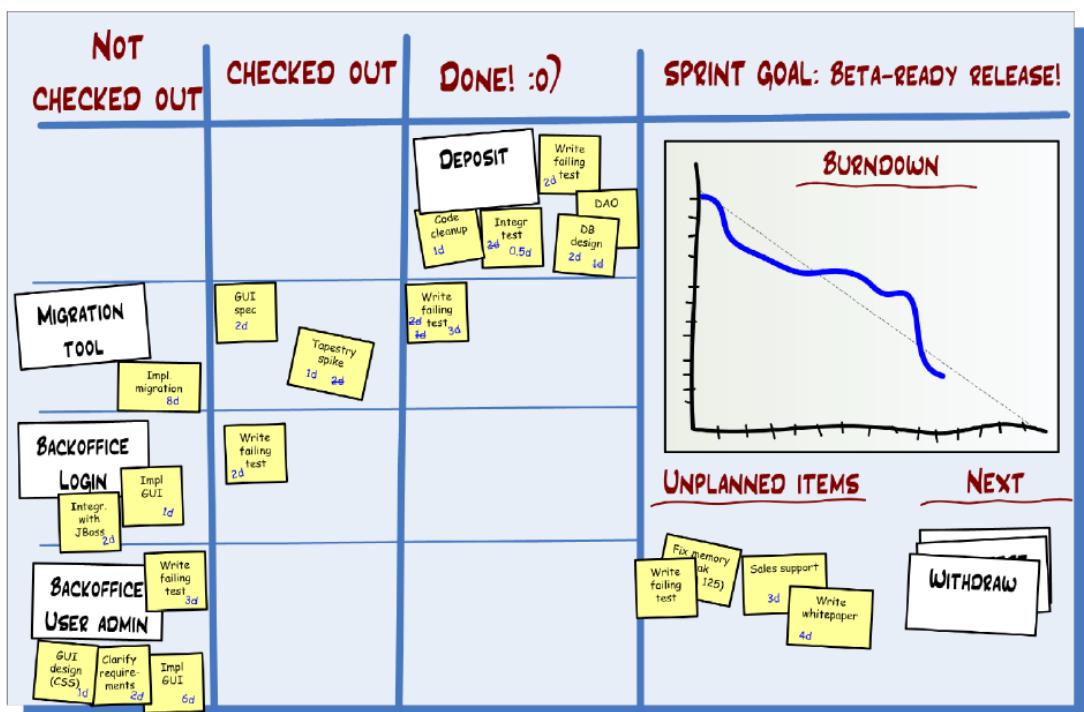


Una historia de guerra Ágil

SCRUM Y XP DESDE LAS TRINCHERAS

Cómo hacemos Scrum



Henrik Kniberg
Prólogos de Jeff Sutherland y Mike Cohn

InfoQ *Enterprise Software Development Series*

VERSION GRATUITA ON-LINE

(versión gratuita no-imprimible)

Si te gusta el libro, por favor, apoya al autor y a InfoQ
comprando la edición impresa:
<http://www.lulu.com/content/899349>
(sólo \$22.95)

Este libro está disponible por cortesía de



Traducido al castellano por



www.proyectalis.com

Este libro se distribuye gratuitamente en InfoQ.com y
en proyectalis.com.

Si ha recibido este libro desde otra fuente, por favor,
ayuda al autor y al editor registrándose en InfoQ.com.

Visita la página Web de este libro en:
<http://infoq.com/minibooks/scrum-xp-fromthetrenches>

Scrum y XP desde las trincheras

Como hacemos Scrum

Escrito por:
Henrik Kniberg

Versión Online Gratuita.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

© 2007 C4Media Inc

Todos los derechos reservados.

C4Media, editor de InfoQ.com.

Este libro es parte de la serie de libros sobre Desarrollo de Software Empresarial de InfoQ.

Para más información sobre cómo adquirir este u otros libros de InfoQ, por favor contacte con books@c4media.com.

Ninguna parte de esta publicación puede ser reproducida, almacenada en un sistema o transmitida mediante ningún medio electrónico, mecánico, fotocopia, recodificación, escaneado o de otras formas, excepto en las permitidas por las secciones 107 o 108 del acta de derechos de copia de 1976 de los Estados Unidos, sin el permiso por escrito del editor.

Las designaciones usadas por las empresas para distinguir sus productos se consideran marcas registradas. En todos los casos en los que C4Media Inc. es consciente de que existe un registro de marca, los nombres de productos aparecerán con la inicial en Mayúscula o TODO EN MAYUSCULAS. Los lectores, en cualquier caso, deberán contactar con las correspondientes empresas para más información acerca de las marcas registradas.

Editor Jefe: Diana Plesa

Portada: Dixie Press

Composición: Dixie Press

Traducción al castellano: Ángel Medinilla (info@proyectalis.com)

Datos de catalogación de la Biblioteca del Congreso:

ISBN: 978-1-4303-2264-1

Impreso en los Estados Unidos de América

Agradecimientos

El primer borrador de este libro solo me tomó un fin de semana, pero sin duda fue un fin de semana intenso (¡150% de factor de dedicación! :o).

Gracias a mi esposa Sophia y a mis hijos Dave y Jeremy por aguantar mi poca sociabilidad ese fin de semana, y a los padres de Sophia, Eva y Jörgen, por pasarse a cuidar de la familia.

Gracias también a mis colegas de Crisp en Estocolmo y a la gente del grupo scrumdevelopment de Yahoo por corregir los borradores y ayudarme a mejorar el libro.

Y, finalmente, gracias a todos mis lectores, quienes han proporcionado un constante flujo de comentarios útiles. Estoy particularmente contento de escuchar que este libro ha empujado a tantos de vosotros a darle una oportunidad al desarrollo Ágil de software!

Índice

AGRADECIMIENTOS	4
PRÓLOGO DE JEFF SUTHERLAND	10
PRÓLOGO DE MIKE COHN	12
PREFACIO - ¡HEY, SCRUM FUNCIONA!	13
INTRODUCCIÓN	14
Limitación de responsabilidad	16
Por qué he escrito esto	16
Pero ¿Qué es Scrum?	16
COMO HACEMOS PILAS DE PRODUCTO	17
Campos de historia adicionales	18
Como mantenemos la Pila de Producto a nivel de negocio	19
COMO NOS PREPARAMOS PARA LA PLANIFICACIÓN DE SPRINT	20
COMO HACEMOS LA PLANIFICACIÓN DE SPRINT	22
Por qué la calidad no es negociable	24
Reuniones de planificación de Sprint que duran, y duran...	25
Agenda de la reunión de planificación de Sprint	25
Definiendo la duración del Sprint	26
Definiendo la meta del Sprint	27
Decidiendo qué historias incluir en el Sprint	27
¿Cómo puede el Dueño de Producto alterar las historias que se incluyen en el Sprint?	28
¿Cómo decide el equipo qué historias incluir en el Sprint?	30
Por qué usamos tarjetas	35
Definición de “terminado”	38

Estimación de tiempos usando planning poker	39
Clarificando historias	41
Dividiendo historias en historias más pequeñas	42
Dividiendo las historias en tareas.	42
Definiendo el sitio y la hora para el Scrum diario	43
Dónde trazar la línea	44
Historias técnicas	45
Sistema de seguimiento de errores vs. Pila de Producto	47
¡Por fin acabó la reunión de planificación de Sprint!	47
COMO COMUNICAMOS LOS SPRINTS	48
COMO HACEMOS PILAS DE SPRINT	50
Formato de la Pila de Sprint	50
Cómo funciona el tablón de tareas	51
Ejemplo 1 – tras el primer Scrum diario	52
Ejemplo 2 – tras unos cuantos días	53
Como funciona el diagrama burn-down	54
Señales de alarma en el burn-down	55
Hey, ¿Qué pasa con la trazabilidad?	57
Estimando en días vs horas	57
COMO DISTRIBUIMOS LA SALA DEL EQUIPO	59
La esquina de diseño	59
¡Sienta al equipo junto!	60
Mantén al Dueño de Producto a mano	61
Mantén a los gerentes y <i>coachs</i> a mano	62
CÓMO HACEMOS SCRUM DIARIOS	63
Cómo actualizamos el tablón	63
Tratando con tardones	64
Tratando con “no se qué hacer hoy”	64

CÓMO HACEMOS LA DEMO DE SPRINT	66
Por qué insistimos en que todos los Sprints acaben con una demo	66
Lista de comprobación para demos de Sprint	67
Tratando con historias “indemostrables”	67
CÓMO HACEMOS RETROSPECTIVAS DE SPRINT	69
Por qué insistimos en que todos los equipos hagan retrospectivas	69
Cómo organizamos las retrospectivas	69
Difundiendo las lecciones entre los equipos	71
Cambiar o no cambiar	71
Ejemplo de cosas que pueden surgir en las retrospectivas	72
DESCANSOS ENTRE SPRINTS	74
Define tus umbrales de aceptación	76
Estimación de los elementos más importantes	77
Estimar la velocidad	78
Uniéndolo todo en un plan de entregas (<i>release plan</i>)	79
Adaptando el plan de entregas	80
CÓMO COMBINAMOS SCRUM CON XP	81
Programación por parejas	81
Desarrollo guiado por pruebas (TDD)	82
Diseño incremental	84
Integración continua	84
Propiedad colectiva del código	85
Espacio informativo	85
Estandarización de código	85
Ritmo sostenible / trabajo enérgico	86
CÓMO HACEMOS PRUEBAS	87
Probablemente no puedas renunciar a la fase de pruebas	87
Minimiza la fase de pruebas	88

Incrementar la calidad incluyendo encargados de pruebas en el equipo	89
Incrementar la calidad haciendo menos en cada Sprint	91
¿Deberían las pruebas de aceptación ser parte del Sprint?	91
Ciclos de Sprint vs. ciclos de pruebas	92
No sobrecargues el eslabón más débil de tu cadena	95
De vuelta a la realidad	96
CÓMO MANEJAR MÚLTIPLES EQUIPOS SCRUM	97
Cuántos equipos crear	97
¿Sprints sincronizados, o no?	100
Por qué introdujimos un rol de “guía de equipo”	101
Como asignamos personas a los equipos	102
¿Equipos especializados – o no?	103
¿Redistribuir equipos entre Sprints – o no?	105
Miembros a tiempo parcial	106
Como hacemos Scrum de Scrums	107
Intercalando los Scrums diarios	108
Equipos apagafuegos	109
¿Dividir la Pila de Producto – o no?	110
Ramificación del código	114
Retrospectivas multi-equipo	114
CÓMO GESTIONAMOS EQUIPOS DISTRIBUIDOS GEOGRÁFICAMENTE	116
Offshoring	117
Miembros de equipo que trabajan desde casa	118
LISTA DE COMPROBACIÓN DEL SCRUM MASTER	119
Comienzo del Sprint	119
Todos los días	119
Final de Sprint	119

EPÍLOGO	120
LECTURAS RECOMENDADAS	121
SOBRE EL AUTOR	122

BOGDANOV

Prólogo de Jeff Sutherland

Los equipos de trabajo deben conocer los principios de Scrum. ¿Cómo se crea y se estima una pila de producto? ¿Cómo se transforma en una pila de Sprint? ¿Cómo se gestiona un gráfico de burn-down y se calcula la velocidad del equipo? El libro de Henrik es un “kit de inicio” con las prácticas básicas que ayudan a los equipos a avanzar de “intentar emplear Scrum” a ejecutar Scrum correctamente.

La ejecución correcta de Scrum se está convirtiendo en un factor cada vez más importante para los equipos que buscan inversión de capital. Como Coach Ágil de una firma de capital riesgo, ayudo en su objetivo de invertir sólo en compañías Ágiles que ejecuten las prácticas Ágiles correctamente. El Socio Senior del grupo pregunta a todas las compañías del portfolio si conocen la velocidad de sus equipos. Actualmente tienen dificultades para responder esta pregunta. Las oportunidades de inversión en el futuro requerirán que los equipos de desarrollo comprendan el concepto de **su velocidad de producción de software**.

