El Rol del Product Owner en la definición y validación de las *user stories*

Marcelo Estayno[1] y Judith Meles[2]

Resumen

Las *user stories* se introducen por primera vez en el contexto de *Extreme Programming* y son la base del desarrollo de software incremental. Se las define como una descripción corta de una funcionalidad valuada por un usuario o cliente de un sistema. Las *user stories* están compuestas por tres aspectos: descripción, conversación y confirmación. La confirmación es el aspecto que posibilita su validación.

El creador de las *user stories* plantea que la confirmación se expresa por medio de un conjunto de pruebas de aceptación y es responsabilidad del *Product Owner* definirlas. Hemos observado que en la práctica esto rara vez ocurre.

Dado que la validación es de vital importancia para cumplir con los compromisos acordados, presentamos una propuesta que plantea la inclusión explícita de los criterios de aceptación como parte de las *user stories*, siendo su definición responsabilidad del *Product Owner*.

Respecto de las pruebas de aceptación, nuestra propuesta plantea asignar la responsabilidad de su definición al equipo de desarrollo con la colaboración del *Product Owner*.

Palabras clave: requerimientos, ágil, historias de usuario, criterios de aceptación, dueño de producto.

[1] Facultad de Ingeniería, Universidad Nacional de Lomas de Zamora, Argentina.

[2] Universidad Tecnológica Nacional, Facultades Regionales Córdoba y Villa María Córdoba Argentina.

Abstract

The *user stories* were first introduced in the context of Extreme Programming and are the basis of incremental software development. They are defined as a short description of a functionality valued by a user or customer of a system. The *user stories* are composed of three aspects: card, conversation and confirmation. Confirmation is the aspect which enables their validation.

The *user stories* creator suggests that confirmation is expressed by a set of acceptance tests and it´s the responsibility of the *Product Owner* to define them. We have observed that, in practice, this rarely happens.

Since validation is vital to meet agreed commitments, we present a proposal that suggests the explicit inclusion of the acceptance criteria as part of the *user stories*, and its definition as part of the *Product Owner* responsibility

Regarding the acceptance tests, our proposal contemplate assign responsibility for its definition to the development team in collaboration with the *Product Owner*.

Key words: requeriments, agile, user stories, aceptance criteria, Product Owner.

I. Introducción

La familia de métodos de desarrollo ágiles evolucionó a partir de los conocidos ciclos de vida iterativo e incremental. Surgieron de la creencia de que un acercamiento más en contacto con la realidad social y la realidad del desarrollo de productos basados en el aprendizaje, la innovación y el cambio daría mejores resultados.

El propósito principal de estas metodologías es lograr un equilibrio entre los procesos disciplinados que proponen una excesiva carga de trabajo y burocracia, y la no existencia de proceso, que lleva a desarrollar software de manera caótica.

La alternativa que ofrecen las metodologías ágiles, propone un cambio radical que hace énfasis en un replanteo de aspectos culturales, comunicacionales y sociales, que son los principales causantes de fracasos en los proyectos de desarrollo de software.

El manifiesto ágil (Agile Alliance, 2001), refleja el acuerdo de la mayoría de los referentes de la industria que plantean prácticas ágiles. Este manifiesto, en su declaración contiene valores y principios, que quienes adhieren a él, se comprometen a respetar.

Este manifiesto gira en torno a privilegiar lo que es de valor para el cliente, es decir, entregas frecuentes de software funcionando. Propone un ambiente de colaboración entre el *Product Owner* y los miembros del equipo y una actitud de aceptación frente a los cambios que surjan durante el desarrollo del producto de software. También se destaca la importancia y la incidencia que tiene la presencia y colaboración del cliente, representado por el *Product Owner* para alcanzar el éxito del proyecto en lugar de las extensas e infructuosas negociaciones defendiendo un contrato.

Por otra parte, las metodologías tradicionales, imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Las metodologías de la ingeniería tradicional han estado presentes durante mucho tiempo, no obstante, no se han distinguido precisamente por ser muy exitosas; aún menos por su popularidad. La crítica más frecuente a estas metodologías es que son burocráticas y "pesadas". Hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda. (Fowler, 2005).

En el contexto de las metodologías tradicionales, el trabajo con los requerimientos es responsabilidad de la Ingeniería de Requerimientos. Los procesos principales que utiliza para el desarrollo de los requerimientos son: *Elicitación, Análisis y Especificación de requerimientos y Validación* (Sommerville, 2011). El propósito de la Elicitación de Requerimientos es adquirir conocimiento sobre el dominio del problema donde operará el software, de modo tal que el informático pueda comprender y familiarizarse con ese dominio que investiga. En este proceso debe

generarse una "transferencia de conocimiento" desde el Cliente/Usuario[3] al Analista de Sistemas/ Ingeniero de Requerimientos. El Análisis de Requerimientos se refiere al proceso por el cual debemos detectar y resolver conflictos entre los requisitos, definir los límites del software y cómo debe interactuar con su entorno. Es en el proceso de Especificación de Requerimientos donde el profesional informático construye un documento formal, comúnmente conocido como Especificación de Requerimientos de Software (ERS), que debe contener la definición de todos los requerimientos que el sistema debe satisfacer. Finalmente, es en el proceso de Validación que el Cliente/Usuario debe confirmar que lo plasmado en la ERS corresponde con lo que él necesita, en otras palabras, que se ha definido el sistema "correcto".

