Contenido

[Planificando entregas con Scrum (I) 2](#_Toc289535049)

[Estimando en Puntos de Historia 4](#_Toc289535050)

[**Relatividad** 4](#_Toc289535051)

[**Tamaño** 4](#_Toc289535052)

[**Velocidad** 5](#_Toc289535053)

[**Tareas** 5](#_Toc289535054)

[**En Conclusión…** 5](#_Toc289535055)

[Experiencias implantando SCRUM 7](#_Toc289535056)

[SCRUM (I) 12](#_Toc289535057)

[SCRUM (II) 14](#_Toc289535058)

[SCRUM ( III ) 17](#_Toc289535059)

[Definiendo historias de usuario en Scrum 20](#_Toc289535060)

# Planificando entregas con Scrum (I)

En el post que nos ocupa vamos a ver los pasos para realizar la planificación de las diferentes entregas o releases de un proyecto, utilizando Scrum (aunque este proceso podría fácilmente aplicarse a cualquier metodología iterativa).

Lo que se quiere conseguir es crear una planificación a más largo plazo, de más duración que una iteración y que abarque las diferentes entregas del proyecto (que pueden ser varias si se realiza por fases o puede ser una única entrega), y que permita definir una previsión de que funcionalidades se podrán tener finalizadas en una determinada fecha.  
  
En su estado más básico, una planificación de entrega o de proyecto debería contar con como mínimo la siguiente información:

* Una lista de historias de usuario, a ser posible estimadas y priorizadas (product backlog)
* Los miembros del equipo que participan en el proyecto (desarrolladores, analistas, administradores de bbdd...)
* Fecha de inicio de la primera iteración y fecha de fin de la última iteración.

Es importante indicar que este nivel de planificación no se debe caer en el error de intentar entrar en el detalle de que personas van a realizar que tareas, en que secuencia se van a realizar las mismas etc... Estos detalles deben dejarse para la planificación de cada iteración, momento en el que las historias de usuario se dividen en tareas a nivel técnico.

Los pasos generales para realizar la planificación de las entregas son los siguientes:

* **Determinar los criterios más relevantes para el éxito del proyecto,** por ejemplo entregar antes de una fecha determinada, tener un conjunto mínimo de funcionalidades pero una fecha flexible, una combinación de ambas...
* **Realizar la estimación de las historias de usuario,** concretamente de aquellas que tienen una alta probabilidad de formar parte de la siguiente entrega. Este proceso se realiza entre el Product Owner y los miembros del equipo (¡¡Solo hace estimaciones el equipo!!) y se estima en puntos de historia o días ideales (y usando alguna técnica estilo Planning Poker)
* **Seleccionar la duración de las iteraciones,** en base a la duración del proyecto. Lo normal es de 2 a 4 semanas, pero en proyectos de larga duración se pueden realizar iteraciones más largas (aunque yo no lo recomiendo) o en proyectos muy cortos se pueden llegar a hacer iteraciones de una única semana.
* **Realizar una estimación de la velocidad,** en puntos de historia (o días ideales) del equipo. Si este ya ha trabajado junto se puede utilizar la velocidad media en anteriores iteraciones, si no lo mejor es hacer una estimación con alguna de las técnicas existentes para ello e ir ajustándola a medida que avance el proyecto.
* **Priorizar las historias de usuario,** de manera que siempre esté claro cuál es la siguiente a desarrollar. Una buena manera de asegurar esto es utilizando el enfoque de Henrik Kniberg en su muy recomendable [Scrum and XP For The Trenches](http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf), donde utiliza el atributo importancia, en lugar de prioridad. La priorización la decide el Product Owner, pero es aconsejable que escuche la opinión de los miembros del equipo, sobre todo en cuestiones relativa a la secuenciación de las historias de usuario.
* **Seleccionar las historias y la fecha de entrega,** en base a la velocidad y a la estimación realizada. Si el factor más importante en el proyecto es la fecha de entrega, lo que se hará será calcular el conjunto de funcionalidades que se prevén tener listas para esa fecha, si por el contrario, el factor importante son las funcionalidades mínimas, se deduce la fecha estimada de finalización.
* **Comprobar que se cumplen los criterios de éxito del proyecto**. Si no es así se debe volver a los procesos de estimación y priorización, haciendo los cambios necesarios (subdividir historias en otras más pequeñas, cambiando la priorización...) hasta conseguir dar con un grupo de funcionalidades para la entrega que cumpla con los criterios de éxito.

Otra cosa a tener en cuenta durante el proceso es la decisión de determinar por adelantado que funcionalidades se desarrollarán en cada iteración, o por el contrario sólo se deciden las funcionalidades a entregar en la iteración más próxima (de entre todas las planeadas para la entrega)

Cómo punto final, remarcar que es importantísimo que la planificación no sea un "papel colgado en la pared", sino que hay que ir actualizándola y revisándola con cierta frecuencia,de manera que se pueda conocer rápidamente las desviaciones o los cambios que ocurren. Una buena opción es re planificar al inicio de cada iteración, justo antes de la planificación de la misma.  
  
Aprovecho para hacer referencia al post anterior [Calendario de Sprint](http://devnettips.blogspot.com/2009/12/calendario-de-sprint.html) relacionando el concepto aquí explicado, con una de las reuniones propuestas en el mismo. Concretamente, la planificación de entregas seria lo que en el calendario llamábamos Project Planning (aunque en algunas organizaciones podría ser una fusión de las dos primeras reuniones explicadas, Project Review y Project Planning)  
  
En el siguiente Post, mostraré un ejemplo práctico de cómo realizar la Planificación de Entregas de un proyecto sencillo.

# Estimando en Puntos de Historia

Después de una interesante discusión producida ayer en twitter y en la lista de [agile-spain](http://groups.google.com/group/agile-spain) sobre la estimación en puntos de historia, he estado dándole vueltas al tema y me da la sensación de que es algo que mucha gente cuando empieza con Scrum no tiene demasiado claro. Lo viví en primera persona cuando me tocó supervisar la implantación de Scrum en mi última empresa, llegando incluso a tener discusiones grandes sobre porqué debíamos estimar así y no cómo se había hecho “toda la vida”… Así que me he decidido a aportar mi granito de arena y tratar de explicar de manera sencilla en qué consiste esto de los puntos de historia y qué beneficios aporta.