¿Por qué es esto tan importante? Si los **equipos no conocen su velocidad**, el Dueño de Producto **no puede crear una hoja de ruta del producto** con fechas de lanzamiento creíbles. **Sin fechas de lanzamiento fiables**, la **compañía podría fracasar** y los **inversores perder su dinero**.

Compañías grandes y pequeñas, nuevas y viejas, con inversores o sin ellos, se enfrentan a este problema. En una discusión reciente en sobre la implantación de Scrum en Google que tuvo lugar durante una conferencia en Londres, pregunté a una audiencia de 135 personas cuantas de ellas estaban haciendo Scrum, y 30 respondieron positivamente. A continuación les pregunté si estaban haciendo desarrollo iterativo según el estándar de Nokia. El desarrollo iterativo es una parte fundamental del Manifiesto Ágil – liberar software funcional cuanto antes y frecuentemente. Después de años de retrospectivas con cientos de equipos Scrum, Nokia desarrolló algunos de los requisitos básicos para el desarrollo iterativo:

- Las iteraciones deben tener una duración fija de menos de seis semanas.
- El código liberado al final de la iteración debe estar testado por Aseguramiento de la Calidad (QA) y debe funcionar correctamente.

De las 30 personas que decían emplear Scrum, sólo la mitad dijeron estar cumpliendo con el primer principio del Manifiesto Ágil según los estándares de Nokia. Les pregunté entonces si cumplían con los estándares de Nokia para Scrum:

- Un equipo **Scrum debe tener un Dueño de Producto** y saber quién es esa persona.
- **El Dueño de Producto debe tener una pila de producto** con estimaciones creadas por el equipo.
- **El equipo debe tener un Gráfico de Burn-Down** y conocer **su velocidad**.
- **Nadie fuera del equipo debe interferir con el mismo durante un Sprint.**

De las 30 personas que hacían Scrum, sólo 3 cumplían el test de Nokia para equipos Scrum. Estos son los únicos equipos que recibirán inversiones de mis socios de capital riesgo en el futuro.

El valor del libro de Henrik reside en que, si sigues las prácticas que describe, tendrás una Pila de Producto, estimaciones para la Pila de Producto, un Gráfico de Burn-Down y conocerás la velocidad de tu equipo, junto con otras muchas prácticas esenciales para un Scrum totalmente funcional. Cumplirás con el test de Nokia y merecerá la pena invertir en tu trabajo. Si eres parte de una compañía tipo Start-Up, puede que incluso recibas inversiones de grupos de capital riesgo. Puede que seas el futuro del desarrollo de software y el creador de una nueva generación de productos software punteros.

Jeff Sutherland,
Ph.D., Co-Creador de Scrum

Prólogo de Mike Cohn

Tanto Scrum como Programación Extrema (XP) requieren que los equipos completen algún tipo de producto potencialmente liberable al final de cada iteración. Estas iteraciones están diseñadas para ser cortas y de duración fija. Este enfoque en entregar código funcional cada poco tiempo significa que los equipos Scrum y XP no tienen tiempo para teorías. No persiguen dibujar el modelo UML perfecto en una herramienta CASE, escribir el documento de requisitos perfecto o escribir código que se adapte a todos los cambios futuros imaginables. En vez de eso, los equipos Scrum y XP se enfocan en que las cosas se hagan. Estos equipos aceptan que puede que se equivoquen por el camino, pero también son conscientes de que la mejor manera de encontrar dichos errores es dejar de pensar en el software a un nivel teórico de análisis y diseño y sumergirse en él, ensuciarse las manos y comenzar a construir el producto.

Es este mismo enfoque en hacer en lugar de teorizar lo que distingue este libro. Que Henrik comprende que esto es evidentemente cierto desde el principio. No ofrece una pesada descripción sobre qué es Scrum. En vez de eso, nos remite a algunos sitios Web simples y salta directamente a describir cómo su equipo gestiona y trabaja su Pila de Producto. Desde ahí, pasa por todos los demás elementos y prácticas de un proyecto Ágil bien ejecutado. Nada de teorías. Nada de referencias. Nada de notas al pie. No hace falta ninguna. El libro de Henrik no es una explicación filosófica sobre por qué Scrum funciona o por qué deberías probar esto o lo otro. Es una descripción de cómo funciona un equipo Ágil bien gestionado.

Es por ello que el subtítulo del libro, “Cómo hacemos Scrum”, es tan adecuado. Puede que no sea la manera en la que *tú* haces Scrum, es cómo hace Scrum el equipo de Henrik. Podrías preguntarte por qué debería interesarte cómo hace Scrum otro equipo. Debería interesarte porque todos podemos aprender a emplear Scrum mejor oyendo historias de cómo lo han hecho otros, especialmente aquellos que lo están haciendo bien.

No hay y nunca habrá una lista de “mejores prácticas en Scrum”, porque el contexto de cada equipo y proyecto impera sobre cualquier otra consideración. En lugar de mejores prácticas, lo que necesitamos conocer son mejores prácticas y el contexto en que fueron exitosas. Lee suficientes historias sobre equipos con éxito y como hicieron las cosas y estarás preparado para todos los obstáculos que se te presenten en el uso de Scrum y XP.

Henrik proporciona un conjunto de mejores prácticas junto con el contexto necesario para ayudarnos a aprender mejor cómo emplear Scrum y XP en las trincheras de nuestros propios proyectos.

Mike Cohn

Autor de *Agile Estimating and Planning* y *User Stories Applied for Agile Software Development*.

Prefacio - ¡Hey, Scrum funciona!

¡Scrum funciona! Para nosotros, al menos (me refiero a mi actual cliente en Estocolmo, cuyo nombre omitiré en este documento). ¡Espero que funcione para ti también! Quizás este libro te ayude en tu camino.

Esta es la primera vez que he visto a una metodología de desarrollo (perdón, Ken, un *marco de trabajo*) funcionar directamente desde el libro. *Plug and Play*. Todos estamos contentos con ella: desarrolladores, encargados de pruebas y gerentes. Nos ha ayudado a salir de una dura situación y nos ha permitido mantener el enfoque y el impulso a pesar de las severas turbulencias en el mercado y las reducciones de plantilla.

No debería decir que me sorprendió pero, en fin, lo hizo. Después de digerir varios libros sobre el tema Scrum parecía bueno, pero casi demasiado bueno como para ser verdad (y todos conocemos el dicho “cuando algo parece demasiado bueno para ser verdad...”). Así que era, justificadamente, un poco escéptico. Pero después de emplear Scrum durante un año estoy suficientemente impresionado (y la mayor parte de la gente en mis equipos también) como para probablemente continuar usándolo por defecto en nuevos proyectos siempre que no haya una fuerte razón para no hacerlo así.

1

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Introducción

Vas a comenzar a usar Scrum en tu organización. O quizás has estado usando Scrum por unos meses. Conocer los principios básicos, has leído los libros, quizás incluso has conseguido tu Certificado de Scrum Master. ¡Enhorabuena!

Pero aun así te sientes confuso.

En palabras de Ken Schwaber, Scrum no es una metodología, es un marco de trabajo. Eso quiere decir que Scrum no te va a decir exactamente lo que debes hacer. Vaya.

Las buenas noticias son que te voy a decir exactamente como empleo Scrum, con una cantidad terriblemente dolorosa de detalles. Las malas noticias son que, bueno, **esto es sólo como empleo Scrum yo. No significa que tú debas hacerlo exactamente de la misma forma**. De hecho, probablemente **yo mismo haga las cosas de una manera diferente si me encuentro en una situación diferente**. La fuerza y lo duro de Scrum es que estás obligado a adaptarlo a tu situación específica.

Mi **aproximación actual a Scrum** es el resultado de un **año de experimentación con un equipo de desarrollo de aproximadamente 40 personas**. La empresa **estaba en una situación dura**, con **muchas horas extra, problemas de calidad severos, apagando fuegos constantemente, incumplimientos de fechas...** La empresa había decidido usar Scrum pero realmente no había completado la implementación, lo cual sería mi tarea. Para la mayor parte de la gente en los equipos de desarrollo, "Scrum" era simplemente un palabro de moda que se escuchaba por los pasillos de cuando en cuando, sin ninguna implicación para su trabajo diario.

A lo largo de un año, implementamos **Scrum en todos los niveles de la empresa, probamos diferentes tamaños para los equipos** (3 a 12 personas), **diferentes duraciones de Sprint** (2 a 6 semanas), **diferentes maneras de definir "terminado"**, **diferentes formatos** para las **Pilas de Producto** y **Pilas de Sprint** (Excel, Jira, tarjetas), **diferentes estrategias de pruebas**, **diferentes maneras de hacer las demostraciones**, **diferentes maneras de sincronizar múltiples equipos Scrum**, etcétera. También **experimentamos con prácticas XP**: **diferentes formas de hacer construcción continua, programación por parejas, desarrollo orientado a test**, etcétera, y como **combinar estas prácticas con Scrum**.

Este es un proceso de aprendizaje continuo, así que la historia no acaba aquí. Estoy convencido de que esta empresa seguirá aprendiendo (si se siguen haciendo las Retrospectivas de Sprint) y adoptando nuevas perspectivas sobre las **mejores formas de implementar Scrum en su contexto particular.**



Limitación de responsabilidad

Este documento no pretende representar “la forma correcta” de usar Scrum. Solo representa una forma de usar Scrum, el resultado de una mejora constante durante todo un año. Puede que incluso decidas que lo hemos hecho todo mal. Todo lo que contiene este documento refleja mi opinión personal y subjetiva, y en ningún caso una declaración oficial de Crisp o de mi actual cliente. Por esta razón he evitado específicamente usar el nombre de ninguna persona o producto.

Por qué he escrito esto

Cuando estaba aprendiendo sobre Scrum leí los libros relevantes sobre Scrum y desarrollo Ágil de software, me dejé caer por los foros y los sitios sobre Scrum, obtuve la certificación con Ken Schwaber, lo abrasé a preguntas y dediqué un montón de tiempo discutiendo con mis colegas. Una de las fuentes más valiosas de información, en cualquier caso, fueron las auténticas *historias de guerra*. Las historias de guerra convierten los Principios y Prácticas en...Bueno...Como Se Hacen Realmente Las Cosas. También me ayudaron a identificar (y a veces, a evitar) los errores típicos de novato al emplear Scrum.

Así que esta es mi oportunidad para devolver algo a la comunidad. Aquí está mi historia de guerra para vosotros. Espero que este libro me aporte algún feedback de aquellos de vosotros en la misma situación. ¡Por favor, iluminadme!

Pero ¿Qué es Scrum?

Oh, lo siento. ¿Eres totalmente Novato en Scrum o XP? En ese caso, quizás quieras echar un vistazo a los siguientes enlaces:

- <http://agilemanifesto.org/>
- <http://www.mountaingoatsoftware.com/scrum>
- <http://www.xprogramming.com/xpmag/whatisxp.htm>

Si eres demasiado impaciente como para hacerlo, siéntete libre seguir leyendo. La mayoría de la jerga la iremos explicando mientras avanzamos, así que posiblemente seguirás encontrando este libro interesante.

2

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Product Backlog

Como hacemos Pilas de Producto

La pila de producto es el corazón de Scrum. Es donde empieza todo. La Pila de Producto es, básicamente, una lista priorizada de requisitos, o historias, o funcionalidades, o lo que sea. Cosas que el cliente quiere, descritas usando la terminología del cliente. Llamamos a esto *historias*, o a veces simplemente *elementos de la Pila*.