El documento de Especificación de Requerimientos validado por el Cliente/ Usuario sirve de guía y conduce el desarrollo del producto.

II. La ingeniería de requerimientos y los requerimientos agiles

Se presenta un resumen de los principios básicos que sustentan el enfoque de los requerimientos ágiles y luego se establecerá una comparación con la Ingeniería de Requerimientos.

- El cliente representado por el *Product Owner*[4], participa activamente en el desarrollo. Es quien define y prioriza el producto, acuerda los objetivos y les asigna el valor de negocio.
- El *Product Owner* es quién escribe los requisitos y lo hace "just in time", es decir que escribe los requisitos cuando se necesitan. Los requisitos se detallan en el momento en que hacen falta, en lugar de hacerlo en forma completa al principio, en respuesta a la experiencia adquirida.
- La comunicación y la colaboración activa en el equipo de proyecto es esencial. los requisitos no son contratos fijos, por el contrario, evolucionan durante el desarrollo.
- La actitud de atender los cambios tanto del *Product Owner* como del equipo es abierta y están dispuestos a recibir cambios a lo largo de todo el proyecto, aún en etapas finales.
- Los requisitos se escriben cuando se necesitan y con el nivel de detalle imprescindible.

[3] **Cliente/Usuario:** que en este contexto, un "cliente" es comparable a un "usuario". Tradicionalmente, en la Ingeniería de Requerimientos el cliente es el que paga por un sistema de software y el usuario es quién hará uso del mismo.

[4] Product Owner: Es comúnmente un usuario principal del sistema o alguien de marketing, gestión de productos, o cualquier persona con un conocimiento sólido de los usuarios, el mercado, la competencia y las tendencias de futuro para el dominio o el tipo de sistema que se está desarrollando. También conocido como Product Champion (Larman, 2004) u On-site Customer (Beck, 2004).

• Los requisitos deben detallarse cuando están a punto de ser diseñados, cuando hay una el detalle o bien existe una legislación o regulación que lo requiera.

De esos principios se puede deducir que uno de los aspectos fundamentales del enfoque ágil para el tratamiento de los requerimientos es el *Involucramiento* del Cliente, esto significa tener al cliente "accesible" o "en el sitio" (Beck, 2004). Este es el cimiento para la retroalimentación rápida y la comunicación entre los involucrados. Ambos conducen a una mejor comprensión de las necesidades y expectativas en ambos lados (equipo de desarrollo y cliente).

El involucramiento del cliente es el primer objetivo del enfoque ágil, que asume un usuario "ideal" representante del cliente, el *Product Owner*, que puede responder correctamente todas las preguntas que el equipo de desarrollo pudiera hacerle y tiene el poder de tomar decisiones vinculantes. Es importante recordar que el *Product Owner* debe tener un entrenamiento previo para poder ejercer este rol adecuadamente.

Ahora bien, el involucramiento del cliente también es un factor clave para la Ingeniería de Requerimientos Tradicional (IRT). El cliente tiene el conocimiento acerca del sistema que será desarrollado, conocimiento que se espera sea trasferido al equipo de desarrollo.

Por lo tanto ambos enfoques, ágil y tradicional, coinciden en la importancia del involucramiento del cliente, y ambos utilizan técnicas de elicitación, sólo que difieren en los aspectos entre los que es importante destacar: *disponibilidad del cliente, priorización, modelado y especificación y validación*.

En la IRT el *cliente está involucrado* principalmente durante el proceso de elicitación, luego durante el proceso de validación de requerimientos y al final para la aceptación del producto, prácticamente no tiene participación durante la etapa de desarrollo. En el enfoque ágil la participación e involucramiento del cliente es a lo largo del proceso de desarrollo completo.

La *priorización* es una práctica muy importante en los métodos ágiles, es responsabilidad del *Product Owner* identificar y definir las prioridades de los requerimientos que serán implementados. Durante el desarrollo, el conocimiento y la comprensión respecto del producto crece y nuevos requerimientos pueden incorporarse, por lo que la priorización se realiza repetidas veces durante este proceso, para mantener las prioridades actualizadas. La priorización ayuda a los clientes a responsabilizarse por sus elecciones y a comprender las diferencias entre los requerimientos y cómo pueden afectar al sistema. En la IRT, la responsabilidad sobre la priorización no está explícitamente definida, si bien se intenta que provenga del cliente, como luego no es él quien modela y crea el documento de requerimientos, trabajo que realiza un analista de sistemas o rol similar, queda diluida y con frecuencia es un punto de conflicto durante

la validación, momento en el cual el Cliente debe aprobar el documento de requerimientos de software como "correcto".

