**User Stories (US)**

La comunicación es uno de los principales problemas de los requerimientos del software. Por un lado, tenemos al equipo técnico o de desarrollo y por el otro tenemos a los clientes. Unos se manejan en un lenguaje técnico y otros en el lenguaje del negocio. Y lo ideal es que ninguno de estas partes domine la comunicación, ya que esto llevará al proyecto de software al fracaso.

Pero el desarrollo de software es impredecible, el cliente puede querer nuevas funcionalidades o cambios al proyecto de software, y debido a esto y a que el software es intangible, hace difícil estimar cuánto tiempo llevará el proyecto.

En la gestión de proyectos tradicional, todas las decisiones son tomadas al inicio del mismo, sin embargo, este enfoque tiene muchos problemas debido a dicha impredecibilidad del software. Por lo que para resolver esto debemos tomar decisiones basadas en la información que disponemos actualmente, y tomamos esas decisiones seguido. Es decir, vamos a tomar decisiones a lo largo de todo el proyecto. Para hacer eso necesitamos procesar la información que nos llega de manera temprana y constante, y para ello surgen las US.

**¿Qué es una User Story?**

Describe una funcionalidad que va a ser de valor para un usuario. Está compuesta por tres aspectos:

* Una frase verbal: Usada para identificar la user.
* Conversaciones acerca de la user que sirvan para brindar más información
* Pruebas que sirvan para determinar cuándo una US está completa.

Los aspectos de una US se pueden definir como las tres C: Card, Conversation y Confirmation.

Card: Las tarjetas representan requisitos de usuario, no los documentan. Si bien la tarjeta contiene el texto de la US, los detalles se encuentran en la conversación y quedan escritos en la confirmación. En una o dos oraciones se describe la story con el formato Como <rol> quiero <actividad> para <valor>

Conversación: Representa una charla entre el equipo, el cliente, el PO, y los stakeholders, la cual es necesaria para esclarecer los detalles requeridos para implementar la story.

Confirmación: representan las pruebas de aceptación, sirven para determinar cuando una story ha sido implementada por completo. Es decir, son las condiciones de satisfacción.

Como las US representan funcionalidad que le debe agregar valor al negocio, las US que contengan requisitos no funcionales son incorrectas, ya que no le agregan valor al negocio o al usuario. Ej: El lenguaje de programación no le interesa al usuario. (Aunque esto depende del contexto de la US).

La clave es que las historias sean escritas de forma tal que el cliente pueda valorarlas, no deben estar escritas en lenguaje técnico.

Esta forma de escribir las User Stories se la conoce como “User Voice” y me permite hacer foto en el espacio del problema (valor entregado) y en el espacio de la solución (actividad). También me permite pensar la funcionalidad desde la perspectiva del usuario.

**Las User Stories no son requerimientos**:

* No son requerimientos detallados, sino que son expresiones negociables de intención.
* Son cortas y fáciles de leer, entendibles tanto para los desarrolladores, stakeholders y usuarios.
* Representan pequeños incrementos de funcionalidad que agrega valor, y pueden ser implementadas en días o semanas.
* Son relativamente fáciles de estimar.
* No se encuentran detalladas en documentación, sino que organizadas en listas que pueden ser reorganizadas.
* Son elaboradas bajo la metodología just-in-time, es decir, evitando especificarlas en etapas tempranas de desarrollo. No son detalladas al inicio del proyecto.
* No necesitan mantenerse y pueden ser descartadas después de la implementación.
* Junto con el código, sirven para posteriormente hacer la documentación.

**¿Dónde están los detalles de una User Story?**

Muchos de los detalles de una US pueden ser expresados en US adicionales. De hecho, es mejor tener más stories que stories muy grandes.

Cuando una US es muy grande, se la conoce como una épica. Una épica entonces es una user story que puede ser separada o dividida en dos o más stories de menor tamaño. Pero tampoco es conveniente separar demasiado las stories, de forma tal que cada pequeño detalle sea una story distinta. También existen los themes (temas) que son colecciones de user stories relacionadas.

Lo que se hace, en cambio, es tener una discusión acerca de los detalles de una US, cuando los detalles se vuelven importantes. Si bien se pueden tomar anotaciones sobre la conversación, la clave es dicha conversación, no las anotaciones. Los detalles se pueden incluir en la documentación interna de los equipos o en pruebas de aceptación automatizadas.