Nuestras historias incluyen los siguientes campos:

- **ID** – un identificador único, simplemente un número auto-incremental. Esto nos permite no perder la pista a las historias cuando cambiamos su nombre.
- **Nombre** – una descripción corta de la historia. Por ejemplo, “Ver tu historial de transacciones”. Suficientemente claro como para que el Dueño de Producto comprenda aproximadamente de qué estamos hablando, y suficientemente clara como para distinguirla de las otras historias. Normalmente, 2 a 10 palabras.
- **Importancia** – el ratio de importancia que el Dueño de Producto da a esta historia. Por ejemplo, 10. O 150. Más alto = más importante. Suelo evitar el término “prioridad” porque típicamente “1” se considera la “máxima prioridad”, lo que es muy incómodo si posteriormente decides que algo es más importante. ¿Qué prioridad le daríamos a ese nuevo elemento? ¿Prioridad 0? ¿Prioridad -1?
- **Estimación inicial** – la valoración inicial del Equipo acerca de cuánto trabajo es necesario para implementar la historia, comparada con otras historias. La unidad son “puntos de historia” y usualmente corresponde a “días-persona ideales”. Pregunta al Equipo: “si tuvierais el número óptimo de personas para esta historia (ni muchos ni pocos, típicamente 2) y os encerraseis en una habitación con cantidad de comida, y trabajaseis sin distracciones, ¿en cuantos días saldríais con una implementación terminada, demostrable, testeada y liberable?”. Si la respuesta es “con 3 tíos encerrados en una habitación nos llevaría 4 días”, entonces la estimación inicial son 12 puntos. Lo importante no es que las estimaciones absolutas sean correctas (es decir, que una historia de 2 puntos deba durar 2 días), lo importante es que las estimaciones relativas sean correctas (es decir, que una historia de 2 puntos debería durar la mitad que una historia de 4 puntos).
- **Como probarlo** – una descripción a alto nivel de como se demostrará esta historia en la Demo al final del Sprint. Se trata, esencialmente, de una simple especificación de un test: “Haz esto, entonces haz lo otro, y entonces debería ocurrir aquello”. Si practicáis TDD (Test-Driven Development, desarrollo orientado a test) esta descripción puede usarse como pseudo-código para vuestro test de aceptación.

- **Notas** – cualquier otra información, clarificación, referencia a otras fuentes de información, etc. Normalmente muy breve.

Pila de Producto (ejemplo)					
ID	Nombre	Imp.	Est.	Como probarlo	Notas
1	Depósito	30	5	Entrar, abrir página de depósito, depositar 10€, ir a página de balance y comprobar que se ha incrementado en 10€	Necesita un diagrama UML. No preocuparse por encriptación aun
2	Ver tu historial de transacciones	10	8	Entrar, ver transacciones. Realizar un depósito de 10€. Ir a transacciones y comprobar que se ha actualizado con el nuevo depósito	Utilizar paginación para no hacer consultas muy grandes a la BB.DD. Diseño similar a la página usuario.

Hemos experimentado con muchos otros campos, pero al final estos seis campos son los únicos que realmente se usaban Sprint tras Sprint.

Mantenemos esta tabla en un documento Excel con “compartir” habilitado (es decir, muchos usuarios pueden editar simultáneamente la hoja). Oficialmente, el Dueño de Producto es el propietario del documento, pero no queremos dejar al resto de usuarios fuera. Muchas veces un desarrollador necesita abrir el documento para clarificar algo o cambiar una estimación.

Por la misma razón, no colocamos este documento en el repositorio de control de versiones; en vez de eso, lo almacenamos en una unidad de red compartida. Esta ha demostrado ser la manera más simple de permitir múltiples editores diferentes sin causar problemas de bloqueo o fusión de documentos.

Sin embargo, casi todos los demás artefactos se colocan en el repositorio de control de versiones.

Campos de historia adicionales

A veces usamos campos adicionales en la Pila de Producto, fundamentalmente como comodidad para el Dueño de Producto a la hora de decidir sus prioridades.

- **Categoría** – una categorización básica de la historia, por ejemplo “backoffice” o “optimización”. Así, el dueño de producto puede filtrar fácilmente “optimización” y cambiar todas las prioridades de este tipo a “baja”, etc.
- **Componentes** - usualmente implementado en la forma de “checkboxes” en el documento Excel, por ejemplo “base de datos, servidor, cliente”. Aquí, el Dueño de Producto puede identificar qué componentes técnicos estarán involucrados en la implementación de la historia. Esto es útil

cualquier momento tienes **varios equipos Scrum**, por ejemplo un **equipo de backoffice** y otro **equipo de cliente**, y quieres que sea fácil para cada equipo saber a qué historias deben dedicarse.

- **Solicitante** – el **Dueño de Producto** puede querer mantener un historial acerca de qué **cliente o persona interesada pidió originalmente la historia**, para poder así **ofrecerle información actualizada** sobre el **progreso** de la misma.
- **Bug tracking ID** – si tienes un sistema de bug *tracking* (seguimiento de errores) aparte, como hacemos nosotros con Jira, es útil mantener un historial de cualquier correspondencia directa entre una historia y uno o más errores reportados.

Como mantenemos la Pila de Producto a nivel de negocio

Si el **Dueño de Producto** tiene una formación técnica, puede que añada historias del tipo “añadir índices a la tabla de eventos”. ¿Por qué quiere algo así? El auténtico objetivo subyacente probablemente será algo como “aligerar el formulario de búsqueda de eventos en el backoffice”.

Puede ocurrir que los **índices no fueran el cuello de botella** que hiciera al formulario ir lento. Puede que **fueran algo completamente diferente**. El equipo está normalmente mejor capacitado para averiguar como resolver algo, así que el **Dueño de Producto** debería concentrarse en los objetivos de negocio.

Cuando veo **historias con orientación técnica** como esta, normalmente **hago al Dueño de Producto** una serie de preguntas tipo “pero ¿Por qué?” hasta que encuentro el objetivo subyacente. Entonces, reformulo la historia en términos del **objetivo subyacente** (“**aligerar el formulario de búsqueda de eventos del backoffice**”). La descripción técnica original acaba siendo una nota (“indexar la tabla de eventos podría resolver esto”).

3

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Sprint planning

Como nos preparamos para la planificación de Sprint

OK, el día de la planificación de Sprint se aproxima rápidamente. Una lección que hemos aprendido una y otra vez es:

Lección: Asegúrate de que la Pila de Producto está perfectamente lista antes de la reunión de planificación de Sprint.

¿Y esto qué significa? ¿Que todas las historias deban estar perfectamente bien definidas? ¿Qué las estimaciones sean correctas? ¿Qué todas las prioridades hayan sido fijadas? ¡No, no y no! Lo que significa realmente es:

- ¡La pila de producto debe existir! (¿Podéis imaginároslo?)
- Debería haber una Pila de Producto y un dueño de producto (por producto, claro).
- Todos los elementos importantes deberían tener ratios de importancia asignados, diferentes ratios de importancia. En realidad, da igual si los elementos menos importantes tienen todos el mismo valor, ya que probablemente no se discutirán durante la planificación de Sprint en cualquier caso. Cualquier historia sobre la que el Dueño de Producto piense que tiene una remota posibilidad de incluirse en el Sprint debería tener un nivel de importancia único definido. El ratio de importancia se emplea solo para ordenar los elementos por relevancia. Así que si el elemento A tiene una importancia de 20 y el elemento B una importancia de 100, simplemente significa que B es más importante que A. No significa que B sea cinco veces más importante que A. Si B tuviera una importancia de 21, jaun significaría lo mismo! Es útil dejar espacio entre la secuencia de números por si aparece un elemento C que es más importante que A pero menos importante que B. Por supuesto, le podríamos dar un ratio de importancia de 20,5 a C, pero queda mal, así que en vez de ello dejamos espacio entre números.
- El dueño de producto debe comprender cada historia (normalmente él es el autor, pero en algunos casos otras personas añaden solicitudes, que el Dueño de Producto puede priorizar). No necesita saber cómo exactamente debe implementarse, pero debería entender por qué la historia está ahí.

Nota: Otras personas aparte del Dueño de Producto pueden añadir sus historias a la Pila de Producto. Pero no pueden asignarles niveles de importancia, ese es

un cometido exclusivo del Dueño de Producto. Tampoco pueden establecer estimaciones, ese es un cometido exclusivo del equipo.

Otras aproximaciones que hemos intentado o evaluado:

- Usar Jira (nuestro sistema de seguimiento de errores) para almacenar la Pila de Producto. La mayoría de nuestros Dueños de Producto encuentran que deben hacer demasiados “clicks” en ella. Excel está bien, y es fácil de manejar directamente. Puedes añadir códigos de color fácilmente, reordenar los elementos, añadir nuevas columnas, notas, importar y exportar datos, etc. Todo ello “ad-hoc”.
- Utilizar una herramienta de soporte a procesos ágiles, como VersionOne, ScrumWorks, XPlanner, etc. Todavía no hemos conseguido probar ninguna de ellas, pero probablemente lo haremos.

4

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Como hacemos la planificación de Sprint

La planificación de Sprint es una reunión crítica, probablemente la más importante de Scrum (en mi subjetiva opinión, por supuesto). Una planificación de Sprint mal ejecutada puede arruinar por completo todo el Sprint.

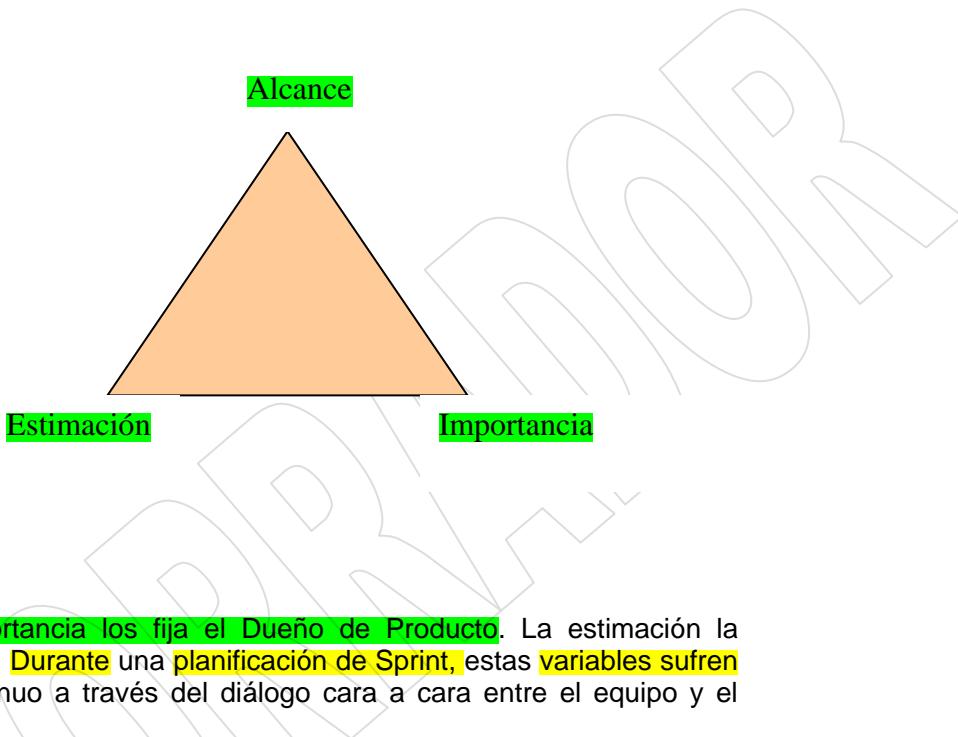
El propósito de la planificación de Sprint es proporcionar al equipo suficiente información como para que puedan trabajar en paz y sin interrupciones durante unas pocas semanas, y para ofrecer al Dueño de Producto suficiente confianza como para permitírselo.

Vale, ha quedado un poco difuso. Una planificación de Sprint produce, concretamente:

- Una **sprint goal**.
- Una **meta de Sprint**.
- Una **lista de miembros** (y su nivel de dedicación, si no es del 100%)
- Una **Pila de Sprint** (lista de historias incluidas en el Sprint)
- Una **fecha concreta** para la **Demo del Sprint**.
- Un **lugar y momento** definidos para el **Scrum Diario**.

Por qué debe asistir el Dueño de Producto

A veces los Dueños de Producto se resisten a pasar horas con el equipo preparando la planificación de Sprint. "Mirad, tíos, ya os he listado lo que quiero. No tengo tiempo para estar en vuestra reunión de planificación". Es un problema bastante serio. La razón por la que el equipo y el Dueño de Producto deben asistir a la planificación de Sprint es que cada historia contiene tres variables que son muy dependientes unas de otras.