El *modelado y la especificación* son los aspectos centrales de la IRT, en este proceso el Analista de Sistemas construye el documento de ERS, con el objeto de dejar plasmados la totalidad de los requerimientos que el producto de software debe satisfacer. Este documento es el artefacto resultante de la ejecución de la ingeniería de requerimientos. El enfoque ágil, en cambio, deja supeditado a la decisión del equipo de proyecto, si es necesario especificar y documentar requerimientos para el software o no, tal como se mencionó en la sección anterior. Como así también las técnicas y herramientas a utilizar.

Por último, la *validación* que propone la IRT, en el contexto de su proceso, es sobre el documento de la ERS, es decir, validar si los requerimientos plasmados sobre el documento están completos y son los correctos. La validación en el contexto ágil juega un rol importante, se realiza repetidamente, a periodos cortos de tiempo, en reuniones de revisión, utilizando pruebas de aceptación y la validación se realiza sobre el producto funcionando. Esto incrementa la confianza del cliente en el producto y en el equipo de desarrollo.

III. Requerimientos ágiles con *user stories*

La técnica de *user story* surge en el seno de un método ágil, XP (Extreme Programming), y en la actualidad es muy utilizada por SCRUM[5], para trabajar la identificación de requerimientos y como unidad de estimación, planificación y gestión del trabajo del equipo en el contexto del proyecto.

Las *user stories* son una práctica común en los métodos ágiles, para introducir los requerimientos en el proceso de desarrollo. A diferencia del enfoque de la ingeniería de requerimientos tradicional, las *user stories* no requieren de la especificación de la solución "*up-front*"[6], en lugar de eso estimulan un diálogo fluido y enriquecedor entre clientes y el equipo de desarrollo, durante la construcción del software, buscando la mejor solución.

Según Albert Mehrabian (Mehrabian, 1971), la comunicación humana se compone de tres partes:

1. En un 7%: El contenido (las palabras, lo dicho)
2. En un 38%: El tono de la voz
3. En un 55%: Las expresiones faciales

[5] **SCRUM:** marco de trabajo para la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.

[6] **Up-front:** por adelantado o desde el principio.

De aquí el fundamento de los métodos ágiles de privilegiar el contacto cara-a-cara entre los involucrados, para poder lograr una comunicación sólida, completa y significativa.

En la figura 1 se presenta la relación entre los espacios que conviven a la hora de construir un producto de software. En el primer círculo, el de la izquierda, está el *Espacio del Problema*, es decir la porción del mundo que necesita el software, allí están los clientes, usuarios y expertos, representados por lo general por el *Product Owner*. En el círculo de la derecha, está el *Espacio de la Solución*, es el lugar donde se va a construir el producto de software, un espacio técnico-disciplinar donde está el equipo de desarrollo. En la intersección de estos espacios, el *Espacio de Coexistencia*, se encuentran las *user stories*, que deben actuar como un objeto frontera[7], facilitando la creación y transferencia de nuevo conocimiento en un espacio de innovación compartido (Shalloway, 2010). Un espacio caracterizado por la ambigüedad y la incertidumbre, donde tanto los representantes de un espacio como los representantes del otro, utilizando un proceso dialéctico de refinamiento sucesivo irán disipando, mientras con un esfuerzo colaborativo construyen el software.

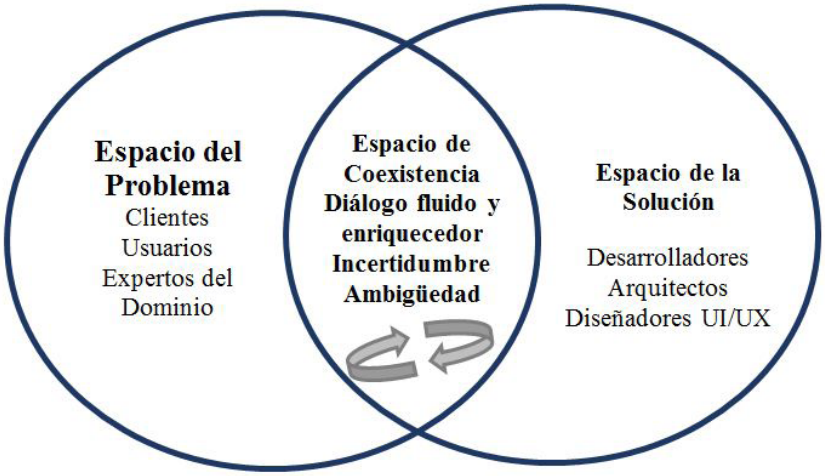

Figura 1: *User stories* como espacio de coexistencia

Ahora bien, para lograr estos objetivos, se debe lograr un equilibrio en este *Espacio de Coexistencia*, para el cual utilizamos las *user stories* como herramienta, puesto