Antes de empezar, simplemente explicar, para quien no lo sepa, que los puntos de historia es la medida de estimación utilizada cuando se trabaja con Historias de Usuario, una de las técnicas más utilizadas al trabajar con la metodología Scrum.

**Relatividad**

En primer lugar  los PH son relativos no absolutos. Es decir estimamos en función de la diferencia relativa entre una historia y otra. Si a una historia le damos 1PH, y a otra 3PH, esto únicamente indica que una es el triple que la otra. ¿Pero el triple de qué? Pues eso lo veremos después. Además, la relatividad se aplica entre diferentes equipos, es decir que 3PH para un equipo no tienen que significar lo mismo para otro equipo (aunque trabajen en el mismo proyecto, no importa!!!)

De ahí viene el utilizar la famosa escala de  0, 1, 2, 3, 5, 8, 13, 40.., ya que lo que interesa no es el valor absoluto, únicamente el orden de magnitud, además de que nos ayuda a evitar la tentación de estimar hasta el más mínimo detalle: ¿Qué importancia tiene estimar 40 o 42? Cuanto más grande es la historia a estimar, menos certeza y precisión tenemos sobre la misma, con lo cual la escala nos ayuda a ser conscientes de esa imprecisión.

**Tamaño**

Cuando se estima en puntos de historia, lo que se intenta es dar un valor del TAMAÑO de la historia. ¿Y qué implica Tamaño? Pues yo entiendo que una combinación de:

* Complejidad: No es lo mismo implementar una alta en un formulario que, por ejemplo, un algoritmo de inteligencia artificial para un juego
* Esfuerzo: ¿Es una tarea que tengo que repetir en muchos sitios? ¿Me va a llevar mucho tiempo aunque sea una cosa fácil?
* Riesgo: ¿Estamos trabajando con una tecnología desconocida? ¿Tiene la gente del equipo experiencia en el negocio?

Utilizando cómo métrica el tamaño, sacamos al tiempo de la ecuación, de manera que evitamos la tentación de buscar conflictos del tipo “esta historia debería estar hecha en 3 días, y tu llevas 4”, o cosas de este estilo que suelen aparecer cuando se utiliza la métrica del tiempo.

Entonces, alguien se preguntará, ¿cómo les decimos a nuestros clientes cuando tenemos listo el producto? Eso lo vamos a ver en el siguiente apartado.

**Velocidad**

La velocidad es el factor que nos sirve para ver cuánto trabajo somos capaces de entregar en cada sprint. Si nuestra velocidad es 15PH, estamos diciendo que entregamos 15PH de media en cada iteración. Con esto conseguimos ese factor de conversión de los puntos de historia a fechas, es la manera que tenemos de poder decirle a nuestro cliente cuando estimamos que nuestro producto estará listo (ojo, es una estimación, no hay que caer en el típico error de considerar que las estimaciones son contratos!!)

Hay que tener cuidado con esto, ya que se puede caer en la tentación de hacer cuentas y empezar a pensar en tiempo en lugar de estimaciones (1 tío 8 horas = 1PH). Esto aunque en media sea cierto no lo va a ser para todas las historias, e incluso es posible que haya historias de 1 punto, que por ejemplo se tarden más en hacer que una de dos puntos. Por suerte, la velocidad actúa también cómo factor de corrección y a la larga, todas las historias que estén estimadas con los mismos PH van a caer más o menos en la misma zona (la distribución aproximada debería tener forma de campana de Gauss)

**Tareas**

Conviene aclarar que lo que he dicho hasta ahora se refiere a la estimación de historias, no a la de las tareas en las que descomponemos dicha historia durante la planificación de Sprint. No es el objetivo de este post entrar en este tema, así que sólo diré que mi recomendación para la estimación de tareas es:

* No estimarlas
* O dividir la historia en tareas que ocupen aproximadamente un día

**En Conclusión…**

* Los puntos de historia permiten abstraernos del tiempo a la hora de realizar estimaciones, de manera que podemos centrarnos en aspectos como la complejidad o el riesgo de la historia, y valorarla en función de eso.
* Al indicar el tamaño relativo de una historia respecto a otra, es sencillo realizar estimaciones triangulando, con lo cual el proceso, con el tiempo, se vuelve muy eficiente.
* Los PH evitan la especificación al detalle, poniendo de manifiesto el aumento de incertidumbre de una especificación a medida que esta aumenta.
* La velocidad actúa como factor de corrección de las desviaciones en las estimaciones y nos permite realizar la conversión a fechas para poder dar plazos de entrega a nuestros clientes o superiores cuando sea necesario.

A partir de aquí lo que toca es experimentar y probar, e intentar encontrar la medida de estimación que mejor se adapte a nuestro equipo, aunque yo sin duda recomiendo encarecidamente probar con los PH!

Ya sabéis, cualquier comentario será bienvenido!!

# Experiencias implantando SCRUM

En este artículo vamos a tratar de explicar nuestra experiencia implantando Scrum y prácticas ágiles en general en nuestro proceso de desarrollo de software. Hemos divido el artículo en diferentes capítulos que explican cada una de las fases por las que hemos pasado. Esperemos que nuestra experiencia os ayude de alguna manera. Nos gustaría remarcar que hablamos en nombre de todo el grupo de trabajo, que ha sido el que, trabajando todos unidos en una misma dirección, ha tirado adelante esta historia.  
  
  
**ANÁLISIS**  
  
Después de un par de años trabajando en nuestra empresa, vimos que necesitábamos una manera mejor de trabajar. El equipo era demasiado especializado, cada persona tenía conocimiento sobre áreas muy concretas del sistema y esto penalizaba mucho las ausencias de algunas personas. Además a pesar de que al principio se trabajaba de manera relativamente tranquila poco a poco, a medida que los clientes aumentaban, la dinámica fue degenerando hasta convertir el trabajo diario en un “apaga fuegos”, solucionando bugs de anteriores versiones, dando asistencia, he intentado desarrollar nuevas funcionalidades en tiempo récord. Además existía una paralelización excesiva de las tareas (se empezaban muchas de golpe y pocas se acababan en un tiempo razonable), los requisitos cambiaban con facilidad al igual que las prioridades y había poca comunicación entre el equipo, tanto de las tareas que realizaba cada uno cómo del conocimiento. Algo que sí que teníamos era un seguimiento del trabajo exhaustivo por parte de nuestra responsable, que también estaba preocupada por mejorar nuestra manera de trabajar.