El alcance y la importancia los fija el Dueño de Producto. La estimación la proporciona el equipo. Durante una planificación de Sprint, estas variables sufren un ajuste fino y continuo a través del diálogo cara a cara entre el equipo y el Dueño de Producto.

Normalmente, el Dueño de Producto comienza la reunión resumiendo cuál es su meta para el Sprint y las historias más importantes. A continuación, el equipo las repasa y les asigna una estimación, comenzando con la más importante. Conforme van haciéndolo, aparecerán dudas importantes respecto al alcance: "esta historia sobre 'borrar usuario', ¿incluye repasar todas las transacciones pendientes del usuario y cancelarlas?". En algunos casos, las respuestas sorprenderán al equipo y les obligarán a cambiar sus estimaciones.

En algunos casos, la estimación para una historia no será la que el Dueño de Producto esperaba. Esto puede forzarle a cambiar la importancia de la historia o su alcance, lo que obligará al equipo a re-estimarla, etc., etc.

Este tipo de colaboración directa es fundamental en Scrum. De hecho, en todo el Desarrollo Ágil de software. ¿Qué ocurre si el Dueño de Producto insiste en que no tiene tiempo de unirse a las reuniones de planificación de Sprint? Usualmente intento alguna de las siguientes estrategias, en el siguiente orden:

- Intentar que el Dueño de Producto comprenda por qué su participación es crucial y esperar que cambie de parecer.
- Intentar que alguien del equipo se presente voluntario como delegado del Dueño de Producto durante la reunión. A continuación, decirle al Dueño de Producto "ya que no puedes venir a nuestra reunión vamos a dejar que Jeff te represente. Tendrá poder de decisión pleno para cambiar las prioridades y el alcance de las historias por ti durante la reunión. Te sugiero que te sincronices con él lo máximo posible antes de la reunión. Si no te gusta Jeff como delegado, por favor, sugiere a otro, siempre que ese otro pueda asistir a toda la reunión".

- Tratar de convencer a la gerencia de que asigne un nuevo Dueño de Producto.
- Posponer el lanzamiento del Sprint hasta que el Dueño de Producto encuentre el tiempo para asistir a la reunión. Mientras tanto, no comprometerse a ninguna entrega y permitir que el equipo pase el día haciendo cualquier cosa que les parezca importante hacer.

Por qué la calidad no es negociable

En el triángulo mencionado anteriormente he dejado intencionadamente fuera una cuarta variable: la calidad. Tiendo a distinguir entre *calidad interna* y *calidad externa*.

- *Calidad externa*: es lo que perciben los usuarios del sistema. Un interfaz de usuario lento y poco intuitivo es un ejemplo de baja calidad externa.
- *Calidad interna*: se refiere a aquellos aspectos que normalmente no son visibles al usuario, pero que tienen un profundo efecto en la mantenibilidad del sistema. Cosas como consistencia del diseño del sistema, cobertura de pruebas, legibilidad del código, refactorización, etc.

Generalizando, un sistema con alta calidad interna puede, aun así, tener una baja calidad externa. Pero un sistema con baja calidad interna rara vez tendrá buena calidad externa. Es difícil construir algo sobre unos cimientos podridos.

Yo trato la calidad externa como parte del alcance. En algunos casos puede tener sentido, desde el punto de vista de negocio, liberar una versión del producto que tenga un internaza de usuario torpe y lento, y más tarde liberar una versión mejorada. Dejo esa decisión al Dueño de Producto, ya que él es el responsable de definir el alcance.

Sin embargo, la calidad interna es algo que no puede ser discutido. Es responsabilidad del equipo mantener la calidad del sistema bajo toda circunstancia y simplemente no es negociable. Nunca. (Vale, OK, casi nunca).

¿Y como distinguimos la diferencia entre aspectos de calidad externa y aspectos de calidad interna? Supongamos que el Dueño de Producto dice "OK tíos, respeto vuestras estimaciones de esfuerzo de 6 puntos de historia, pero estoy seguro de que podríamos usar algún apaño rápido para hacerlo en la mitad de tiempo si le das una pensada".

¡Ajá! Está intentando usar la calidad interna como una variable. ¿Cómo lo se? Porque quiere que reduzcamos la estimación de la historia sin "pagar el precio" de reducir el alcance. La palabra "apaño" debería disparar una alarma en vuestras mentes...

¿Y por qué no permitimos esto? Mi experiencia es que sacrificar la calidad interna es, prácticamente siempre, una idea terrible. El tiempo que se ahorra es mucho menor que el coste, tanto a corto como a largo plazo. Una vez que permites que una base de código comience a deteriorarse es muy duro volver a conseguir su calidad original más adelante.

En lugar de eso intento reconducir la discusión hacia el alcance. "Ya que es importante para ti que entreguemos esta historia pronto, ¿podemos reducir su alcance de forma que sea más rápida de implementar? Quizás podríamos

simplificar el manejo de errores y convertir "Manejo Avanzado de Errores" en una historia separada que reservemos para el futuro. O podríamos reducir la prioridad de otras historias de forma que podamos centrarnos en esta".

Una vez que el Dueño de Producto aprende que la calidad interna no es negociable, normalmente se hace muy bueno en manipular las otras variables.

Reuniones de planificación de Sprint que duran, y duran...

Lo más difícil de las Planificaciones de Sprint es que:

- 1) La gente no piensa que vayan a durar tanto...
- 2) ... ¡Pero lo hacen!

Todo en Scrum tiene una duración determinada (*time-boxed*). Me encanta esa única, simple y consistente regla. Intentamos cumplirla al cien por cien. Así que ¿qué hacemos cuando la reunión de planificación de Sprint está llegando al final y no hay señales de una meta de Sprint o Pila de Sprint? ¿Limitamos la reunión al tiempo establecido? ¿O la extendemos una hora? ¿O terminamos por hoy y continuamos al día siguiente?

Esto nos pasa una y otra vez, especialmente con los equipos nuevos. Así que, ¿qué hacer? No lo sé. Pero, ¿qué hacemos nosotros? Oh, um, vale, usualmente suelo cerrar la reunión brutalmente a la hora en punto. Acabarla. Dejar que el Sprint sufra. Más específicamente, le digo al equipo y al Dueño de Producto "bueno, esta reunión acaba en 10 minutos. No tenemos lo que se dice un Plan de Sprint, realmente. ¿Trabajamos con lo que tenemos o planifico otra reunión de cuatro horas mañana a las ocho de la mañana?". Podéis imaginar cual es la respuesta... :o)

He intentado dejar que las reuniones se alarguen. Eso normalmente no vale para nada, porque la gente está cansada. Si no han producido un Plan de Sprint decente en 2 – 8 horas (o lo que sea que duren vuestras reuniones de planificación), probablemente no lo lograrán en otra hora más. La siguiente opción no está tan mal, planificar otra reunión al día siguiente. Excepto que la gente normalmente está impaciente y quieren comenzar con el Sprint, y no emplear otro montón de horas en planificación.

Así que acabo con la reunión. Y sí, el Sprint sufre. La ventaja, en cualquier caso, es que el equipo ha aprendido una lección muy valiosa, y en la próxima reunión de planificación de Sprint serán mucho más eficientes. Adicionalmente, ofrecerán menos resistencia cuando propongas una duración para las reuniones que anteriormente hubieran considerado excesiva.

Aprende a mantener tus duraciones determinadas, aprende a establecer duraciones realistas. Esto aplica tanto a las reuniones como a los Sprints.

Agenda de la reunión de planificación de Sprint

Tener algún tipo de agenda u orden del día de la reunión de planificación de Sprint reducirá el riesgo de sobrepasar la duración determinada. Este es un ejemplo típico de nuestras agendas:

Reunión de planificación de Sprint: 13:00 – 17:00 (10 minutos de descanso cada hora)

- **13:00 – 13:30.** El Dueño de Producto comenta la meta del Sprint y resume la Pila de Producto. Se establece un lugar, fecha y hora para la Demo.
- **13:30 – 15:00.** El equipo da estimaciones de tiempo, y divide los elementos tanto como sea necesario. El Dueño de Producto actualiza los ratios de importancia. Se clarifican los elementos. Para todos los elementos de alta importancia se establece el "Como probarlo".
- **15:00 – 16:00.** El equipo selecciona las historias que se incluirán en el Sprint. Se realizan cálculos de velocidad para chequear si es factible.
- **16:00 – 17:00.** Se selecciona un lugar y hora para el Scrum Diario (si es que cambio respecto al último Sprint). Se continúa dividiendo las historias en tareas.

La agenda no es en absoluto inamovible. El Scrum Master puede ampliar o acortar los períodos según sea necesario conforme progrese la reunión.

Definiendo la duración del Sprint

Uno de los resultados de la planificación de Sprint es una fecha de Demo de Sprint definida. Eso significa que debes determinar una duración del Sprint.

¿Y cuánto debería durar un Sprint?

Bueno, los Sprints cortos están bien. Permiten a la compañía ser "ágil", es decir, cambiar de dirección frecuentemente. Sprints cortos = ciclo de feedback corto = más entregas y más frecuentes = más feedback del cliente = menos tiempo desarrollando en dirección incorrecta = aprender y mejorar más rápido, etc.

Pero los Sprints largos tampoco están mal. El equipo tiene más tiempo para conseguir impulso, tienen más espacio para recuperarse de los problemas que surjan y aun así cumplir la meta del Sprint, tiene menos carga de gestión en términos de reuniones de planificación de Sprints, Demos, etc.

Generalmente, los Dueños de Producto prefieren los Sprints cortos y a los desarrolladores les gustan los Sprints largos. Así que la duración del Sprint es un valor de compromiso. Hemos experimentado con varias duraciones y al final encontramos nuestra duración favorita: 3 semanas. La mayoría de nuestros equipos (pero no todos) hacen Sprints de 3 semanas. Suficientemente cortos para proporcionarnos agilidad corporativa, suficientemente largos para lograr flujo y recuperarse de los problemas que aparezcan durante el Sprint.

Una conclusión a la que hemos llegado es: experimentad con las duraciones al principio. No perdáis mucho tiempo analizando, simplemente seleccionad una duración decente y dale un Sprint o dos, y entonces cambiad la duración. En cualquier caso, una vez que hayáis establecido una duración, mantenedla durante un buen periodo de tiempo. Después de unos pocos meses de experimentación, nosotros encontramos que 3 semanas estaba bien. Así que hacemos Sprints de 3 semanas, punto. A veces puede parecer un poco corto, a veces un poco largo. Pero manteniendo la misma duración conseguimos un

latido corporativo al que todo el mundo se acostumbra confortablemente. No hay discusiones sobre las fechas de entrega ya que todo el mundo sabe que cada 3 semanas hay una entrega. Punto.

sprint goal

Definiendo la meta del Sprint

Pasa todo el tiempo. En algún momento durante la reunión de planificación de Sprint, pregunto “¿Y cual es la meta del Sprint”, y todo el mundo me mira anonadado mientras el Dueño de Producto arruga la frente y se rasca la barbilla.

Por alguna razón, es difícil conseguir una meta de Sprint. Pero aun así he descubierto que realmente merece la pena exprimir una. Mejor una medio-metá roñosa que no tener ninguna meta. La meta podría ser “ganar más dinero” o “impresionar al Director General” o “hacer el sistema lo suficientemente bueno como para entregarlo a un grupo de usuarios beta reales” o “añadir soporte de backoffice básico” o lo que sea. Lo importante es que esté descrito en términos de negocio. Eso significa en términos en los que la gente de fuera del equipo lo pueda entender.

La meta de Sprint debería responder a la pregunta fundamental “¿Por qué hacemos este Sprint en vez de irnos todos de vacaciones?”. De hecho, una forma de obtener la meta del Dueño de Producto es precisamente hacerle esa misma pregunta.

La meta debería ser algo que no se haya logrado aun. “Impresionar al Director General” puede ser una meta aceptable, pero sólo si no está impresionado aun por el sistema tal y como se encuentra ahora mismo. Si fuera así, todo el mundo podría irse a casa y aun así lograr la meta del Sprint.