Una US no son obligaciones contractuales. Los acuerdos son documentados por las pruebas de usuario, que demuestran que una US ha sido completada correctamente.

**Criterios de Aceptación de las User Stories**

Además de la frase verbal, y la descripcion de la user story, es posible incluir los criterios de aceptación. Muchas discusiones sobre una story van a tener lugar entre el equipo y los clientes. Los distintos flujos en la actividad, límites de aceptación y otras particularidades deben incluirse con la story. Muchos de estos criterios de aceptación se transformarán en pruebas de aceptación.

Definen los límites para una US, y sirven para que los PO comuniquen aquello que necesitan para que la US provea de valor.

Los criterios de aceptación buenos definen una intención, no una solución. Deben ser independientes de la implementación.

**¿Cuándo terminar con una User Story?**

Es importante saber las expectativas de los usuarios, que es lo que esperan del proyecto. Dichas expectativas son capturadas en forma de pruebas de aceptación. Se suelen colocar detrás de la tarjeta, y son recordatorios de qué cosas se deben probar de la US. Se escriben con el verbo infinitivo “Probar” seguido del criterio.

Las descripciones de las pruebas de aceptación deben ser cortos, ya que las pruebas pueden ser añadidas o removidas en cualquier momento. La idea detrás de las mismas es escribir suficiente información acerca de la US para que los desarrolladores puedan determinar cuando la misma esta completa.

**Las User Stories son multipropósito**

Las stories son: una necesidad del usuario, una descripción del producto, un ítem de planificación, un token de conversación, un mecanismo para diferir una conversación.

El enfoque tradicional nos enseñó que, al construir un sistema, se avanza capa por capa, se inicia por la capa de red, luego por la capa de persistencia, una capa de lógica y por último una de presentación. Pero este enfoque demora mucho en entregar valor al cliente. Por lo que las User Stories actúan como porciones verticales en lugar de horizontales, esto me permite brindarle al cliente una porción vertical, mostrándole un pedazo de toda la arquitectura, que le brinda una funcionalidad que le agrega valor.

**¿Cómo será el proceso de desarrollo?**

En un proyecto ideal, tendremos una única persona que establece las prioridades de trabajo para los desarrolladores, sabe mucho sobre el negocio y tiene la capacidad para tomar las decisiones importantes. Esta persona suele ser el Product Owner, y deberá responder todas las preguntas del equipo de desarrollo y escribir todas las stories. Sin embargo, eso no suele ocurrir, por lo general hay varias personas de parte del cliente, y en el peor de los casos, ninguna (esto lleva al fracaso). En general, en la vida real, cualquier miembro del equipo con suficiente conocimiento sobre el dominio puede escribirlas, pero depende del PO aceptar y priorizar dichas stories.

Hay dos razones para que los clientes escriban las US. La primera es que cada story debe estar escrita en el lenguaje del negocio, no con lenguaje técnico, de esta manera, el PO puede priorizar las stories para incluirlas en iteraciones y en versiones. Como segunda razón, el cliente, quien quiere el producto, está en la mejor posición para describir el comportamiento del mismo.

Un proyecto que utiliza stories tendrá un ritmo distinto que un proyecto tradicional. Este último, puede utilizar un proceso en cascada donde se escriben los requerimientos, se analizan, se diseña una solución, se programa y por último se testea y se entrega. (También pueden usar un proceso iterativo e incremental). El cliente solo participa al inicio (en la elicitación) y al final (en la entrega del producto). Pero esto no funciona, y muchas veces conduce al fracaso del proyecto.

En los proyectos conducidos por stories, los clientes y usuarios permanecen involucrados en toda la duración del proyecto. Esto es lo mismo independientemente del framework ágil que se utilice. El proceso de escritura de user stories se suele empezar considerando los distintos tipos de usuarios del sistema y luego pensando US que le agreguen valor a dichos usuarios.

**Proceso** (No se si va): Las stories iniciales de un proyecto suelen ser escritas en un workshop, pero las stories se pueden escribir a lo largo de todo el proyecto. Al inicio, en el workshop, se hace un brainstorm de todas las posibles US, y luego se comienzan a estimar.

Juntos, el equipo de desarrollo y el cliente establecen una longitud de la iteración, que va a ser usada durante todo el proyecto. Al final de cada iteración se debe entregar un producto funcional, aunque no el producto completo. El cliente participa en todo el proyecto, y trabaja durante las iteraciones para validar pruebas.