**DESCUBRIMIENTO**  
  
Una vez que vimos que teníamos que buscar otra manera de trabajar, fuimos investigando por foros de internet, páginas web, experiencias de conocidos hasta encontrar un documento que cambió completamente nuestra visión del desarrollo. El documento es el conocido [“Scrum y XP desde las trincheras”](http://www.proyectalis.com/2008/02/26/scrum-y-xp-desde-las-trincheras/) y en él descubrimos conceptos tan importantes como Scrum, Integración Continua o testeo unitario. Algunos de estos conceptos era nuevos para nosotros, otros ya los conocíamos pero no éramos conscientes del impacto que podían tener en nuestro día a día. El hecho de conocer la experiencia de gente que trabajaba en proyectos grandes e importantes y que consideraban esto vital, nos dio el empujón definitivo para emprender esta aventura.

**PRIMEROS PASOS**

Cuando tuvimos claro por dónde tirar empezamos a hacer una prueba piloto con un grupo reducido de personal, en concreto tres personas. Lo primero que hicimos fue montar un repositorio de código (con Subversion) y empezar a utilizar Scrum para nuestra organización interna, todo esto a espaldas de nuestros jefes. Antes de llenarle la cabeza con ideas “extrañas y novedosas” que pudieran provocar su rechazo creímos conveniente experimentar por nuestra cuenta un tiempo, para intentar mostrarles algo sólido, con técnicas ya probadas y con resultados que respaldaran nuestra idea.  
  
Nuestros primeros pasos con Scrum consistieron básicamente en aglutinar una serie de tareas (o historias) que teníamos en cola (¿product backlog?), priorizarlas, realizar la división en subtareas, hacer una estimación (utilizando planning poker) y la realización de las reuniones diarias.

Aunque era todo muy “precario”, nuestra primera impresión con Scrum fue muy positiva. De tener a un equipo al que tan sólo se le controlaba la faena a intervalos relativamente grandes de tiempo, con la consiguiente tardanza en aplicar cambios de diseño o requerimientos y la lentitud en detectar errores, pasamos a tener un equipo que cada día compartía la información del trabajo realizado y al que era muy fácil orientar a resultados. Por otra parte, la utilización de un repositorio de código hacía que se eliminaran las tediosas sesiones de juntar versiones de diferentes programadores, ejercicio que era un generador de bugs terrible.  
  
  
**EXPANDIENDO LA SEMILLA**

Una vez vimos que íbamos por la senda correcta, y que Scrum podía ayudarnos a mejorar nuestro trabajo, nos dispusimos a introducir el gusanillo de las metodologías ágiles al resto del grupo de desarrolladores, todavía a espaldas de nuestros responsables. No queríamos decirles nada hasta ver que esta técnica era válida para un grupo grande de personas, aunque nuestra responsable directa ya supiera algo de todo esto, pues es complicado esconder ciertas cosas. Cómo queríamos que el grupo se sintiera implicado en la toma de la decisión y no queríamos imponer nada, hicimos una presentación al grupo mostrándoles la manera de trabajar que teníamos ahora con nuestros diferentes gestores y a la vez explicando cómo contrapunto la gestión que se podría llevar a cabo con una metodología ágil (en concreto Scrum). Una vez acabada la presentación les preguntamos si creían que la manera actual de trabajar era una buena manera y, obviamente, la respuesta fue negativa; todo el mundo creía que se podían hacer las cosas mejor, que nuestra gestión del tiempo era deficitaria y que el hecho que la toma de decisiones se centrara en nuestra responsable, que por su parte tenía un exceso de trabajo, hacía que el desarrollo se viera penalizado. Por suerte nuestra, esto coincidió con la decisión de nuestra responsable de delegar todo el trabajo que tenía que ver con desarrollo en nuestras manos, decisión que vino apoyada por el hecho que viera que teníamos capacidad para auto organizarnos. Así que decidimos proponer al grupo un cambio en la manera de trabajar, básicamente centrándonos en dos aspectos:

* Desarrollo: Les propusimos empezar a utilizar buenas prácticas de programación de manera conjunta y coordinada -> Instaurando el repositorio de código y enseñando cómo utilizarlo, creando una guía de estilo para homogenizar el código de todo el departamento, instaurando el testeo unitario, configurando un servidor de integración continua, etc.
* Gestión: Les propusimos instaurar Scrum como nuestra metodología de trabajo.

De esta manera empezamos nuestros primeros sprints. Al principio nos centramos en las reuniones de inicio de sprint, en los seguimientos diarios y en pequeñas retrospectivas muy básicas. La reacción del grupo fue muy positiva y poco a poco empezamos con las demostraciones internas, aunque esto todavía tenemos que mejorarlo.  
  
**IMPULSO DEFINITIVO**

El impulso definitivo ha sido la participación de nuestra jefa en todo el proceso de Scrum, principalmente en la reunión de inicio de sprint y en las demostraciones y retrospectivas. Esto ha sido muy importante para poder definir de manera más detallada las tareas a realizar, para la decisión de que entra y que no entra en cada sprint y, sobretodo, en que viera que somos un grupo que trabaja unido para obtener resultados. Ha sido también vital hacerle ver que la paralelización de tareas es contraproducente y tan sólo lleva a tener muchas tareas sin acabar al final del sprint (que es lo que nos pasaba en los primeros sprints). Ahora todo el equipo se centra en una historia de usuario (a lo sumo dos) y a final de sprint siempre podemos mostrar un producto con incremento de valor sobre la iteración anterior.  
  
