grandes como para que sean independientes.
- **Testeable:** Deben estar escritas de manera que si los Test son exitosos, puede considerarse que la HU se ha desarrollado correctamente. Las HUs no testeables suelen advertirse cuando entre sus criterios de aceptación se incluyen Requisitos No Funcionales imprecisos (p.ej. "un usuario debe encontrar el software fácil de usar" o "un usuario nunca debe esperar mucho para que aparezca una pantalla").

Catálogo de errores comunes

Problema	Evidencia	Solución
HUs excesivamente pequeñas	• Gran volumen de HUs por Sprint. • Difícil alinear la visión de un Sprint con el resultado de la planificación. • Gran número de HUs dependientes.	• Revisar HUs dependientes e intentar fusionarlas en una nueva. Si esto resulta en una HU excesivamente grande, probar con otro patrón de división de HUs (División de Historias de Usuario). • Añadir un mayor número de tareas en la HU en lugar un mayor número de HUs con pocas tareas.
HUs excesivamente grandes	• A pesar de tener conocimiento técnico y del dominio, hay baja confianza en la estimaciones. • De manera consistente, los desarrolladores descubren que el alcance del Sprint es mucho mayor del que inicialmente se estimó.	• Antes de estimar la HU (o de planificarla), los desarrolladores pueden revisar y añadir conjuntamente las tareas que cubran dicha HU.
Dependencias entre HUs	• Difícil estimar HUs. • HUs del Sprint incompletas porque necesitaban de otras para comenzar.	• Revisar HUs dependientes e intentar fusionarlas en una nueva. Si esto resulta en una HU excesivamente grande, probar con otro patrón de división de HUs (División de Historias de Usuario). • Si tras fusionar y probar otros patrones de división la HU sigue siendo demasiado grande, probar a definir mejor el alcance (tareas) para ganar claridad sobre la HU.
Dificultad de priorización de HUs	• Product Owners incapaces de saber por qué debe planificarse una HU.	• Evitar el uso excesivo de HUs técnicas, dando preferencia a HUs funcionales con gran cantidad de tareas técnicas. • Asegurar que la redacción del "para" (o valor de negocio) se expresa en términos del negocio.

"Gold Platting" o Bañado en Oro	• HUs que van mucho más allá de lo que se esperaba. • HUs que añaden funcionalidades extra.	• Reforzar la opinión del equipo QA que valide lo que se pidió / pactó. • Buscar mecanismos de transparencia y visibilidad del resultado con la HU (p.ej. repasando los criterios de aceptación durante la Sprint Review).
Pretensiones de alcance muy definido y controlado	• Pretensiones de HUs estimadas de manera precisa (con horas). • HUs de Sprints lejanos (+3 / 4) definidas.	• Reforzar la responsabilidad del Product Owner.
Detalles de interfaz de usuario en los comicios	• Esfuerzos mayoritarios en definir las interfaces de usuario en los primeros Sprints.	• Los detalles estéticos o de interfaz del aplicativo terminarán apareciendo, pero como inicialmente no sabemos cuál será la solución final, es preferible centrarse en otras capas del aplicativo (lógica de negocio, identificar casos de uso,...).
HUs con todos los detalles ya cerrados e innegociables antes de refinamiento	• Comunicación unidireccional durante el refinamiento. • Baja participación por parte del equipo en la configuración de la HU.	• Redactar una estructura simple de la HU previa al refinamiento, y añadir los detalles en la propia HU tal como vayan emergiendo durante el refinamiento.

Spikes

Los Spikes, provenientes del marco Extreme Programming (XP), son un tipo especial de historia de usuario que se utiliza para obtener el conocimiento necesario para reducir el riesgo de una solución técnica, comprender mejor un requisito o aumentar la confiabilidad de una estimación de la historia. Un Spike es una excelente manera de mitigar los riesgos de manera temprana y permite al equipo obtener feedback de un elemento del backlog que se avecina.

Ejemplos de cuándo utilizar un Spike

- Los desarrolladores no tienen conocimiento de una nueva tecnología. Un Spike podría utilizarse para investigar y garantizar la viabilidad de la solución.
- Una HU utiliza una biblioteca de terceros cuya API está pobremente documentada.
- Una HU contiene un riesgo técnico significativo, y el equipo puede tener que hacer algunos experimentos o prototipos para mejorar el enfoque tecnológico.
- Se desea investigar cómo reacciona un usuario ante una nueva interfaz.

Carácterísticas de un Spike

- **Tiene estimaciones**: Un spike debe estimarse como el resto de HUs del Sprint. Si el equipo asignó cuatro horas a un Spike, SOLO se deben dedicarse esas cuatro horas.
- **Tiene un claro propósito y está orientado a generar un entregable específico**: Un Spike tiene un objetivo claro, que, por lo general, es el de orientar o re-orientar el rumbo de una funcionalidad a través de un entregable específico previamente definido (p.ej. proporcionar un prototipo, una PoC, un informe,...).
- **Tiene criterios de aceptación**: Por ello deben ser estimables, demostrables y aceptables por el el Product Owner.