Una vez elegida la duración de la iteración, los desarrolladores deben estimar cuánto trabajo pueden hacer por iteración, esto recibe el nombre de velocity. La primera estimación de la velocity será errónea (si el equipo no viene trabajando motivado y junto desde hace tiempo). Pero se usa esta primera estimación para hacer un plan de entrega, estableciendo qué trabajo se hará en cada iteración, y cuántas iteraciones se necesitarán.

Para planificar la entrega, se ordenan las stories en varias pilas que representan las iteraciones. Cada pila contiene una cierta cantidad de stories, y el tamaño estimado de las stories sumado, debe ser inferior a la velocity estimada. Las stories con mayor prioridad deben ir en la primera pila, y así se van ordenando en las pilas sucesivamente.

Al inicio de cada iteración, el equipo del cliente puede realizar correcciones al plan. Mientras las iteraciones son completadas, nos podemos dar una idea de la velocity actual del equipo, y podemos trabajar con esta en lugar de la estimación inicial.

También es posible que algunas stories sean mucho más fáciles de lo estimado, por lo que se pueden incluir stories adicionales en una iteración. Aunque también es posible que pase justo lo contrario.

El producto backlog lo podemos entender como una gran pila donde el PO organiza las stories. En general, en cada iteración se implementan los ítems de trabajo de mayor prioridad, y son aquellos que se modelan con más detalle. Si es necesario agregar un requerimiento, este es priorizado y agregado a la pila. Aquellos ítems que se encuentran más debajo en la pila (prioridad baja) no son modelados hasta que es necesario hacerlo (just-in-time)

**Planear las entregas y las iteraciones**

Una entrega está compuesta por una o más iteraciones. La planificación de entregas se refiere a establecer un balance entre una línea de tiempo esperada y un set de funcionalidades. La planificación de iteraciones se refiere a determinar que stories incluir en la iteración. Se debe involucrar al cliente en ambas planificaciones.

El cliente prioriza las stories al planificar la entrega. Pero también los desarrolladores tienen diferentes prioridades para varias stories, dependiendo del riesgo técnico u otros motivos, pero al final de cuentas, las stories se priorizan para maximizar el valor otorgado para el cliente.

Sin embargo, tampoco se puede priorizar las stories sin tener en cuenta su peso. Es decir, esta es otra variable que se debe tener en cuenta a la hora de priorizar. El peso de una story es dado por los desarrolladores en forma de story points.

El plan de entrega es construido asignando stories a las iteraciones de la entrega. Los desarrolladores establecen su velocity esperada, la cual es el número de story points que pueden completar por iteración. Luego el cliente ubica las stories en las iteraciones, asegurándose que el número de story points asignados a cualquier iteración no supere la velocity del equipo

**¿Qué son las Pruebas de Usuario o Pruebas de Aceptación?**

Es el proceso de verificar que las stories hayan sido desarrolladas de forma tal que funcione exactamente igual a la forma en la que el cliente espera que funcione. Los test deben ser escritos lo más pronto posible dentro de una iteración.

Este proceso es extremadamente importante ya que le permite a los desarrolladores saber las expectativas del cliente.

**Modelado de Usuarios / Roles**

Existen también usuarios representantes también llamados proxies, que no son los ideales, ya que no son usuarios verdaderos, pero pueden ser gerentes de usuarios, gerentes de desarrollo, vendedores, expertos en el dominio, etc.

**Razones por las que utilizar User Stories**

* Las User Stories hacen énfasis en la comunicación verbal y cara a cara con el cliente: De esta manera las decisiones importantes no se guardan en documentos que rara vez se leen.
* Las User Stories son fácilmente comprensibles por tanto los clientes como los desarrolladores: Esto se debe a que están libres de lenguaje técnico
* Las User Stories son del tamaño adecuado para la planeación: Esto se debe a que representan una porción discreta de funcionalidad
* Las User Stories funcionan bien con el desarrollo iterativo: Permitiendo hacer progreso mediante el refinamiento sucesivo del producto. En cada iteración el producto de software es mejorado mediante la incorporación de mayor detalle. Se dice que las stories funcionan bien con desarrollo iterativo debido a que también es posible iterar sobre stories. Ya que para una funcionalidad que queremos eventualmente, pero no es importante tenerla al principio, se puede escribir una épica. Cuando se deba agregar la story al sistema, se puede refinar al separar la épica en varias stories más pequeñas-
* Las User Stories permiten hacer énfasis en los detalles una vez que ya comprendemos lo que vamos a realizar. Ya que podemos escribir épicas, no es necesario escribir stories de partes del sistema hasta que estemos cerca de desarrollar dichas partes. Es decir, me permite no gastar tiempo en los detalles de una nueva funcionalidad, hasta que dicha funcionalidad sea necesaria de incluir en la iteración.