[7] Objetos frontera: elementos débilmente estructurados en el uso común, que se vuelven muy estructurados en el uso individual del sitio. Pueden ser abstractos o concretos. Tienen diferentes significados en diferentes mundos sociales, pero su estructura es bastante común a más de un mundo para que sean medios reconocibles de traducción. La creación y gestión de objetos frontera es clave en el desarrollo y el mantenimiento de la coherencia en la intersección mundos sociales "-. Leigh & Griesemer

que si domina el Espacio del Problema, las necesidades de funcionalidades y fechas se impondrán, sin considerar la realidad o si el equipo de desarrollo comprende los requerimientos. Si por el contrario domina el Espacio de la Solución, el lenguaje técnico se impondrá sobre el lenguaje del negocio y el equipo de desarrollo perderá la oportunidad de aprender del diálogo, de la interacción, es decir lo que escuche.

A. ¿Qué es una *user story*?

Las *user stories* reflejan una necesidad del usuario, una descripción del producto, un mecanismo para diferir una conversación, como así también un ítem de planificación.

El modelo de las 3C´s, propuesto por Ron Jeffries (Jeffries, 2001) para distinguir *user stories* "sociales" de las prácticas de requerimientos "documentales", plantea estos tres componentes:

• **Card (Tarjeta):** Es la manifestación más visible de la *user story*, pero no la más importante, debe "plasmar" el requerimiento, no "especificarlo" La *user story* debe poder escribirse en una tarjeta o *post-it* de tamaño estándar. Si no alcanza el espacio será una señal de que estamos traspasando las fronteras y comunicando demasiada información que debería compartirse cara a cara.

• **Conversación:** Toda *user story* debe tener una conversación con el *Product Owner*. Una comunicación cara a cara que intercambia no sólo información sino también experiencias, pensamientos, opiniones y sentimientos. *Son la parte más importante de la user story*. El detalle que se obtiene debe ser un resultado de esta conversación, no un reemplazo de la misma.

• **Confirmación** – Sirve para determinar lo que el *Product Owner* espera, es decir para que el equipo de desarrollo sepa lo que debe construir. Son pruebas que comunican y documentan detalles y que se pueden utilizar para determinar si una historia está completa.

B. Card: la parte más visible de la *user story*

La sintaxis sugerida para expresar las *User stories* nos permite responder a las preguntas:

• *quien*, representa el rol que necesita la funcionalidad;

• *qué*, permite la segmentación de la funcionalidad del producto;

• *porqué*, comunica la utilidad de la funcionalidad desde la perspectiva del negocio.

Por lo tanto la sintaxis se expresa de la siguiente forma:

Como <nombre del rol>, yo puedo <actividad> de forma tal que <valor de negocio que recibo> (Cohn, *user stories* Applied, 2004)

La figura 2 muestra un ejemplo de una *user story* para el software de un GPS.

Como Conductor quiero buscar un destino a partir de sus coordenadas para poder conocer el camino a recorrer para llegar al destino deseado.

Figura 2: Ejemplo de la tarjeta de una user story con la sintaxis sugerida

Se llama a esta forma "la voz del usuario" [3], es una construcción sumamente útil porque abarca el espacio del problema (valor de negocio entregado) y el espacio de la solución (actividad que el usuario realizará utilizando el software). También proporciona una perspectiva del primer usuario (rol), que mantiene enfocado al equipo en el valor del negocio y en la solución de problemas reales para la gente real.

Mike Cohn, en su artículo (Cohn, Advantages of the "As a user, I want" *user story* template, 2008) defiende la importancia de respetar la sintaxis de las *user stories* fundamentada en tres razones:

1) Primera Persona: La redacción en primera persona invita a quien lee la *user story* a ponerse en el lugar del usuario, identificarse de manera cercana con lo que piden.

2) Priorización: Tener esta estructura ayuda al *Product Owner* a priorizar. Sin esa estructura, debe trabajar más para comprender cuál es la funcionalidad, quien se beneficia y cuál es el valor de la misma.

3) Propósito: El valor comunica porqué la actividad es necesaria. Conocer el propósito de una funcionalidad permite al equipo de desarrollo plantear alternativas que cumplan con el mismo propósito de una manera diferente, en caso que eso sea necesario.

1) **Definición de Listo (Ready) y definición de Hecho (Done) en las *User stories***

La definición de Listo, en inglés ***D**efinition **o**f **R**eady* (DoR) se aplica a las *user stories* y se utiliza para asegurar que están en condiciones de ser incluidas en la siguiente iteración del proyecto, que Scrum llama Sprint Backlog[8].