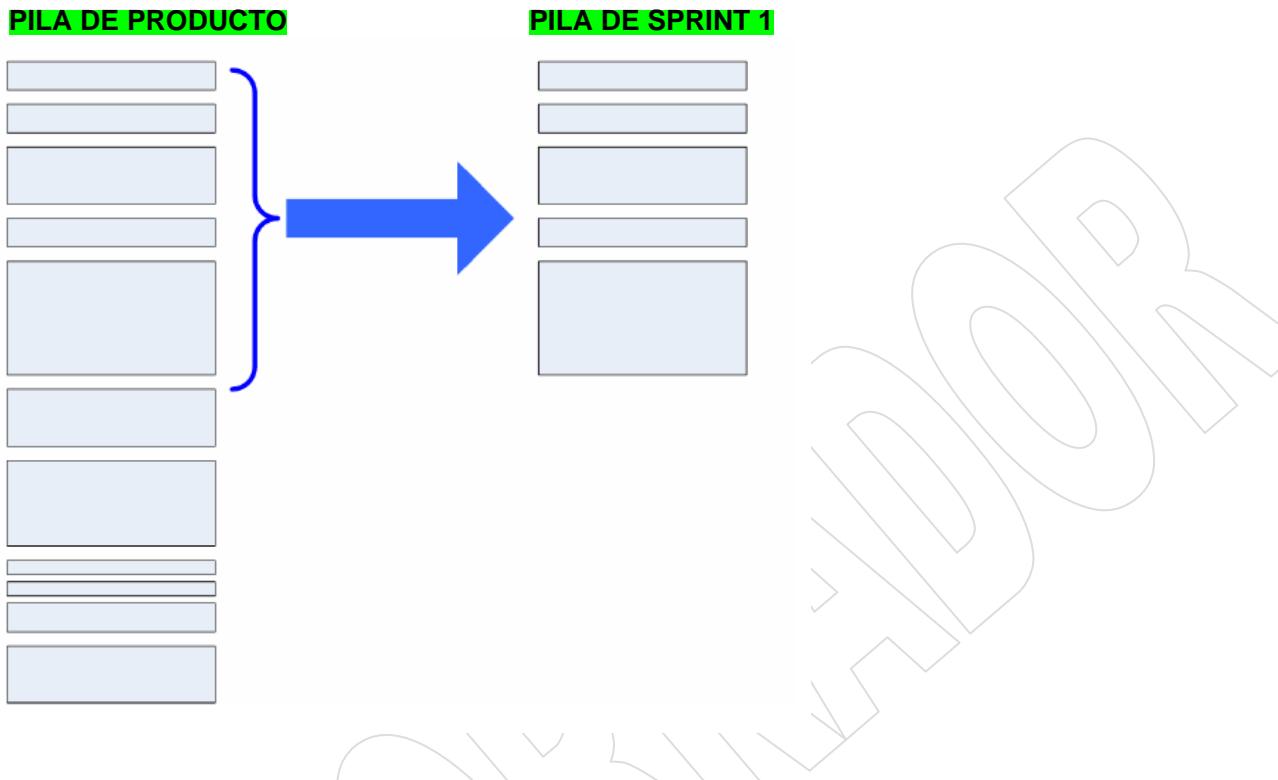
La meta del Sprint puede parecer bastante tonta y artificial durante la planificación de Sprint, pero muchas veces resulta útil a mediados de Sprint, cuando la gente comienza a sentirse confusa acerca de lo que deberían estar haciendo. Si tienes bastante equipos Scrum (como nosotros) trabajando en diferentes productos, es muy útil poder listar todas las metas de Sprint en una página Wiki (o donde sea) y colocarla en un sitio prominente donde todo el mundo en la empresa (no solo la alta dirección) pueda saber qué está haciendo la compañía - y por qué.

Decidiendo qué historias incluir en el Sprint

Una de las principales actividades durante la planificación de Sprint es decidir qué historias se incluyen en el Sprint. Más específicamente, qué historias de la Pila de Producto copiar en la Pila de Sprint.

product backlog

sprint backlog



Observa la imagen anterior. Cada rectángulo representa una historia, ordenadas por importancia. La historia más importante está al principio de la lista. El tamaño de cada rectángulo representa el tamaño de la historia (es decir, el tiempo estimado en puntos de historia). La altura del corchete azul representa la velocidad estimada del equipo, es decir, cuántos puntos de historia cree el equipo que puede completar durante el próximo Sprint.

La Pila de Sprint de la derecha es una instantánea de las historias la Pila de Producto. Representa las historias a las que el equipo se compromete durante este Sprint. *El equipo decide cuántas historias incluirá en el Sprint. No el Dueño de Producto ni nadie más.*

Esto plantea dos cuestiones:

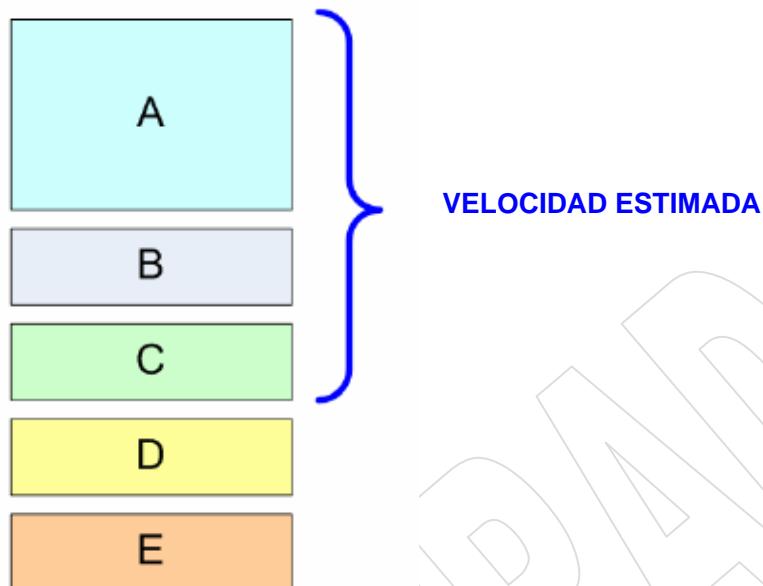
1. ¿Cómo decide el equipo qué historias incluir en el Sprint?
2. ¿Cómo puede el Dueño de Producto alterar la decisión del equipo?

Empezaré con la segunda cuestión.

¿Cómo puede el Dueño de Producto alterar las historias que se incluyen en el Sprint?

Supongamos que tenemos la siguiente situación durante una reunión de planificación de Sprint.

PILA DE PRODUCTO



Al Dueño de Producto no le gusta que la historia D no se vaya a incluir en el Sprint. ¿Cuáles son sus opciones durante la reunión de planificación de Sprint?

Una es **re-priorizar**. Si le da al **elemento D la mayor importancia**, el equipo se verá obligado a **añadirlo al Sprint** en primer lugar (**descartando** en ese caso la historia C).

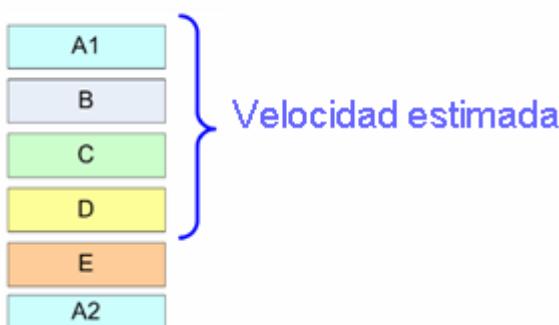
Opción 1:



La **segunda opción** es cambiar el **alcance** – **reducir el alcance de la historia A** hasta que el equipo crea que **la historia D podría caber en el Sprint**.

Opción 2:

La tercera sería dividir una historia. El Dueño de Producto podría decidir que hay algunos aspectos de la historia A que no son tan importantes, así que dividiría la historia A en dos historia A1 y A2 con diferentes niveles de importancia.

Opción 3:

Como ves, aunque el Dueño de Producto no puede controlar normalmente la velocidad estimada, hay varias formas en las que puede influenciar qué historias entran en cada Sprint.

¿Cómo decide el equipo qué historias incluir en el Sprint?

Utilizamos dos técnicas para esto:

1. A ojo de buen cubero
2. Cálculos de velocidad

Estimando a ojo de buen cubero

- **ScrumMaster:** “A ver chicos, ¿podemos terminar la historia A en este Sprint?” (señala el elemento más importante de la Pila de Producto).
- **Lisa:** “Bah. Por supuesto que podemos. Tenemos tres semanas, y es una funcionalidad bastante trivial.”

- **ScrumMaster;** "OK, ¿y si le añadimos la historia B?" (señala el segundo elemento más importante)
- **Tom&Lisa al unísono:** "Sigue siendo muy fácil."
- **ScrumMaster:** "OK, ¿qué tal las historias A, B y C entonces?"
- **Sam (al Dueño de Producto):** "¿La historia C incluye manejo avanzado de errores?"
- **Dueño de Producto:** "No, no es necesario por ahora, podéis implementarlo con un manejo básico de errores."
- **Sam:** "Entonces C podría entrar también."
- **ScrumMaster:** "OK, ¿y si añadimos la historia D?"
- **Lisa:** "Hmmm..."
- **Tom:** "Creo que podríamos hacerlo."
- **ScrumMaster:** "¿Seguro al 90%? ¿Al 50%?"
- **Lisa y Tom:** "Más bien al 90%."
- **ScrumMaster:** "OK, D entra entonces. ¿Y si añadimos la historia E?"
- **Sam:** "Quizás."
- **ScrumMaster:** "¿90%? ¿50%?"
- **Sam:** "Yo diría que más cerca del 50%".
- **Lisa:** "Tengo mis dudas."
- **ScrumMaster:** "OK, entonces lo dejamos fuera. Nos comprometeremos a A, B, C y D. Por supuesto, terminaremos E si podemos, pero nadie debería contar con ello, así que lo dejamos fuera del plan de Sprint. ¿Qué tal así?".
- **Todo el mundo:** "OK!"

El ojo de buen cubero funciona bastante bien para equipos pequeños y Sprints cortos.

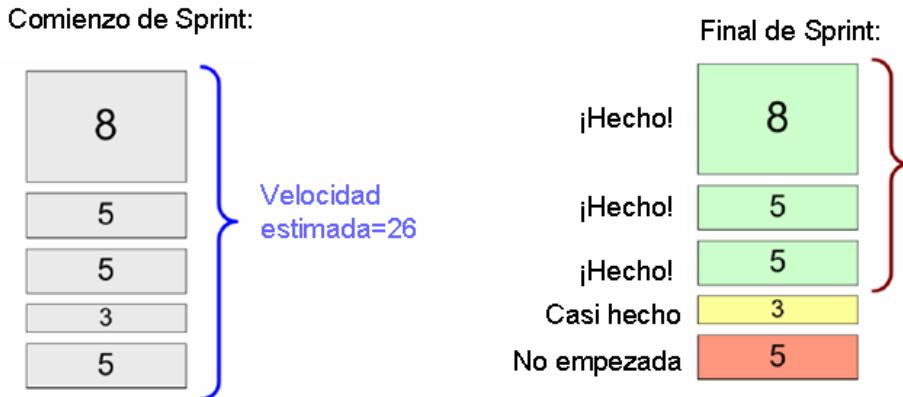
Estimando usando cálculos de velocidad

Esta técnica consta de dos pasos

1. **Decidir la velocidad estimada**
2. **Calcular cuántas historias se pueden añadir sin sobrepasar la velocidad estimada**

La velocidad es una medida de "cantidad de trabajo realizado", donde cada elemento se evalúa en función de su estimación inicial.

El siguiente gráfico muestra un ejemplo de velocidad estimada al principio de un Sprint y la velocidad real al final de dicho Sprint. Cada rectángulo es una historia, y el número interior es la estimación inicial de dicha historia.



Nota que la velocidad real esta basada en las estimaciones iniciales de cada historia. Cualquier actualización a la estimación de la historia realizada durante el Sprint es ignorada.