**INVEST Model**

Una User Story escrita con calidad es Independiente, Negociable, Valuable para el cliente, Estimable, Small (Pequeña) y Testeable. Estos son los atributos que debe tener una user, para asegurar la calidad de la misma.

**Independiente**: Se debe evitar tanto como se pueda introducir dependencias entre stories. Ya que, si estas son dependientes entre sí, introducen problemas a la hora de priorizar y planear. En caso de encontrarte stories dependientes, es recomendable combinarlas en una historia más grande, o encontrar otra forma de dividirlas. Deben poder implementarse en cualquier orden

**Negociable**: Las stories son negociables, no son contratos escritos o requerimientos que el software si o si debe implementar. Las stories son descripciones cortas de funcionalidad, y sus detalles son negociados en una conversación entre el PO y el equipo de desarrollo. Además, les brinda mayor flexibilidad a las user stories, haciendo que el equipo tenga más flexibilidad a la hora de alcanzar los objetivos de una entrega.

**Valuable**: Debe tener valor para el cliente. Se deben evitar todas las historias que solo tienen valor para el equipo de desarrollo. También es conveniente dejar fuera de las stories definiciones sobre la interfaz de usuario. La mejor forma de asegurar que las stories tengan valor para el cliente, es hacer que el Product Owner las escriba. Es el atributo más importante para una user story.

**Estimable**: Es importante que los desarrolladores puedan estimar el peso de una story o la cantidad de tiempo que demorarán en implementar dicha story. Hay 3 casos comunes en los que una story puede ser no estimable: Existe en el equipo de desarrolladoras incertidumbre técnica sobre cuestiones de implementación, existe incertidumbre sobre el dominio del negocio, o la historia puede ser demasiado grande.

En caso de duda del dominio, lo recomendable es escribir nuevamente la story, ya que los desarrolladores necesitan tener un entendimiento general de la story para poder implementarla.

En el caso de duda técnica, puede ser que el equipo de desarrolladores nunca haya tratado con cierta tecnología, por lo que no hay forma de saber cuánto se demorará en implementarla. La solución a esto es desviar recursos del proyecto para armar una spike (concepto se explicará más adelante). A una spike se le da siempre un tiempo máximo de investigación que nos permitirá estimar la spike. De esta manera una story inestimable se vuelve dos stories: una spike que nos permitirá investigar y conseguir información, y luego una story completa.

Muchas veces si una story es muy grande, conviene escribir épicas, porque sirven de placeholder o recordatorios de que grandes partes del sistema tienen que ser discutidas posteriormente.

**Small (Pequeña):** El tamaño o peso de la story es importante, ya que, si son o muy grandes o muy pequeñas, no se pueden incluir adecuadamente a la hora de planear. Las épicas son difíciles de tratar porque contienen múltiples stories. Una story debe poder ser consumida en una iteración.

**Testeable**: Las stories deben ser escritas de forma tal que se puedan testear. Si las mismas pasan sus test de manera exitosa se prueba que la story fue desarrollada exitosamente. Si la story no puede ser testeada, los desarrolladores nunca sabrán si han finalizado la implementación de la misma. Generalmente las stories que no pueden ser probadas provienen de requerimientos no funcionales. Pero esto está mal, ya que las stories son imposibles de testear.

Además, siempre que sea posible, se debe automatizar el proceso de prueba de las stories. Incluso surgió un enfoque llamado Test Driven Development, en donde lo primero que se hace es escribir las pruebas. Evitar adjetivos como rápido, manejar, limpio, etc.

**Pruebas De Aceptación en las User Stories**

Una razón para escribir las pruebas de aceptación es para expresar detalles que se hayan rescatado de la conversación entre el PO y los desarrolladores. Es decir, las pruebas de aceptación se usan para rellenar los detalles de una story.