La definición de hecho, en inglés *Definition of Done (DoD)*, se utiliza al final de un Sprint, para determinar si la funcionalidad construida está en condiciones de ser utilizada. El significado de que una *user story* este "*hecha*", es algo que debe ser comprendido por todo el equipo y se utiliza para evaluar cuándo se ha completado el trabajo. A medida que los equipos de desarrollo maduran, se espera que su definición de "hecho" se amplíe para incluir criterios más rigurosos, que permitan incrementar la calidad del producto.

[8] **Sprint Backlog:** es el conjunto de elementos tomados del Product Backlog, seleccionados para ser desarrollados durante un Sprint. Es una predicción hecha por el Equipo de Desarrollo acerca de qué funcionalidad formará parte del próximo incremento y del trabajo necesario para entregar esa funcionalidad.

Tanto la definición de listo (DoR) como la definición de hecho (DoD) deben ser acordadas por los miembros del equipo, no obstante ello, se recomienda utilizar como referencia los checklists de las figuras 3 y 4 propuestas en el libro de Rubin (Rubin, 2013).

» Valor de negocio claramente expresado.
» Detalles suficientemente comprendidos por el Equipo de forma tal que puedan tomar una decisión informada, sobre si pueden completar la *user story*.
» Dependencias identificadas, no hay dependencias externas que puedan impedir que la *user story* se complete.
» El equipo ha sido asignado adecuadamente para completar la *user story*.
» La *user story* ha sido debidamente estimada y es lo suficientemente pequeña para ser completada en un Sprint.
» Los criterios de aceptación son claros y testeables.
» El equipo comprende como mostrar la *user story* durante la presentación al *Product Owner* (en Scrum Sprint Review[9]).

Figura 3: Checklist para la verificación del criterio de listo (Ready)

» Diseño revisado
» Código Completo
» Código re factorizado (con formato estándar, comentado, en el repositorio, inspeccionado)
» Documentación de Usuario actualizada
» Probado (con pruebas de unidad, integración y regresión hechas)
» Plataforma probada
» Lenguaje probado
» Cero defectos conocidos
» Prueba de Aceptación realizada
» Instalado en los servidores de producción

Figura 4: Checklist para la verificación del criterio de hecho (Done)

C. Conversación: la parte más importante de la *user story*

Como un medio de comunicación, la conversación en la que se detallan las *user stories*, facilita una compresión común de lo que es necesario construir. Permite que la gente que comprende lo que debería crearse, comunique claramente sus deseos a quienes deben crearlo. La mejor forma de asegurarse que se construya el producto

[9] **Sprint Review:** es una de las ceremonias de Scrum cuyo propósito principal es presentar el incremento del producto obtenido durante el Sprint. En ese momento el Product Owner es responsable de la revisión y aceptación de las user stories que el equipo de desarrollo se comprometió a entregar.

deseado, es que la persona que sabe tenga conversaciones oportunas y periódicas con la gente que está diseñando, construyendo y probando ese producto.

La conversación representa una discusión entre el equipo, el cliente, el *Product Owner* y otros involucrados, que se necesita para determinar el comportamiento detallado, requerido para implementar la *user story*.

Jeffries (Jeffries, 2001), plantea que la conversación es lo que le da contenido a lo escrito en la tarjeta, representa los detalles que ponen de manifiesto la intención del *Product Owner.* En otras palabras, la tarjeta representa una "*promesa para una conversación*" acerca de la intención expresada en la misma.

Al utilizar el término "conversación", es importante destacar que en realidad, no es un evento de única vez, sino que es un *diálogo permanente y continuo*. Puede haber una conversación inicial cuando se escribe la *user story*, otra conversación cuando es refinada, otra cuando es estimada, otra durante la planificación del sprint, momento en el cual la *user story* se descompone en tareas, y finalmente conversaciones mientras la *user story* está siendo diseñada, construida y probada durante el sprint.

La comunicación verbal tiene el beneficio de ofrecer un espacio de comunicación amplio y brinda realimentación inmediata, haciendo que sea más rápido y más económico lograr una visión compartida.

En los ambientes de desarrollo ágil, la conversación es la herramienta que permite asegurar que los requerimientos han sido adecuadamente discutidos y comunicados.

D. Confirmación: base para la validación de la *user story*

La confirmación está representada por las pruebas de aceptación. Las pruebas de aceptación son la forma de verificar que las *user stories* tienen la funcionalidad que el *Product Owner* tenía en mente cuando escribió la historia.

Leffingwell (Leffingwell, 2011) en su libro, las nombra pruebas de aceptación de historia (story acceptance tests) para separarlas de otro tipo de pruebas de aceptación, por ejemplo de sobrecarga, dado que estas pruebas son pruebas funcionales. En la figura 5 se muestra un conjunto de pruebas de aceptación para la *user story* presentada en la figura 2.