  
**IMPEDIMENTOS**  
  
Son muchos los impedimentos que nos hemos ido encontrado a lo largo de nuestra aventura, y seguramente nos encontraremos muchos más. Vamos a intentar exponer en forma de lista los que consideramos más importantes:

* Dificultad a la hora de estimar: Aunque cada día afinamos más, todavía nos falta un buen trecho para llegar a hacer buenas estimaciones de las historias de usuario. Esto normalmente se ve reflejado en que conseguimos realizar menos historias de las previstas. Afortunadamente, el hecho de que últimamente el equipo se centre en menos historias a la vez ha hecho que se minimice el impacto de este impedimento.
* El equipo no sólo se dedica a programar. Nuestro equipo no es un equipo de desarrollo al 100%. También hace testeo, instalación en el cliente, resolución de incidencias, etc. Esto hace que nuestras estimaciones se vean penalizadas por una carga de trabajo no previsto que retrasa el desarrollo.
* Incomprensión por parte de otros gestores. La respuesta de nuestra jefa a nuestra propuesta ha sido espléndida, nos apoya al 100% e intenta colaborar en todo lo posible, pero no sólo trabajamos con ella. El hecho que haya en nuestra empresa gente con mentalidad tradicional respecto al desarrollo de software (o incluso sin ninguna mentalidad), hace que a veces choquemos frontalmente con su forma de dirigir los proyectos y que sea complicado realizar nuestro proceso productivo tal cómo nos gustaría.
* Respeto a las iteraciones: En ocasiones no hemos podido llevar a cabo sprints completos, debido a “grandes marrones” que llevaban tiempo aparcados y que de golpe se deben solucionar, cambiando de un día para otro (a veces justo al día siguiente de la reunión de inicio) todas las tareas programadas.
* Grandes aprendizajes. Todos nosotros somos nuevos en el mundo de la gestión de proyectos informáticos y, aunque intentamos aplicar todas las técnicas de manera razonable y a pasos pequeños, hay muchas cosas que aprender, tanto técnicas (a configurar un servidor de integración continua, a programar con testeo unitario, a realizar testeo funcional, etc.) como metodológicas. Por suerte, cada vez tenemos más experiencia y vamos dando pasos más grandes y seguros.
* Reticencia de algunos desarrolladores. Algunos desarrolladores veían algunas de estas técnicas como algo que disminuía su velocidad de trabajo, cosa que se veía reforzada por el hecho de trabajar con los gestores tradicionales que antes comentábamos. Aquí nuestra tarea “evangelizadora” ha sido importante, haciendo charlas, proponiendo lecturas, analizando artículos, etc. Al final, hemos podido convencer a todo el mundo y esto se ha notado en el día a día.
* Dificultad para realizar algunas prácticas: Aunque parezca increíble, practicas básicas de Scrum cómo poder estar sentado todo el equipo junto, o tener una sala con el panel de scrum donde poder realizar las reuniones diarias son “difíciles” de llevar a cabo en algunas empresas.

**EL FUTURO**

Creemos que el futuro que nos espera va a ser mucho mejor. Todavía tenemos muchas cosas que mejorar: demostraciones, retrospectivas, reuniones de inicio… La preparación de las demostraciones nos llevan mucho tiempo (ganar experiencia con nuestra herramienta de ALM nos ayudará mucho) y tenemos poco tiempo para hacer buenas retrospectivas.  
  
Esperamos también aumentar la visibilidad de nuestro proceso y hacerlo extensivo a toda nuestra sección, hacer más públicas nuestras demostraciones (por ahora internas), dar permiso a todo el mundo para que vean el avance de nuestro proyecto vía web con los diferentes reportes (tablero de scrum, burndown, tasa de resolución de bugs, etc.) e intentar impregnar al resto del departamento o incluso de la empresa de nuestra filosofía de trabajo.  
  
También esperamos que nuestros desarrollos sean cada vez más estables gracias a la incorporación masiva del testeo unitario, la integración continua, mejora en los procesos de testeo, etc. Esto hará que las interrupciones en el desarrollo por la irrupción de bugs también sea menor. El hecho de tener sistemas más estable también hará que podamos delegar trabajo de instalación y mantenimiento a otras secciones de nuestra empresa, ganando así tiempo para desarrollar más y mejor.

**CONCLUSIONES**  
  
Cómo conclusión podemos decir que nuestra experiencia con las metodologías ágiles en general y con Scrum en particular está siendo muy positiva, tanto en el ámbito personal cómo en el profesional (mejora en el trabajo diario al disminuir los impedimentos, alineación con el cliente para alcanzar objetivos, focalización en incremento de valor…). También es importante tener claro que Scrum y las metodologías ágiles en general no son una bala de plata. Son metodologías que requieren de un esfuerzo “extra” a corto plazo por parte de todos los roles implicados en el desarrollo del proyecto (esfuerzo que es posible que algunas personas no estén dispuestas a pagar) y que requieren de un contexto apropiado para su buen funcionamiento (hay tipos de proyectos que difícilmente encajarían en Scrum). Cómo dice Rodrigo Corral en este [artículo](http://geeks.ms/blogs/rcorral/archive/2007/06/06/nadie-dijo-que-no-exigiese-un-esfuerzo.aspx) “nadie dijo que no exigiera un esfuerzo”.  
  
Por nuestra parte os animamos de manera entusiasta a que hagáis el esfuerzo de conocer Scrum porque el esfuerzo vale realmente la pena.

# SCRUM (I)

**Qué es SCRUM**

Scrum es un marco de trabajo para el desarrollo ágil de proyectos de software, aunque lo podéis aplicar a cualquier otro ámbito. Si queréis gestionar vuestra vida de pareja con Scrum, adelante!  
  
Scrum, cómo buena metodología ágil y moderna divide el trabajo en iteraciones, a las que llama sprints, que generalmente se alargan entre 2 y 4 semanas. Cada sprint parte de un conjunto de historias de usuario (el llamado product backlog) de las que el equipo de desarrollo con la ayuda del product owner y en la reunión de inicio de sprint escogen aquellas que entraran en el sprint, decidiendo así el sprint backlog. A lo largo del sprint el equipo realiza una reunión diaria (sprint daily meeting) en la que cada miembro describe el estado de su tarea. Al final del sprint se realiza una demo para mostrar el avance del proyecto en el sprint y después una reunión para evaluar el sprint y su desarrollo ( sprint review meeting ). El objetivo de todo esto es poder entregar un incremento de valor del producto al cliente cada cierto periodo de tiempo de una manera regular.  
  
Si todo esto os suena a chino, no os preocupéis que ahora lo vamos detallando un poco.  
  