Ya puedo escuchar las objeciones: “¿Cómo va a ser esto útil? ¡Una velocidad alta o baja dependerá de un montón de factores! Programadores con poco sentido común, estimaciones iniciales incorrectas, cambios en el alcance, distracciones imprevistas durante el Sprint, etc.

Estoy de acuerdo en que se trata de un número bastante aproximado. Pero aun así es bastante útil, especialmente si lo comparamos con “hada en absoluto”. Te proporciona algunos verdades crudas: “sean cuales sean las razones, aquí está la diferencia aproximada entre cuánto pensábamos que podríamos hacer y cuánto hicimos en realidad”.

¿Qué pasa con las historias que casi se consiguieron durante el Sprint? ¿Por qué no conseguimos algunos puntos por ellas en la velocidad real? Bueno, esto es así para reforzar el hecho de que Scrum (y de hecho todo el Desarrollo Ágil de Software y Lean Manufacturing en general) gira en torno al concepto de conseguir que las cosas se hagan completamente, hasta un estado de “potencialmente entregable”. El valor de las cosas medio hechas es cero (podría de hecho ser negativo). Coged el libro de Donald Reinertsen “Managing the Design Factory” o uno de los libros de Poppendieck para aprender más sobre este concepto.

Así que, ¿mediante qué tipo de magia arcana estimamos la velocidad?

Una manera muy fácil de estimar la velocidad es revisar la historia del equipo. ¿Cuál fue su velocidad durante los últimos Sprints? Y entonces asumir que la velocidad será más o menos la misma en el próximo Sprint.

Esta técnica se conoce como *el tiempo que hizo ayer*. Solo es factible para equipos que ya han hecho algunos Sprints (de forma que haya estadísticas disponibles) y que harán el próximo Sprint más o menos de la misma manera, con el mismo tamaño de equipo, las mismas condiciones de trabajo, etc. Este, claro está, no siempre es el caso.

Una forma más sofisticada de hacerlo es realizar un simple cálculo de recursos. Digamos que estamos planificando un Sprint de 3 semanas (15 días laborables) con un equipo de 4 personas. Lisa estará de vacaciones 2 días. Dave sólo estará disponible al 50% y estará un día de vacaciones. Poniéndolo todo junto...

DÍAS DISPONIBLES	
TOM	15
LISA	13
SAM	15
DAVE	7
<u>50 DÍAS-HOMBRE DISPONIBLES</u>	

...Tenemos 50 días-hombre disponibles en este Sprint.

¿Es esta nuestra velocidad estimada? ¡No! Porque nuestra unidad de estimación son puntos de historia lo que, en nuestro caso, corresponde más o menos a "días-hombre ideales". Un día-hombre ideal es un día perfectamente efectivo, sin distracciones, lo cuál es raro. Lo que es más, debemos tener en cuenta cosas como trabajo no planificado que se añade al Sprint, gente que se pone enferma, etc.

Así que nuestra velocidad estimada será sin duda menor de 50. ¿Pero cuánto menor? Para esto usamos el "factor de dedicación".

VELOCIDAD ESTIMADA DE ESTE SPRINT

$$(DÍAS-HOMBRE DISPONIBLES) \times (\text{FACTOR DE DEDICACIÓN}) = \text{VELOCIDAD ESTIMADA}$$

El factor de dedicación es una estimación de cómo de centrado va a estar el equipo. Un factor de dedicación bajo puede significar que el equipo espera encontrar muchas distracciones e impedimentos o que considera que sus propias estimaciones son optimistas.

La mejor manera de determinar un factor de dedicación razonable es estudiar el último Sprint (o incluso mejor, la media de los últimos Sprints).

FACTOR DE DEDICACIÓN DEL ÚLTIMO SPRINT

$$\text{FACTOR DE DEDICACIÓN} = \frac{(\text{VELOCIDAD REAL})}{(\text{DIAS-HOMBRE DISPONIBLES})}$$

La velocidad real es la suma de las estimaciones iniciales que se completaron en el último Sprint.

Digamos que en el último Sprint se completaron 18 puntos de historia utilizando un equipo de 3 personas formado por Tom, Lisa y Sam trabajando las tres

semanas hasta un total de 45 días-hombre. Y ahora estamos intentando calcular la velocidad del próximo Sprint. Para complicar las cosas, un nuevo tipo, **Dave**, se une al equipo para este Sprint. Teniendo en cuenta las vacaciones y demás asuntos tenemos 50 días-hombre ideales este Sprint.

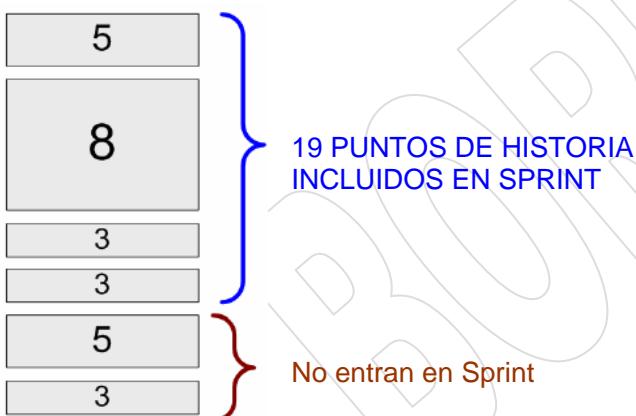
FACTOR DE DEDICACIÓN DEL ÚLTIMO SPRINT

$$\frac{(18 \text{ PUNTOS HISTORIA})}{(45 \text{ DÍAS-HOMBRE})} = 40\%$$

$$50 \text{ DÍAS-HOMBRE} \times 40\% = 20 \text{ PUNTOS}$$

Así que nuestra velocidad estimada para el próximo Sprint es de 20 puntos de historia. Eso significa que el equipo debe añadir historias al Sprint hasta que sume aproximadamente 20.

COMIENZO DEL SPRINT



En este caso, el equipo puede escoger las 4 historias más importantes hasta un total de 19 puntos de historia, o las 5 historias más importantes hasta 24 puntos de historia. Digamos que escogen 4 historias, ya que se aproximan más a los 20 puntos de historia. Siempre que haya dudas, escoge añadir menos historias.

Dado que estas 4 historias suman 19 puntos, la velocidad estimada final para este Sprint es de 19.

“El tiempo que hizo ayer” es una técnica sencilla, pero úsala con cierta dosis de sentido común. Si el último Sprint fue especialmente malo porque la mayoría del equipo estuvo enfermo una semana, entonces podría ser adecuado asumir que no volverás a tener tan mala suerte y podrías estimar un factor de dedicación mayor el próximo Sprint. Si el equipo ha instalado recientemente un sistema super-rápido de compilación continua probablemente también podrás incrementar el factor de dedicación gracias a ello. Si una nueva persona se une a este Sprint deberías reducir su factor de dedicación para tener en cuenta su formación. Etc.

Siempre que sea posible, ten en cuenta varios Sprints y saca medias para conseguir estimaciones más acertadas.

proponer un

¿Qué ocurre si el equipo es completamente Nuevo y no tienes ninguna estadística? Mira al factor de dedicación de otros equipos en circunstancias similares. ¿Qué pasa si no tienes otros equipos en los que fijarte? Adivina un factor de dedicación. Las buenas noticias son que sólo deberás adivinar para el primer Sprint. Después, tendrás estadísticas que podrás ir midiendo continuamente para aproximar mejor el factor de dedicación y la velocidad estimada.

El factor de dedicación "por defecto" que usamos para equipos nuevos es habitualmente el 70%, dado que es ahí donde terminan la mayoría de nuestros otros equipos con el tiempo.

¿Qué técnicas de estimación usamos?

velocity calculation based on yesterday's weather,

He mencionado varias técnicas anteriormente – ojo de buen cubero, cálculo de velocidad basado en el tiempo que hizo ayer y cálculo de velocidad basado en días-hombre disponibles y factor de dedicación.

Así que, ¿qué técnica usamos?

Usualmente combinamos todas estas técnicas hasta cierto punto. Realmente no conlleva mucho esfuerzo.

Miramos el factor de dedicación y la velocidad real del último Sprint. Miramos al total de recursos disponibles para este Sprint y estimamos el factor de dedicación. Discutimos cualquier diferencia entre estos dos factores de dedicación y hacemos los ajustes que sean necesarios.

Una vez que tenemos la lista preliminar de historias a incluir en el Sprint hacemos un chequeo a "ojos de buen cubero". Le pido al equipo que ignore los números por un momento y que plensen sobre si les parece un bocado realista como para zampárselo en un Sprint.

Si parece demasiado, quitamos una historia o dos. Y viceversa. Al final del día, el objetivo es simplemente decidir qué historias se incluyen en el Sprint. Factores de dedicación, disponibilidad de recursos y velocidad estimada son sólo medios para conseguir dicho objetivo.

Por qué usamos tarjetas

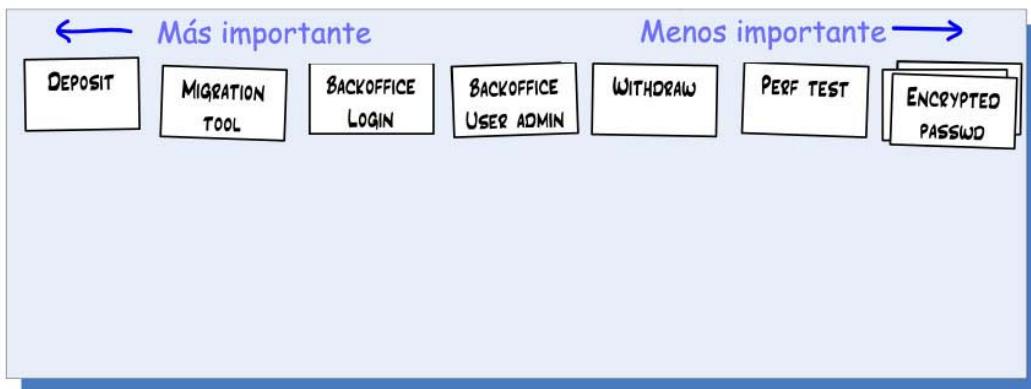
La mayor parte de la reunión de planificación de Sprint se dedica a las historias de la Pila de Producto. Estimándolas, repriorizándolas, clarificándolas, dividiéndolas, etc.

¿Cómo hacemos esto en la práctica?

Bueno, por defecto, los equipos solían conectar el proyector, mostrar la Pila basada en Excel y alguien (típicamente el Dueño de Producto o el ScrumMaster) toma el teclado, murmura algo sobre cada historia e invita a la discusión. Conforme el equipo y el Dueño de Producto discuten las prioridades y los detalles, la persona en el teclado actualiza cada historia directamente en Excel.

¿**Suena bien?** Bueno, pues no. De hecho apesta. Y **lo que es peor**, el equipo *no se da cuenta de que apesta* hasta que llega el final de la reunión y se dan cuenta de que **no han conseguido revisar toda la lista de historias**.

Una solución que funciona mucho mejor es crear tarjetas y ponerlas en la **pared** (o en una mesa grande).



Este es un interfaz muy superior, comparado con un ordenador y un proyector, debido a que:

- La gente se pone de pie y camina alrededor → se mantienen despiertos y alerta más tiempo
- Todo el mundo se siente personalmente más involucrado (y no solamente el tipo del teclado)
- Se pueden editar múltiples historias simultáneamente
- Repriorizar es trivial – simplemente se trata de mover las tarjetas
- Tras la reunión, las tarjetas pueden trasladarse directamente a la sala de equipo y usarse como un tablón de tareas en la pared (ver “cómo hacemos la Pila de Sprint”).

Puedes escribirlas a mano o (como hacemos nosotros) utilizar un simple programa para generar las tarjetas directamente desde la Pila de Producto (PD – el programa está disponible en mi blog, en <http://blog.crisp.se/henrikniberg>).