- Probar buscar un destino en un país y ciudad existentes, de dos coordenadas existentes (pasa).
- Probar buscar un destino en un país y ciudad existentes, de una coordenada inexistente (falla).
- Probar buscar un destino en un país y ciudad existentes, de dos coordenadas existentes sin indicar la orientación (falla).
- Probar ingresar coordenadas de latitud y longitud válidas (pasa).

Figura 5: Ejemplo de Confirmación para la user story de la figura 2. Las pruebas de aceptación para la user story. Suelen estar al dorso de la tarjeta (Card).

Los desarrolladores construyen la historia y los *Product Owners* escriben las pruebas de aceptación en el dorso de la tarjeta. Las pruebas de aceptación deberían escribirse lo más temprano posible en la iteración. Escribir las pruebas de aceptación en forma temprana es muy útil para el *Product Owner* ya que las expectativas y asunciones son puestas en evidencia y comunicadas en forma temprana a los desarrolladores. Los desarrolladores por su parte utilizarán toda la información que proveen las pruebas de aceptación para construir la historia.

Una buena práctica recomendada por Cohn (Cohn, *User stories* Applied, 2004), que un *Product Owner* puede adoptar para escribir las pruebas de aceptación, es mirar la *user story* que escribió y plantearse preguntas tales como: ¿Qué otra cosa necesita saber el equipo de desarrollo sobre esta historia? ¿Hay alguna circunstancia por la cual esta historia podría tener un comportamiento diferente? ¿Qué podría funcionar mal durante esta historia? ¿Estoy asumiendo algo respecto de cómo será implementada la historia?

Como se mencionó anteriormente, el creador de las *user stories*, Cohn (Cohn, *user stories* Applied, 2004), asigna la responsabilidad de escribir las pruebas de aceptación al *Product Owner*, justificado en el hecho que el software se crea para satisfacer la visión del *Product Owner*, si bien el equipo de desarrollo puede colaborar, él es quien debe especificar las pruebas de aceptación que se utilizarán para determinar si la *user story* está correctamente desarrollada.

Leffingwell (Leffingwell, 2011) plantea que lo siguiente debería cumplirse respecto de la definición de las pruebas de aceptación para una *user story*:

- Escritas en el lenguaje del negocio.
- Desarrolladas durante la conversación entre el *Product Owner* y el equipo de desarrollo.
- El *Product Owner* es el dueño de las pruebas de aceptación, aunque las mismas puede escribirlas cualquiera en el equipo.
- Son pruebas de caja negra, sólo verifican que los criterios de satisfacción se cumplan sin observar "cómo" se cumplen.
- Son implementadas durante el curso de la iteración en la cual se implementa la historia.

Además plantea las siguientes características que las buenas pruebas de aceptación de historia deberían cumplir:

- Deben probar una buena *user story* (que cumpla con el modelo INVEST).
- Deben probar todos los escenarios, dado que la *user story* es en sí misma liviana, las pruebas de aceptación cargan con los detalles.
- Deben ser persistentes durante toda la vida de la aplicación, lo que permite saber cómo trabaja el software y permite que las *user stories* sean descartadas después de haberlas implementado.

Las pruebas de aceptación tienen por objeto demostrar que una aplicación es aceptable para el *Product Owner* que ha sido responsable de guiar el desarrollo del sistema, Leffingwell (Leffingwell, 2011). Esto significa que es él quien debería ejecutar las pruebas de aceptación al final de cada iteración. Dado que el tiempo que lleva esta actividad es considerable y tiende a crecer exponencialmente en las sucesivas iteraciones, debido al incremento de funcionalidad y a la necesidad de probar incrementalmente (pruebas de regresión), es recomendable tratar de automatizar las pruebas tanto como sea posible.

1) **Criterios de Aceptación**

Los criterios de aceptación son condiciones de satisfacción, específicos para cada *user story* bajos las cuales el producto debe satisfacer los requerimientos funcionales y no funcionales. Estos criterios son definidos por el *Product Owner* y serán verificados en las pruebas de aceptación, además de la verificación del "criterio de hecho" definido, que aplica a todos los ítems del product Backlog. En este sentido una *user story* puede ser considerada hecha sólo si cumple con ambos, el criterio de aceptación específico y el criterio de hecho definido para el sprint, este último afecta a todas las *user stories*.

Leffingwell (Leffingwell, 2011), plantea que en la tarjeta de la *user story* se puede incluir información adicional, notas, asunciones y/o criterios de aceptación que posibiliten clarificar la historia. En la figura 6 se muestra la *user story* de la figura 2 a la que se le agregaron los criterios de aceptación.

Como Conductor quiero buscar un destino a partir de sus coordenadas para poder conocer el camino a recorrer para llegar al destino deseado.

Criterios de Aceptación: Las coordenadas se representan con tres números que indican longitud y tres números que indican latitud. Cada número representa los grados, minutos y segundos respectivamente. Además se debe indicar

Figura 6: Ejemplo de la user story de la figura 2 con criterios de aceptación.