**Los actores de Scrum**

Antes de ver cuáles son los diferentes roles en un equipo de Scrum os voy a contar un cuento. Resulta que un día se encontraron una gallina y un cerdo en el campo. La gallina le dijo al cerdo: He tenido una idea. Vamos a montar un restaurante entre tú y yo. El cerdo le preguntó que cómo se llamaría el restaurante, a lo que la gallina le respondió que huevos con jamón. Cuando lo escuchó el cerdo desestimó la idea alegando que el estaría realmente comprometido con el proyecto, mientras que la gallina sólo estaría involucrada. Si aplicamos esto a Scrum, significa que los desarrolladores son los cerdos ( con perdón :D ) y son los que están realmente comprometidos con el avance del proyecto, ya que son los que lo desarrollan, mientras que la demás gente son gallinas, y aunque de alguna manera u otra están involucrados en el proyecto no están tan involucrados como los desarrolladores.

Scrum define básicamente tres roles: El equipo, el scrum master ( o facilitador ) y el product owner.  
  
Product owner

O propietario del producto. Es el encargado de mantener el product backlog ( la lista con las historias de usuario ) actualizado y priorizado. Representa al cliente y está presente con voz y voto en la reunión de inicio de sprint para explicar al equipo de trabajo las historias con detalle y llegar a un acuerdo con este sobre las que entran en el sprint backlog.

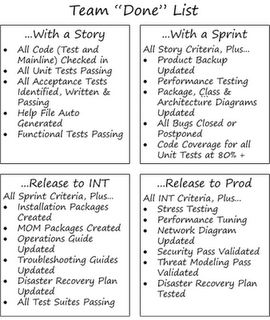
Equipo de desarrollo

Son los más cerdos de todos ( en el sentido scrumiano de la palabra ). Son los responsables de desarrollar el proyecto. Se reúnen con el product owner para aclarar todas las dudas sobre las historias, estiman lo que van a tardar en hacerlas y se reúnen cada día con el scrum master para controlar el avance del proyecto. Al finalizar el sprint, realizan la demo para mostrar el resultado de su trabajo y se reúnen con el scrum master para hacer la evaluación del sprint.  
  
Scrum master

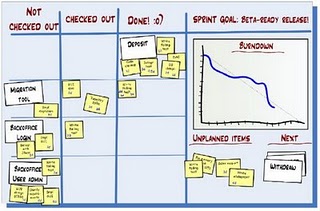
O facilitador. Es el encargado de hacer todo lo posible para que el equipo de desarrollo pueda realizar su faena de la mejor de las maneras. Es su representante en el exterior. Se encarga de eliminar todos los problemas que pueda tener el equipo y de hacer que las reglas de Scrum se cumplan.  
  
El resto de personajes que puedan estar en el proceso son las gallinas de nuestra historia. Pueden asistir a las reuniones pero no tienen voto en ninguna de ellas y voz en muy pocas. Entre estas gallinas podemos encontrar a otros grupos de desarrollos, gente de la empresa interesada en el proyecto, comerciales, etc. Pueden dar su opinión, en ocasiones muy valiosa, pero no pueden interferir en el desarrollo del proyecto.

# SCRUM (II)

En este segundo artículo sobre Scrum veremos toda aquella "documentación" que generamos a lo largo de la gestión de un proyecto utilizando Scrum. Más allá de la propia documentación del proyecto cómo pueden ser manuales de usuario, archivos de ayuda, diagramas de diseño, etc., hablaremos de la documentación que se genera por el hecho de utilizar Scrum. Dado que Scrum es un marco de desarrollo ágil y cómo tal prefiere el software que funciona por encima de la documentación exhaustiva, veremos que la documentación que generamos no es muy amplia aunque, eso sí, muy útil.

**Product Backlog** El product backlog es el documento donde se describen las historias de usuario que se quiere que alguna vez estén presentes en nuestra aplicación. Es cómo la lista a los reyes magos. Este documento está mantenido por el *dueño del producto*, de echo mantener este documento es su principal tarea. Debe mantenerlo actualizado con las historias que él crea conveniente y, sobretodo, lo debe mantener priorizado. El equipo tiene que saber siempre cuáles son las tareas más importantes para nuestro cliente ( es decir, que tareas tienen un mayor retorno de la inversión ) para así poder centrarse en ellas y dar el máximo valor posible en cada iteración.  
  
Esta lista no tiene por qué estar completa al principio del proyecto, de hecho, dado que somos un equipo ágil y valoramos el cambio cómo una oportunidad, no nos importa que esta lista vaya variando a lo largo del proyecto. Tampoco hace falta que todos los elementos estén detallados de igual manera; los que tienen que estar más detallados son los que tengan mayor prioridad, los otros ya se irán detallando más adelante. De esta manera gastamos el tiempo del dueño del producto en las tareas más importantes a día de hoy, dejando las menos importantes para más adelante porque quien sabe si al final se van a realizar o se van a modificar.  
[](http://3.bp.blogspot.com/_ZaNkQuiMjcA/SRcm4a5sCII/AAAAAAAABtQ/aEQREwP_SNI/s1600-h/Lacey_1-sampledonelist.png)  
También es importante acordar la definición de completado ( Definition Of Done - DOD). Sobre este tema, hay un montón de literatura. Cómo muestra os adjunto una imagen sacada del artículo [*How Do We Know When We Are Done?*](http://www.scrumalliance.org/articles/107-how-do-we-know-when-we-are-done) de lo que entiende el autor por acabado:  
  
Como veis el autor define varios niveles de definición de acabado y es realmente detallista. Es importante tener una buena definición de acabado y que todo el mundo la tenga clara para evitar posibles desviaciones en nuestra estimación por no realizar todas las tareas necesarias.  
  
**Sprint Backlog**

Es el documento resultante de la reunión de inicio de sprint. En él se detallan lo máximo posible aquellas tareas que se van a realizar en el sprint actual. Se describen con profundidad las historias de usuario, se dividen en tareas y se estima su esfuerzo. Son estas tareas las que compondrán el sprint backlog. Durante el desarrollo del sprint, cada miembro del equipo seleccionará una tarea de entre las más prioritarias cuando acabe de realizar la actual. Esta lista la tendremos que ir manteniendo día a día en la reunión de sprint diaria actualizando el estado de las tareas ( no empezada, empezada, acabada ) y actualizando también las horas restantes de desarrollo. Esto nos permitirá mantener los siguientes documentos que pasamos a explicar.