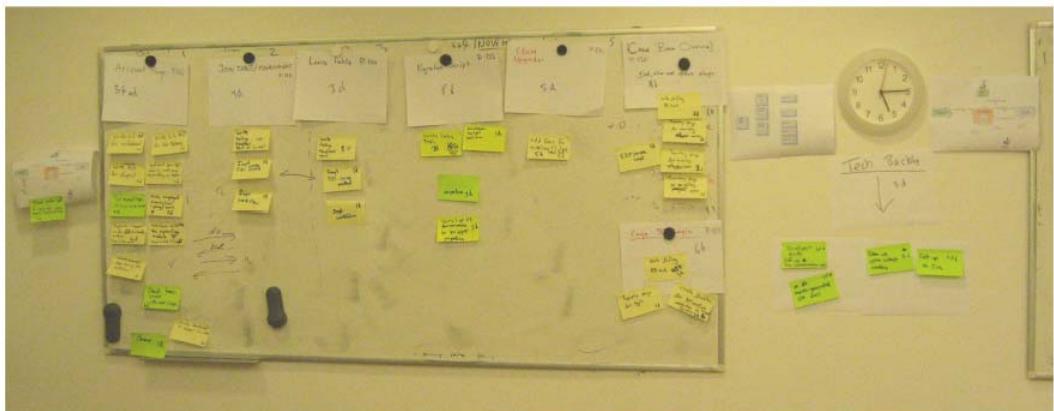
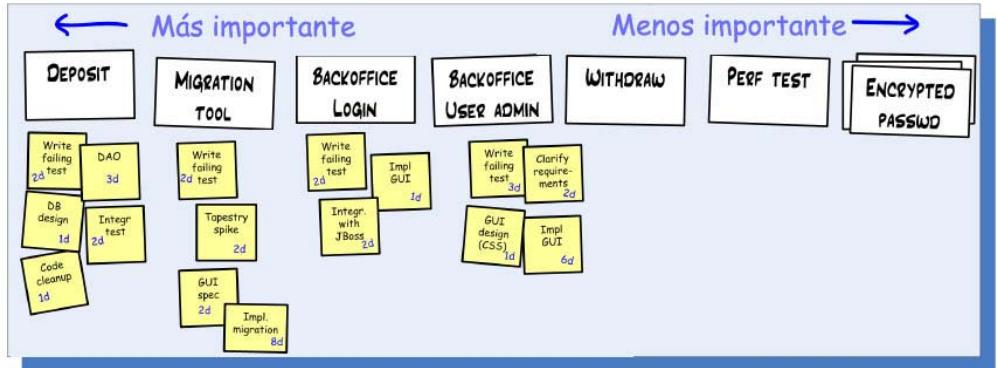
<p>Elemento de Pila</p> <h1>Depósito</h1> <p>Notas</p> <div style="border: 1px solid black; padding: 5px;"> Necesita un diagrama secuencial UML. No hay que preocuparse aún por el cifrado. </div> <p>Como probarlo</p> <div style="border: 1px solid black; padding: 5px;"> Entrar, abrir la página de depósito, depositar 10€, ir a la página de balance y comprobar que se ha incrementado en 10€ </div>	<p>Importancia</p> <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> 30 </div> <p>Estimación</p> <div style="border: 1px solid black; padding: 5px; height: 40px;"></div>
--	--

Importante: Tras la reunión de planificación de Sprint, nuestro ScrumMaster actualiza manualmente la Pila de Producto en Excel respecto a cualquier cambio que se haya realizado sobre las tarjetas de historia físicas. Sí, es una pequeña molestia administrativa, pero lo encontramos perfectamente asumible considerando cuánto más eficiente es la reunión de planificación de Sprint utilizando tarjetas físicas.

Una nota sobre el campo “importancia”: se trata de la importancia tal y como figura en la Pila de Producto en Excel en el momento de la impresión. Tenerla en las tarjetas hace que sea más sencillo ordenarlas físicamente por importancia (normalmente, ponemos las más importantes a la izquierda y las menos importantes a la derecha). En cualquier caso, una vez que las tarjetas están en la pared puedes ignorar el ratio de importancia y en lugar de ello usar el orden físico para indicar la importancia relativa. Si el dueño de producto cambia dos historias de sitio no pierdas el tiempo actualizando los ratios de importancia en la tarjeta. Simplemente asegúrate de actualizarlas en la hoja Excel después de la reunión.

Las estimaciones de tiempo son normalmente más fáciles de hacer (y más exactas) si una historia se subdivide en tareas. De hecho, utilizamos la palabra “actividades” porque “tareas” significa algo completamente diferente en Sueco :o)

Esto es algo que también se hace de forma sencilla y agradable con las tarjetas de historia. Puedes hacer que el equipo se divida en parejas y que cada una subdivida una historia en paralelo. Físicamente, hacemos esto añadiendo pequeñas notas post-it bajo cada historia, de forma que cada post-it representa una tarea dentro de dicha historia.



No actualizamos la Pila de Producto en Excel respecto a nuestra división en tareas por dos razones:

- La división en tareas suele ser bastante volátil, es decir, se cambia y se refina con frecuencia durante el Sprint, así que es bastante molesto mantener la Pila de Producto en Excel sincronizada.
- De todas formas, el Dueño de Producto no necesita estar involucrado a ese nivel de detalle.

Al igual que con las tarjetas de historia, los post-it de tareas pueden reutilizarse directamente en la Pila de Sprint (ver “cómo hacemos Pilas de Sprint”).

Definición de “terminado”

Es importante que el dueño de producto y el equipo estén de acuerdo en una definición clara de “terminado”. ¿Está terminada una historia cuando todo el código ha sido chequeado? ¿O está terminada cuando se ha instalado en un entorno de pre-producción y ha sido verificada por un equipo de pruebas de integración? Siempre que es posible, utilizamos la definición “lista para pasar a producción”, aunque a veces debemos conformarnos con “instalado en pre-producción y lista para las pruebas de aceptación”.

Al principio solíamos tener listas de comprobación detalladas para esto. Ahora usualmente decimos “una historia está terminada cuando el encargado de pruebas del equipo Scrum lo dice”. Así que es labor del encargado de pruebas asegurarse de que la intención del Dueño de Producto es correctamente

entendida por el equipo, y que el elemento esté lo suficientemente "terminado" como para cumplir con la definición de terminado.

Nosotros hemos llegado a la conclusión de que todas las historias no se pueden tratar igual. Una historia llamada "formulario de consulta de usuarios" se tratará de una forma muy diferente a otra llamada "manual de operaciones". En el último caso, la definición de terminado puede significar simplemente "aceptada por el equipo de operaciones". Es por eso que el sentido común es, a menudo, mejor que las listas de comprobación formales.

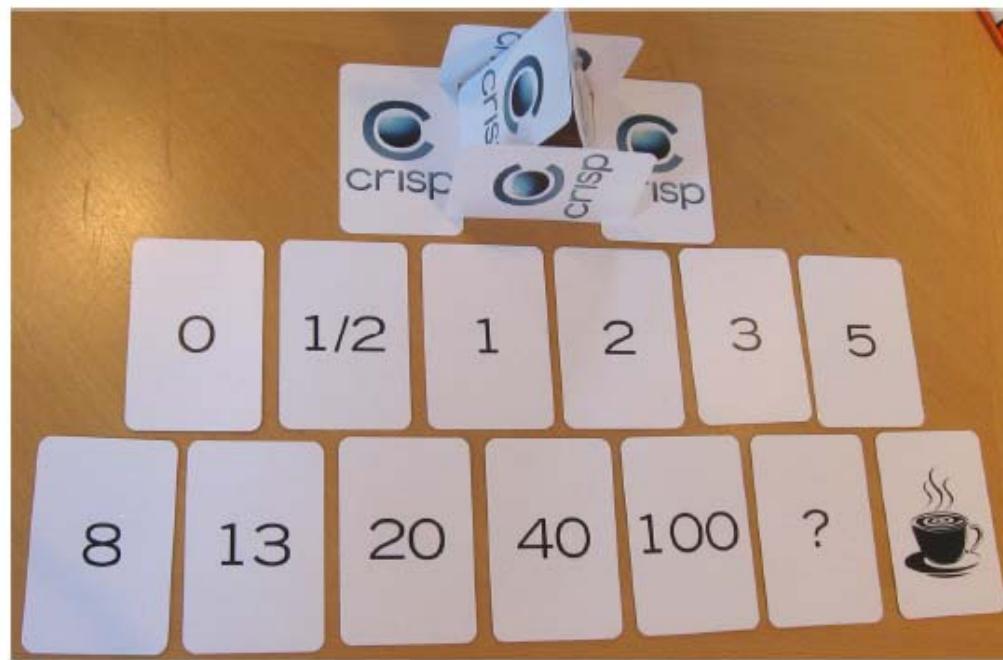
Si os sentís confusos con frecuencia acerca de la definición de terminado (cosa que nos ocurría al principio) probablemente deberíais tener un campo "definición de terminado" en cada historia individual.

Estimación de tiempos usando planning poker

La estimación es una labor de equipo: todos los miembros del equipo deben involucrarse en estimar cada historia. ¿Por qué?

- A la hora de planificar, normalmente no sabemos exactamente quién implementará qué partes de cada historia.
- Las historias normalmente involucran a bastantes personas y de diferentes áreas de experiencia (diseño de interfaz de usuario, programación, pruebas, etc.).
- Para poder proporcionar una estimación, un miembro del equipo necesita comprender de alguna forma de qué trata la historia. Pidiendo a todo el mundo que estime la historia nos aseguramos de que cada miembro del equipo comprende de qué trata cada elemento. Esto incrementa las posibilidades de que unos miembros del equipo ayuden a otros durante el Sprint. También mejora las posibilidades de que aparezcan pronto las preguntas importantes sobre cada historia.
- Cuando pedimos a todo el mundo que de estimaciones muchas veces encontramos discrepancias en las que dos miembros del equipo tienen estimaciones tremadamente distintas sobre la misma historia. Es mejor descubrir tales cosas al principio.

Si pides al equipo que proporcione una estimación, normalmente la persona que entiende mejor la historia será el primero en soltar una. Desafortunadamente, esto afectará severamente a las estimaciones de los demás. Hay una técnica excelente para evitar esto: se llama planning poker (creo que el término fue acuñado por Mike Cohn).



Cada miembro del equipo cuenta con una baraja de 13 cartas, como las que se muestran en la imagen. Cada vez que hay que estimar una historia, cada miembro del equipo selecciona una carta que representa su estimación de tiempo (en puntos de historia) y la coloca boca abajo en la mesa.

Cuando todos los miembros del equipo han preparado sus cartas, se les da la vuelta al mismo tiempo. Así obligamos a cada miembro del equipo a pensar por sí mismo en lugar de seguir la estimación de otro.

Si hay mucha discrepancia entre dos estimaciones, el equipo discute las diferencias y trata de construir una imagen común del trabajo necesario para la historia. Pueden hacer algún tipo de división en tareas. Despues, el equipo estima de nuevo. Este bucle se repite hasta que la estimación de tiempo converge, es decir, que todas las estimaciones sean aproximadamente las mismas para esa historia.

Es importante que los miembros del equipo recuerden que deben estimar el total de tiempo necesario para la historia. No solamente "su" parte del trabajo. El encargado de pruebas no debería estimar solo la cantidad de trabajo de pruebas.

Date cuenta de que la secuencia de números no es lineal. Por ejemplo, no hay nada entre 40 y 100. ¿Por qué?

Esto es para evitar una falsa sensación de exactitud para las estimaciones de tiempo más grandes. Si una historia se estima aproximadamente en 20 puntos, no es relevante discutir si deberían ser 20, 18 o 21. Todo lo que sabemos es que es una historia grande y es difícil de estimar. Así que 20 es nuestra idea aproximada.

Quieres estimaciones más detalladas? Divide la historia en historias más pequeñas y trata de estimar las historias pequeñas.

Y no, no se puede hacer trampas combinando un 5 y un 2 para hacer un 7. Tienes que escoger entre 5 y 8, no hay 7.

Algunas cartas especiales sobre las que hablar:

- 0 = "esta historia ya está hecha" o "esta historia es prácticamente nada, apenas unos minutos de trabajo".
- Z = "no tengo ni la más remota de las ideas. Nada".
- Taza de café = "estoy demasiado cansado para pensar. Tomemos un descanso".