Rubin (Rubin, 2013), por su parte, plantea que dado que las condiciones de satisfacción pueden expresarse como pruebas de aceptación de alto nivel, el *Product Owner* debería encargarse de la definición de esas condiciones de satisfacción, al dorso de la tarjeta de la *user story*.

Posteriormente y basado en lo anterior, el equipo de desarrollo, debería definir las pruebas de aceptación y detallarlas a nivel técnico, con la colaboración del *Product Owner*, puesto que ese nivel de detalle técnico requerido, el *Product Owner* no tiene porqué conocerlo.

IV. *User stories* y los requerimientos no funcionales

Los requerimientos no funcionales (RNF), también conocidos como requerimientos de calidad, están relacionados con atributos como confiabilidad, eficiencia, usabilidad, mantenibilidad, portabilidad, entre otros.

Los RNF representan restricciones a nivel de sistema y son importantes porque afectan el diseño y la prueba de la mayoría de las *user stories*. Por ejemplo el RNF del navegador, presentado en la figura 7, debería ser común en cualquier proyecto web. Según expresa Rubin (Rubin, 2013), los requerimientos no funcionales pueden escribirse tanto respetando como no respetando la sintaxis sugerida para la *user story*. La Figura 7 se muestra un RNF que no respeta la sintaxis y la figura 8 en cambio, se muestra un ejemplo en el que el RNF está expresado respetando la sintaxis de la *user story*.

El sistema debe soportar IE8, IE9, Firefox 7, Safari 5 y Chrome 15

Figura 7: Ejemplo de Tarjeta con Requerimiento no Funcional sin sintaxis de User story

Como usuario quiero una interfaz en Inglés, una lengua romance, así tengo alta probabilidad de que funcione en los 70 lenguajes requeridos

Figura 8: Ejemplo de Tarjeta con Requerimiento no Funcional respetando la sintaxis de la User story

Según Cohn (Cohn, *User stories* Applied, 2004), los requerimientos no funcionales deben ser considerados como restricciones al comportamiento del sistema. Mewkirk and Martin, citados en (Cohn, *User stories* Applied, 2004) recomiendan un práctica para la definición de los requerimientos no funcionales, ellos plantean que se utilice una tarjeta, con la palabra "Restricción" en la parte inferior tal como se muestra en el ejemplo de la figura 9.

El sistema debe soportar hasta 50 usuarios concurrentes.
Restricción

Figura 9: Ejemplo de Tarjeta de Requerimiento no Funcional como restricción.

Los RNF´s condicionan la funcionalidad, quitando algún grado de libertad en el diseño, es por eso que se los plantea como restricciones. Algunos RNF´s condicionan trasversalmente a todo el producto y otros son aplicables a alguna *user story* en particular. Por lo tanto y considerando este último caso, nuestra propuesta se basa en la posibilidad de expresar los RNF´s como parte de los criterios de aceptación, según se muestra en la figura 10.

Como Conductor quiero buscar un destino a partir de una calle y altura para poder conocer el camino a recorrer para llegar al destino deseado.
Criterios de Aceptación: La búsqueda no puede demorar más de 10 segundos.

Figura 10: Ejemplo de una user story con un requerimiento no funcional expresado como criterio de aceptación.

V. Uso de las *user stories* en los proyectos

En función de lo descripto en las secciones anteriores, se puede acordar que los métodos ágiles plantean una redefinición de los roles y responsabilidades de los actores que intervienen en el desarrollo de software, redistribuyendo en algunos casos esas responsabilidades. Tal es el caso, cuando se analiza el rol del *Product Owner* quien siendo un referente del negocio que no tiene un perfil técnico-informático, es el responsable de la definición de los requerimientos del producto, priorizando aquellas *user stories* que son de valor para el negocio. También debe definir las pruebas de aceptación, que luego se emplearán para validar el producto al final de la iteración (sprint).

Cuando las organizaciones comienzan a implementar estas prácticas, se advierte que quienes asumen el rol de *Product Owner*, muchas veces comienzan a tener dificultades para cumplir con todas las responsabilidades que se espera puedan desempeñar.

En relación específica con las *user stories* es que, si bien se presentan algunos inconvenientes a la hora de escribirlas y priorizarlas, las mayores dificultades están relacionadas con la definición de las pruebas de aceptación.

Esta situación es riesgosa, si el equipo no asume la responsabilidad de escribirlas y luego validarlas con el *Product Owner*; situación que se agrava si el equipo decide comenzar a implementar las funcionalidades de las *user stories* sin tener las pruebas de aceptación definidas previamente.

VI. Rol del *Product Owner* en las *user stories*

A modo de resumen, se repasan en esta sección las responsabilidades que debe asumir el *Product Owner* en el contexto de un proyecto de desarrollo ágil.