**Tablero de**[](http://2.bp.blogspot.com/_ZaNkQuiMjcA/SRcrdJ7smpI/AAAAAAAABtY/bqcK7PWXOJY/s1600-h/scrum+dashboard.png) **scrum**

Este tablero nos indica el estado actual de nuestras tareas. Vemos que se compone de una cuadrícula compuesta por filas ( las historias del product backlog ) y columnas ( el estado de las tareas ). El equipo de desarrollo debe actualizar dicho tablero cada día en la reunión diaria y así todo el mundo puede tener una visión rápida y exacta de cómo se está desarrollando el sprint. También podemos incluir en el tablero las tareas que no estaban planeadas y que se han acabado realizando y las próximas historias del product backlog, por si acabamos las planeadas inicialmente.  
(Imagen extraída del libro Scrum y XP from the trenches de Henrik Kniberg)  
 **Sprint Burndown Chart**

En el tablero de sprint también pondremos el sprint burndown chart. Este gráfico nos indica de una manera muy visual cuál es la velocidad del equipo y nos permite vislumbrar si el equipo acabará todas las tareas en el tiempo marcado. Cada día, en la reunión de sprint, el equipo actualizará el tablero de sprint con los datos de avance de estas tareas. Estos datos de avance ( las horas que le quedan a cada tarea ) también se pondrán de manifiesto en el gráfico, haciendo una línea desde el punto de las horas que quedaban por hacer ayer y las que quedan por hacer hoy ( ya que hemos restado las que hemos echo ). Esto, si lo comparamos con la línea ideal que se dibuja desde el punto de máximas horas del día de inicio de sprint, al punto de horas cero en el último día de sprint nos da información sobre si el equipo avanza a la velocidad adecuada o no. En próximos artículos ya hablaremos de cómo analizar un gráfico de avance.

# SCRUM ( III )

En el artículo de hoy veremos las diferentes reuniones que se realizan a lo largo de cada uno de los sprints.

**Reunión de inicio de sprint**

Esta es la reunión de la que nos acordaremos más en el transcurso del sprint sino la hacemos correctamente. Se realiza al principio de cada sprint, a poder ser un lunes por la mañana, cuando la gente está bien despierta y sin los agobios de toda la semana de trabajo. A ella acuden el dueño del producto, el scrum master y el equipo de desarrollo. La faena del dueño del producto ( como ya hemos visto en anteriores artículos ) es llegar a la reunión con el product backlog priorizado y detallado en sus historias más prioritarias. En la reunión (que acostumbra a durar toda la mañana) el dueño del producto explica al equipo cada una de las historias que cree que van a entrar en el sprint y el entre todos se dedican a dividir la historia en las diferentes tareas que la van a componer. Este diálogo es importantísimo ya que ayuda mucho a detallar exactamente lo que se va a implementar en el sprint y evita que al final del mismo haya malentendidos entre el dueño del producto y el equipo de desarrollo. De la misma forma, ayuda a detallar la historia tal y cómo la entiende el dueño del producto, ya que el equipo puede creer que para implementar una tarea tiene que hacer un acceso a datos complicadísimo y en cambio el dueño del producto no lo cree necesario. Sin este diálogo, el equipo de desarrollo implementaría este acceso a datos, tardando un montón y haciendo código innecesario ya que el cliente no lo cree necesario.  
En esta reunión, una vez detalladas las tareas, el equipo las debe estimar. Hay varias maneras de estimar una tarea, de las cuales ya hablaremos algún día. La estimación también ayuda mucho a refinar las tareas. En la estimación puede quedar reflejado que el equipo de desarrollo cree que para una tarea se debe hacer un algoritmo muy complicado y en cambio el dueño del producto quiere algo más simple. La estimación del equipo saldría con un valor muy alto, valor que sorprendería mucho al dueño del producto y se iniciaría otro diálogo para delimitar el alcance de la tarea.

Cómo podemos ver, aquí queda de manifiesto una de las cosas más importantes de las metodologías ágiles: la comunicación. Es muy importante la comunicación entre los miembros del equipo, desde el dueño del producto hasta los desarrolladores.  
  
La reunión de planificación debe terminar decidiendo entre el equipo y el dueño de producto qué historias entran en el sprint. Esto lo podemos hacer también de varias maneras (que también veremos en otro artículo) pero básicamente de lo que se trata es de estimar la velocidad del equipo en el sprint (la cantidad de puntos de historia que puede hacer el equipo) y mirar qué historias de usuario entran en el sprint. Si todavía no conocemos la velocidad del equipo la podemos intentar deducir. Primero asumiremos que un punto de historia es el trabajo que puede hacer un desarrollador en un día de trabajo (no en un día ideal, sino en un día "normal”). Esta definición no es importante. Podéis utilizar la definición que queráis. Lo que es importante es que la definición la mantengáis constante a lo largo de la vida del proyecto. No puede ser que un día utilicemos días ideales y otro día días "normales".

Una vez aclarado esto, sumamos los puntos de historia que puede hacer el equipo. Por ejemplo, si somos un equipo de cinco personas, tres de las cuales están a tiempo completo y dos a mitad de jornada, tendremos que en tres semanas el equipo puede asumir 60 puntos de historia. A este valor, nosotros le aplicamos otro modificador, para intentar estimar los posibles impedimentos que podamos tener. Un valor normal puede ser un 70%. Por lo tanto, diremos que nuestro equipo es capaz de asumir 42 puntos de historia en un sprint. Teniendo ordenadas las historias por prioridad, podremos ir viendo qué historias nos entrarán en el sprint. Por ejemplo, si nuestras historias hemos estimado que nos ocuparán 6, 10, 8, 12, 4, 8 y 12 puntos de historia, decidiremos qué haremos las primeras 5 historias.  
  
Hay dos cosas más importantes que deben decidirse en esta reunión:

* *El objetivo del sprint.* Puede parecer superfluo pero es importante definir un objetivo del sprint para que el equipo sepa porqué está trabajando.
* *Definición de acabado.* Cómo vimos en el artículo anterior, es vital tener una definición común de acabado entre el dueño del producto y el equipo de desarrollo.