El testeo se hace en un proceso de dos pasos: Primero se toma nota de los futuros test en la parte trasera de la tarjeta de la user story. Y luego, dichas pruebas son transformadas en pruebas completas que son utilizadas para demostrar que la user story ha sido implementada de manera completa.

Las pruebas de aceptación proveen un criterio básico para determinar si una story ha sido implementada completamente. Esto nos sirve para evitar poner demasiado o muy poco tiempo y esfuerzo en la story.

Es una buena idea escribir las pruebas antes de programar, ya que las mismas le brindan al desarrollador una gran cantidad de información

**Épicas y Spikes**

Generalmente estas caen dentro de dos categorías: Una story compuesta, o una story compleja.

Una story compuesta, es una épica que involucra múltiples stories cortas. Hay numerosas maneras de desagregar una story compuesta. Una forma es separar según las posibilidades que presente, por ejemplo, crear, editar, eliminar. Otra forma es separar según los campos o conjuntos de campos, ej: Añadir y editar información sobre educación (involucra varios campos). Pero muchas veces se corre el riesgo de separar demasiado las stories y tener stories muy pequeñas.

Una story compleja, es inherentemente grande, y no puede desagregarse en un set de stories consistentes. Si la story es compleja debido a la incertidumbre asociada a ella, es conveniente separar la story en dos: una investigativa y otra para desarrollar la funcionalidad. De esta manera se crea una spike, y se debe definir una investigación y un tiempo de duración de dicha investigación. Recién cuando se finaliza la investigación se arma la story que surge del spike. Generalmente es conveniente poner la spike en una iteración, y la story que surja de la misma en una o más iteraciones subsecuentes.

Esto también tiene la ventaja de que permite al PO priorizar la investigación por separado de la funcionalidad

Combinar Stories: Muchas veces las stories son demasiado pequeñas, tanto que los desarrolladores no quieren escribir la story y estimarla, porque hacerlo demora más que implementarla. Ejemplo: reportes de errores y cambios en la UI. Una forma de resolver este problema es combinar las pequeñas stories en una más grande. Se le otorga un nombre, se estima y se calendariza.

**Spikes**

Las spikes son un tipo especial de story utilizada para eliminar el riesgo y la incertidumbre en una user story del proyecto. Hay varias razones para utilizar spikes:

1. El equipo no tiene conocimiento sobre el domino en el que está trabajando.
2. La story puede ser demasiado grande para ser estimada, y el equipo puede usar una spike para analizar el comportamiento y dividir la story en varias estimables.
3. La story puede contener incertidumbre técnica, y el equipo tiene que hacer investigación para poder estimarla.
4. La story puede contener riesgo funcional, ya que puede entenderse la intención de la story, pero no está claro como el sistema debe interactuar con el usuario para alcanzar valor.

Spike Técnica: Es utilizada para realizar investigaciones en el dominio de la solución. Ej: Evaluar si construir o comprar una funcionalidad, evaluar la performance de una nueva user story, evaluar una tecnología que se pueda usar para implementar una funcionalidad.

Spike Funcional: Se usan cuando hay incertidumbre de como el usuario interactuará con el sistema. Para resolverlas se suele utilizar prototipos.

Algunas user stories requieren de ambos tipos de spikes.

**Lineamientos para Spikes**

Estimables, demostrables y aceptables: Como todas las stories, los spikes son colocados en el backlog, estimados y colocados para entrar en una iteración. El resultado de la spike es un artefacto, un reporte, información. Se utiliza para resolver la incertidumbre de la story a partir de la cual surgió el spike. Siempre la salida de un spike tiene que ser demostrable al equipo, debe aportar visibilidad, es decir, resultados claros y visibles, que permitan al equipo tomar decisiones. Como toda otra story, los spikes son aceptados por el PO, cuando los criterios de aceptación para dicho spike han sido completados.

La excepción, no la regla: Toda story tiene incertidumbre y riesgos. El objetivo del equipo es aprender a aceptar y resolver cierta incertidumbre en cada iteración. Los spikes deben dejarse para incógnitas más críticas y grandes, y deben ser utilizados como última opción.

Implementar la spike en una iteración separada de las historias resultantes: Salvo que el spike sea pequeño y sencillo, y sea probable encontrar una solución rápida, en cuyo caso, el spike y la story pueden incluirse en la misma iteración.