En un primer momento es responsable de conducir al equipo en la creación de una visión compartida respecto del producto. Es responsable de maximizar el valor de negocio que se va a obtener. Debe escribir las *user stories* y priorizarlas. Debe definir los criterios de aceptación y escribir las pruebas de aceptación

También debe estar disponible para todas las sesiones de conversación que

sean necesarias para ampliar y completar lo escrito en las *user stories*, recordando que estas conversaciones se darán en varios momentos conforme el desarrollo evoluciona, siendo uno de los primeros momentos, las sesiones de estimación de las *user stories*.

Por último cuando está listo el incremento del producto, es quien durante la Sprint Review, lo validará y decidirá cuáles de las *user stories* que han sido desarrolladas, satisfacen sus necesidades y por consiguiente las dará como aceptadas.

De lo antes dicho, se puede deducir que este rol conlleva una importante carga de trabajo que contiene responsabilidades técnicas para las que no siempre está preparado. Una de las responsabilidades que se ha detectado, es la definición de las pruebas de aceptación, por lo tanto consideramos, que esta actividad debería ser absorbida por el equipo de desarrollo, con la participación fundamental del *Product Owner* y mantener para sí la responsabilidad de la definición de los criterios de aceptación, los cuales recomendamos sean incorporados de forma explícita como parte de las *user stories*.

VII. Conclusiones

Las organizaciones que están incorporando prácticas ágiles, se enfrentan con los desafíos que traen aparejados estos cambios, tanto en la definición de responsabilidades, en la estructuración de los puestos de trabajo, en la formación de sus colaboradores, en la cultura misma de la compañía.

Las *user stories* se han convertido en la técnica para la gestión de requerimientos de usuario, más adoptada por quienes desarrollan software utilizando métodos ágiles.

A pesar de su gran aceptación en la industria, en la actualidad existen algunos aspectos de las *user stories* en los que se presentan diferencias para su creación, en la determinación de que elementos debe incluir y respecto de las responsabilidades de cada uno de los roles involucrados.

Una de las mayores ventajas que ofrecen las *user stories* por sobre los otros enfoques de requerimientos es que estimulan el diseño participativo. Quien asume el rol de *Product Owner* debe convertirse en un participante activo del proceso de construcción del software y debe estar capacitado para hacerlo.

De este análisis se ve la importancia que tiene el rol del *Product Owner* quien habla a través de las *user stories*, escribiéndolas, priorizándolas y definiendo los criterios para su validación.

En definitiva, el *Product Owner* debería encargarse de la definición de los criterios de aceptación y posteriormente, el equipo de desarrollo debería definir las pruebas de aceptación y detallarlas a nivel técnico, con la colaboración del *Product Owner*.

También se considera importante destacar el riesgo asociado a la asignación de responsabilidades técnicas como la definición de las pruebas de aceptación al rol de *Product Owner*, cuando quién asume ese rol, se espera que sea un referente del negocio.

En base a esto nos planteamos como línea de trabajo futuro desarrollar un conjunto de lineamentos para fortalecer el rol de *Product Owner* como integrante de un proyecto ágil, considerando que las expectativas puestas sobre este rol no son usuales en los usuarios/clientes.

Referencias

Agile Alliance. (2001). The agile manifesto. Obtenido de http://www.agilealliance.org/the-alliance/the-agile-manifesto/

Beck, K. (2004). Extreme Programming Explained: Embrace Change . Boston: Addison Wesley.

Cohn, M. (2004). User stories Applied. Estados Unidos: Addison Wesley.

Cohn, M. (2006). Agile Estimating and Planning. Estados Unidos: Pearson Education.

Cohn, M. (25 de Abril de 2008). Advantages of the "As a user, I want" user story template. Obtenido de http://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template/

Fowler, M. (Diciembre de 2005). Martin Fowler. Obtenido de http://www.martinfowler.com/articles/newMethodology.html

Jeffries, R. (30 de 08 de 2001). XProgramming.com. Recuperado el 25 de 02 de 2014, de http://xprogramming.com/articles/expcardconversationconfirmation/

Larman, C. (2004). Agile & Iterative Development. A Manager Guide. Estados Unidos: Addison Wesley.

Leffingwell, D. (2011). Agile Software Requeriments. Estados Unidos: Addison Wesley.

Mehrabian, A. (1971). Silent messages. Oxford: Wadsworth.

O´hEocha, C. &. (2010). The Roles of the User story Agile Practice in Innovation. En Lean Enterprise Software and Systems. Alemania: Springer.

Rubin, K. (2013). Essential Scrum- A practical guide to the most popular agile process. Estados Unidos: Addison Wesley.

Shalloway, A. (2010). Lean-Agile Software Development: Achiving Entreprise Agility. Upper Saddle River, NJ: Addison-Wesley.

Sommerville, I. (2011). Ingeniería de Sfotware (Novena ed.). Mexico: Addison-Wesley.

Wake, W. C. (2000). Extreme Programming Explored. William C. Wake.