**Sprint diario**

El sprint diario es la herramienta que nos proporciona scrum para que el equipo pueda compartir conocimiento y se comunique a diario. En esta reunión ( de unos 15 minutos en general cómo máximo ) cada miembro del equipo debe contestar tres preguntas:

* Qué hice ayer.
* Qué haré hoy.
* Qué impedimentos tengo para realizar mi trabajo.

Esta reunión es muy importante por la visibilidad que da del proyecto y del trabajo de cada uno. Haciendo asistir a todo el equipo a la reunión evitamos que haya alguien que no haga nada durante un día. El hecho de tener que decir delante de todo el mundo que no se ha avanzado en nada, hace que la gente trabaje cada día.

Esta reunión también nos sirve para tener el control del avance del proyecto. Gracias al sprint diario podemos actualizar nuestro tablero de scrum ( real o virtual ) y ver cómo avanzan las historias, ver si tenemos demasiadas historias en paralelo, demasiadas en testeo, etc. Lo otro que podemos actualizar gracias a la esta reunión es el gráfico de burndown, que nos da una información muy visual del estado del proyecto y, sobretodo, de la velocidad del equipo, pudiendo ver de un solo vistazo si se llegará a final de sprint con todas las tareas realizadas o no.

**Demostración**

La penúltima reunión del sprint es la demostración. En la demostración el equipo de desarrollo debe mostrar a todo el mundo los avances realizados en este sprint, es decir, el incremento de valor que han dado al producto. Esto es muy importante porque el equipo se obliga a tener un producto que se pueda enseñar, en un entorno de preproducción lo más fiable posible. También es importante porque el dueño del producto puede ver si lo que se ha implementado es lo que él pedía, si ha habido algún malentendido, si todo va a la velocidad requerida, etc. A esta reunión también puede asistir otra gente de la empresa, cuya opinión puede ser tenida en cuenta por el dueño del producto para planificar las próximas historias a realizar. El equipo nunca debe sentirse influenciado por estas opiniones, ya que su interlocutor en cuestión de producto es el dueño del mismo.

**Retrospectiva**

Es la última reunión de cada sprint. En esta reunión se hace una revisión de cómo ha ido el sprint, qué ha ido bien y qué ha ido mal. Hay varias formas de hacer una retrospectiva. Una de las más habituales y rápidas es hacer una línea temporal en la pizarra e ir apuntando todo lo que se ha hecho en el sprint (reuniones, impedimentos, logros, etc.). Una vez hecho esto, cada miembro del equipo señala aquellas cosas que le parecen más positivas y aquellas que le parecen más negativas. Las positivas se intentan repetir sprint a sprint. Las negativas se estudian porqué han ido mal y se intenta ver qué se puede hacer para solventar la situación y hacer que vaya mejor en posteriores sprints.

**Mini reuniones de diseño**

Estas reuniones no son obligatorias pero nosotros recomendamos hacerlas para no sobrecargar en demasía las reuniones de revisión diarias. Su cometido es comentar entre varios miembros del grupo de desarrolladores cosas relativas a decisiones de diseño de la aplicación. Por ejemplo si el miembro A del equipo empieza hoy la tarea "backup de datos de la aplicación" y no tiene claro cómo hacerla, en lugar de empezar una discusión de cómo hacerlo en la reunión de revisión, los miembros implicados se reúnen a posteriori para hablar del tema.

# Definiendo historias de usuario en Scrum

Uno de los puntos que más interrogantes provocan en la gente que se acerca a Scrum es cómo dividir el desarrollo de un proyecto en historias de usuario, es decir en tareas lo más pequeñas posibles que aporten valor funcional al cliente.

En este artículo voy a tratar de explicar el enfoque que yo adopto cuando trato de describir un requisito funcional cómo historia de usuario. Obviamente no es la única opción, ni la mejor, de echo es probable que esté pasando un montón de cosas por alto y que muchos se lleven las manos a la cabeza, pero creo que puede ser útil cómo primera aproximación para aquellas personas que no tengan del todo claro un procedimiento a seguir.  
  
Vamos a tomar cómo ejemplo un problema de la asignatura de Especificación de Software de la Facultad de Informática de Barcelona ([Ingeniería del Software: Especificación. Especificación de sistemas orientats a objectes amb la notació UML. EDICIONS UPC](http://bibliotecnica.upc.es/edupc/locate4.asp?codi=IN019XXX)), que presenta el siguiente enunciado:

*Se quiere desarrollar un sistema sencillo de control de préstamos en una biblioteca. El sistema debe admitir el alta y la baja de socios y de libros. Los socios pueden pedir libros en préstamo, pero no se pueden tener más de tres libros en préstamo en un momento determinado. Los libros se han de devolver antes de un mes de la fecha del préstamo. Cada vez que un socio devuelve un libro después de la fecha de la devolución, se penaliza reduciendo en una unidad el número de libros que puede tener simultáneamente. Cuando llega a cero el socio se dé de baja automáticamente.*

Ahora supongamos que este problema nos lo ha planteado nuestro cliente, que quiere que le construyamos exactamente el sistema descrito en el enunciado ¿Cómo llegamos desde el mismo, a la lista de tareas del backlog? Por el momento vamos a obviar aquellas tareas que a simple vista no aportan valor al cliente pero que son del todo necesarias para el proyecto (definir e implementar la arquitectura del sistema por ejemplo) y vamos a centrarnos únicamente en los objetivos funcionales.

Lo que vamos a hacer va a ser un “clásico” análisis de requerimientos, intentando extraer del texto (lo que nos dice el cliente) las diferentes funcionalidades que debe tener el sistema. Para ello analizamos los párrafos del texto. A simple vista podemos extraer las siguientes funcionalidades:

De la siguiente frase: *El sistema debe admitir el alta y la baja de socios y de libros* extraemos las siguientes posibles historias de usuario:

* Alta Libro
* Baja Libro
* Alta Socio
* Baja Socio

Del párrafo siguiente: *Los socios pueden pedir libros en préstamo, pero no se pueden tener más de tres libros en préstamo en un momento determinado* obtenemos una única funcionalidad o historia

* Préstamo libro

En este párrafo: *Los libros se han de devolver antes de un mes de la fecha del préstamo. Cada vez que un socio devuelve un libro después de la fecha de la devolución, se penaliza reduciendo en una unidad el número de libros que puede tener simultáneamente* creo que podrían identificarse dos historias (es mi opinión, quizás otra gente opine que sólo hay una posible historia)

* Devolver libro
* Penalizar socio

En el último párrafo obtenemos otra posible historia: *Cuando llega a cero el socio se dé de baja automáticamente*. Aunque pueda parecer que es una historia repetida, yo considero que a nivel de valor al cliente es una funcionalidad diferente.

* Baja automática de socio

Aparte de las historias que aparecen más o menos explícitas en el código, quizás podrían deducirse otras historias cómo:

* Iniciar sesión en el sistema
* Cerrar sesión
* Alta usuario
* Baja usuario

La lista completa de historias que tendríamos sería la siguiente:

* Alta libro
* Baja libro
* Alta socio
* Baja socio
* Préstamo de libro
* Devolver libro
* Penalizar socio
* Baja automática de socio
* Iniciar sesión en el sistema
* Cerrar sesión
* Alta usuario
* Baja usuario

Si estuviéramos en una metodología clásica de desarrollo, en este punto deberíamos hacer los casos de uso para los requerimientos identificados, sus diagramas de secuencia, etc. En una metodología cómo Scrum, por definición, no es necesaria esta documentación (queda a elección de cada uno decidir que documentación necesita su proyecto) y lo que nos quedaría por hacer es definir e introducir las historias en el backlog. ¿Y cómo se define una funcionalidad en forma de historia de usuario? Pues la idea es expresarla desde el punto de vista del cliente, o del usuario que va a usar el sistema, con un lenguaje que pueda ser entendido perfectamente por ellos (nada de UML, OCL o demás lenguajes de especificación) y que no cree ambigüedades. Por ejemplo:  
  
**Historia**: Préstamo de libro

**ID**: 5

**Descripción**: Cómo cliente quiero que los socios puedan pedir prestado un libro, indicando su número de socio y la referencia del libro, siempre y cuando no tengan ya tres libros en préstamo en ese momento.

**Importancia**: 300

**Cómo probarlo**:

* Introducir un número de socio incorrecto y comprobar que se indica error
* Introducir un socio que ya tiene 3 libros en préstamo y comprobar que se indica error
* Introducir un libro del que no hay ejemplares y comprobar que se indica un error
* Introducir todos los datos correctos y comprobar que el número de ejemplares disponibles del libro disminuye y el número de préstamos del socio aumenta en uno.

Como vemos, en Scrum se detalla de manera poco explícita la funcionalidad, únicamente se explica a nivel de cliente. La ventaja es que no se fijan los detalles de la implementación hasta el momento en que se va a realizar (en la descomposición en tareas) con lo que se puede reaccionar más ágilmente ante los cambios de los requisitos o de las necesidades del cliente.  
  
Haciendo lo mismo con las demás funcionalidades, nos quedaría un product backlog similar al siguiente (se ha obviado la parte de test de las historias):

|  |  |  |
| --- | --- | --- |
| Historia | Importancia | Descripción |
| Alta libros | 600 | Dar de alta un libro en el sistema |
| Baja libros | 250 | Dar de baja un libro del sistema |
| Alta socio | 500 | Dar de alta un socio en el sistema |
| Baja socio | 300 | Dar de baja un socio del sistema |
| Pedir libro | 350 | Pedir prestado un libro |
| Devolver libro | 340 | Devolver un libro |
| Penalización socio | 330 | Penalizar al socio por retrasarse en la devolución |
| Baja automática | 320 | Dar de baja automáticamente de socio |
| Iniciar sesión en el sistema | 550 | Iniciar una sesión de usuario en el sistema |
| Cerrar sesión | 500 | Terminar sesión con el sistema |
| Alta usuario | 450 | Dar de alta un nuevo usuario en el sistema |
| Baja usuario | 400 | Dar de baja un usuario del sistema |

En este punto tenemos identificadas todas las tareas funcionales del proyecto, y se ha descrito cada una de ellas cómo una historia de usuario.  
  
Aunque ya tenemos completa la lista de funcionalidades, es obvio que el product backlog tal y como está no es suficiente para poder avanzar en el proyecto, ya que no aparecen todas aquellas tareas que no son funcionales (historias técnicas, requisitos no funcionales o como se le quieran llamar) pero que son necesarias para el desarrollo del proyecto. Pero la mayoría de estas tareas no aportan valor al cliente, entonces ¿Se deben incluir estas tareas en el backlog? ¿Y de qué manera? No vamos a discutir en este artículo el tema, ya que daría para muchos posts, pero a modo de apunte se podría gestionar de la siguiente forma:  
  
En proyectos que se están iniciando, es fácilmente justificable introducir tareas técnicas o no funcionales, ya que sin ellas el proyecto no existe. Así bajo la forma de Epics (o súper historias de usuario) se podrían englobar tareas tales cómo Implementar la capa de datos, implementar la capa de comunicaciones, etc. Estas tareas se llevarían a cabo en las primeras iteraciones, o incluso en una primera iteración mayor que las otras.  
  
En fases más avanzadas del proyecto, es difícil justificar tareas no funcionales, cómo la refactorización de módulos, instalación de servidores de pruebas, etc. Ya que el cliente querrá priorizar aquellas historias que le aporten valor. En este caso lo ideal es intentar que el cliente comprenda la importancia de las tareas a realizar, o si eso no es posible utilizar otras técnicas, cómo introducirlas dentro de alguna de las historias de usuario, o mantener una lista separada de historias técnicas con un porcentaje de tiempo no negociable con el dueño de producto.

Cómo hemos visto el proceso de definir historias de usuario es tan simple (o tan complicado) cómo realizar un análisis de requerimientos, proceso para el cual no hay ningún truco infalible y dependerá en gran medida de la experiencia y pericia del analista o analistas que se encarguen de él, y expresar las funcionalidades resultantes del análisis en un lenguaje que pueda ser fácilmente entendido por el cliente de nuestro proyecto.