Clarificando historias

Lo peor ocurre cuando el equipo demuestra orgulloso una nueva funcionalidad en la demo de Sprint y el Dueño de Producto frunce el ceño y dice "vale, muy bonito, pero ¡eso no es lo que yo pedí!".

¿Cómo te aseguras de que el concepto que tiene el Dueño de Producto sobre una historia coincide con el concepto del equipo? ¿O de que cada miembro del equipo tenga el mismo concepto sobre la historia? Bueno, no se puede. Pero hay algunas técnicas simples para identificar los malentendidos más flagrantes. La técnica más simple es simplemente asegurarse de que todos los campos están llenos para cada historia (o más específicamente, para cada historia con suficiente importancia como para ser considerada en este Sprint).

Ejemplo 1:

El equipo y el Dueño de Producto están contentos con el plan de Sprint y están listos para terminar con la reunión. El Scrum Master dice "esperad un segundo, esta historia llamada "añadir usuario", no tiene estimación. Hagamos una estimación." Tras un par de rondas de planning poker el equipo acuerda 20 puntos de historia, haciendo que el Dueño de Producto se levante preso de un ataque de rabia, "¡¿Como?!". Tras unos minutos de discusión acalorada, resulta que el equipo entendió mal el alcance de "añadir usuario" y pensaban que significaba "un bonito interfaz Web para añadir, eliminar, buscar usuarios", cuando el Dueño de Producto quería decir "añadir usuarios a mano haciendo sentencias SQL contra la base de datos". Estiman de nuevo y acaban con 5 puntos de historia.

Ejemplo 2:

El equipo y el Dueño de Producto están contentos con el plan de Sprint y están listos para acabar la reunión. El Scrum Master dice "esperad un segundo, esta historia que se llama "añadir usuario", ¿cómo debería demostrarse?". Se producen algunos murmullos y tras un minuto alguien dice "bueno, primero nos autenticamos en la Web y luego..." y el Dueño de Producto interrumpe "¿autenticarse en la Web? No, no, no, esta funcionalidad no debería de ninguna forma ser parte del sitio Web, debería ser sólo un script muy sencillo para los administradores técnicos".

La descripción de "como probarlo" de cada historia puede (y debe) ser muy breve. De otra forma no terminarás con la planificación de Sprint a tiempo. Es básicamente una descripción básica a alto nivel, en castellano simple, de cómo ejecutar el escenario de pruebas más común manualmente. "Haz esto, entonces haz lo otro y verifica aquello". He descubierto que estas descripciones simples a

menudo descubren malentendidos importantes sobre el alcance de una historia. Menos mal que las descubrimos pronto, ¿verdad?

Dividiendo historias en historias más pequeñas

Las historias no deberían ser demasiado pequeñas ni demasiado grandes (en términos de estimación). Si tienes un montón de historias de 0.5 puntos probablemente vas a ser una víctima de la microgestión. Por otra parte, una historia de 40 puntos corre un importante riesgo de acabar parcialmente completa, lo que no aporta valor a tu empresa y simplemente incremente la administración. Lo que es más, si tu velocidad estimada es 70 y tus dos historias más prioritarias pesan 40 puntos cada una, la planificación se vuelve difícil. Tienes que elegir entre comprometerte a más de lo que deberías (escoger las dos historias) o a menos (elegir sólo una).

Creo que casi siempre es posible dividir una historia grande en historias más pequeñas. Simplemente asegúrate de que las historias pequeñas siguen representando entregables con valor de negocio.

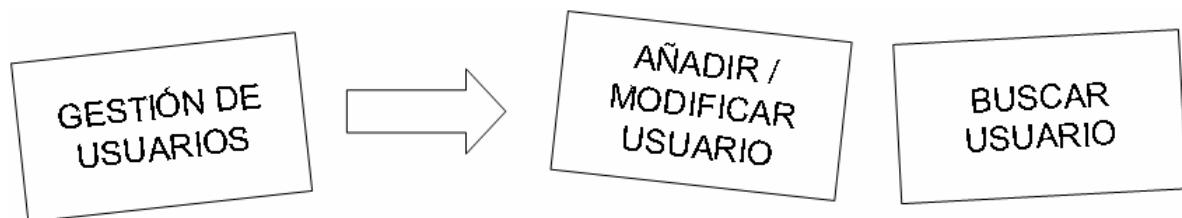
Normalmente buscamos historias con estimaciones de 2 a 8 días/hombre. Nuestra velocidad es usualmente 40-60 para un equipo típico, así que eso nos da para unas 10 historias por Sprint. A veces sólo 5, y a veces hasta 15. Es un número de tarjetas gestionable con el que trabajar.

Dividiendo las historias en tareas.

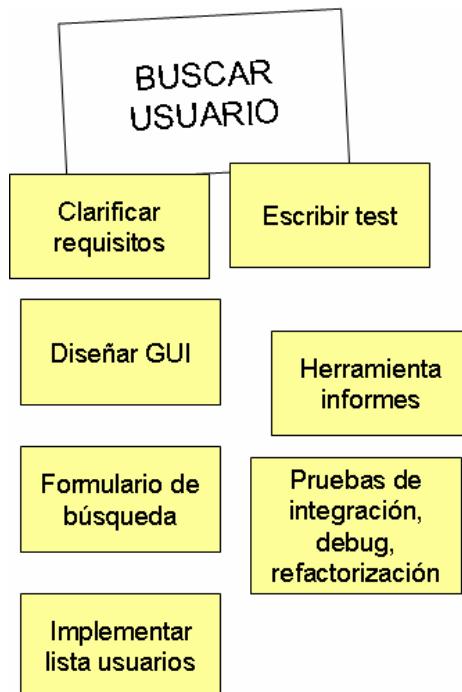
Un momento, ¿Cuál es la diferencia entre historias y tareas? Una pregunta muy válida.

La diferencia es muy simple. Las historias son entregables de los que el Dueño de Producto se preocupa. Las tareas son no-entregables, o aspectos de los que el Dueño de Producto no se preocupa.

Ejemplo de división de una historia en historias más pequeñas:



Ejemplo de división de una historia en tareas:



Algunas observaciones interesantes:

- Los equipos que están empezando con Scrum son reticentes a perder tiempo dividiendo un montón de historias en tareas desde el principio. Algunos piensan que es una aproximación tipo "cascada".
- Para historias que se entienden bien, es tan fácil hacer esto desde el principio como hacerlo más tarde,
- Este tipo de división frecuentemente revela trabajo adicional que hace que las estimaciones suban, con lo que se consigue un plan de Sprint más realista.
- Este tipo de división desde el principio hace que los Scrum diarios sean notablemente más eficientes.
- Incluso si la división es inexacta y cambia una vez que empezamos, todas las ventajas anteriormente mencionadas siguen siendo válidas.

Así que tratamos que el tiempo disponible para la planificación de Sprint sea suficientemente largo como para que de tiempo a hacer esto, pero si el tiempo se acaba lo dejamos como está (ver "Dónde trazar la línea" más adelante).

Nota – nosotros practicamos TDD (test driven development, desarrollo orientado a test) lo que en la práctica significa que la primera tarea para casi todas las historias es "escribir una prueba" y la última es "refactorizar" (es decir, mejorar la legibilidad del código y eliminar la duplicación).

Definiendo el sitio y la hora para el Scrum diario

Uno de los productos frecuentemente olvidados de la planificación de Sprint es "un sitio y una hora determinados para el Scrum diario". Sin ello, tu Sprint está condenado a un mal comienzo. El primer Scrum diario es esencialmente el lanzamiento, donde todo el mundo decide por dónde va a empezar a trabajar.

Yo **prefiero las reuniones por la mañana**. Pero debo admitir que realmente no hemos probado a hacer Scrum diario por las tardes o al mediodía.

Desventaja de Scrums por las tardes: cuando llegas al trabajo por la mañana, tienes que acordarte de qué le dijiste a la gente sobre lo que deberían hacer hoy.

Desventaja de los Scrums por las mañanas: cuando llegas al trabajo por la mañana, debes acordarte de qué hiciste ayer para informar sobre ello hoy.

En mi opinión, la primera desventaja es peor, ya que lo más importante es *lo que vas a hacer, no lo que hiciste ayer*.

Nuestro procedimiento por defecto es seleccionar la hora más temprana a la que ningún miembro del equipo vaya a quejarse. **Usualmente las 9:00, 9:30 o 10:00.** **Lo más importante** es que sea a una hora a la que todo el equipo acepte con total convencimiento.

Dónde trazar la línea

OK, el tiempo se está agotando. De todos los asuntos que queremos resolver durante la planificación de Sprint, **¿qué abandonamos si nos quedamos sin tiempo?**

Bueno, normalmente uso la siguiente lista de prioridades:

Prioridad 1: Una **meta de Sprint** y **una fecha para la demo**. Esto es lo **mínimo** que necesitas para comenzar un Sprint. El equipo tiene una meta y una fecha de finalización, y pueden trabajar directamente con la **Pila de Producto**. Apesta, sí, y deberías considerar seriamente organizar una nueva reunión de planificación de Sprint mañana mismo, pero **si realmente necesitas que el Sprint comience entonces probablemente puedes hacerlo con esto**. Para ser honesto, **yo nunca he empezado un Sprint con tan poca información**.

Prioridad 2: Lista de qué historias ha aceptado **terminar el equipo en este Sprint**.

Prioridad 3: Una estimación para cada historia del Sprint.

Prioridad 4: "Como probarlo", relleno para cada historia del Sprint.

Prioridad 5: **Cálculos de velocidad y recursos**, como **chequeo** de la **planificación del Sprint**. Incluyendo una lista de los miembros del equipo y sus compromisos (de otra forma, no podrías calcular la velocidad).

Prioridad 6: Un **sitio y hora específicos para la realización del Scrum diario**. Solo necesitas un momento para decidirlo, pero si te quedas sin tiempo el Scrum Master puede simplemente decidir esto después de la reunión y mandar un correo a todo el mundo.

Prioridad 7: Historias divididas en tareas. Esta **división** puede sin embargo **hacerse diariamente durante los Scrum diarios**, pero **interferirá ligeramente el flujo del Sprint**.

Historias técnicas

He aquí un **asunto complejo: historias técnicas**. O elementos no-funcionales, o como quieras llamarlos.

Me refiero a **cosas que deben hacerse pero que no son un entregable ni están directamente relacionadas con ninguna historia específica, y no son de valor inmediato para el Dueño de Producto**.

Las llamamos “**historias técnicas**”.

Por ejemplo:

- **Instalar un servidor de compilación continua:**
 - Por qué debe hacerse: porque ahorra cantidades inmensas de tiempo a los desarrolladores y reduce el riesgo de problemas explosivos de integración al final de la iteración.
- **Escribir una descripción general del diseño:**
 - Por qué debe hacerse: porque los desarrolladores olvidan constantemente el diseño general, y entonces escriben código inconsistente. Necesitan una visión global documentada para mantener a todo el mundo en la misma línea de diseño.
- **Refactorizar la capa de acceso a datos:**
 - Por qué debe hacerse: porque la capa de acceso a datos se ha vuelto realmente desordenada y le está costando tiempo a todo el mundo debido a la confusión y los errores innecesarios. Limpiar el código ahorraría tiempo a todo el mundo y mejoraría la robustez del sistema.
- **Actualizar Jira** (seguimiento de errores)
 - Por qué debe hacerse: la actual versión es demasiado lenta y falla demasiado. Actualizar a una nueva versión ahorrará tiempo a todo el mundo.

¿Son historias en el sentido normal? ¿O son tareas que no están conectadas a ninguna historia específica? ¿Quién las prioriza? ¿Debería involucrarse el Dueño de Producto en estos asuntos?

Hemos **experimentado** mucho con **diferentes maneras de manejar las historias técnicas**. Hemos **intentado tratarlas como historias de primera clase**, como todas las demás. Esto no funcionó bien, ya que cuando el Dueño de Producto priorizaba la Pila de Producto era como comparar peras con manzanas. De hecho, por razones obvias, las historias técnicas obtenían siempre mínima prioridad con razonamientos del tipo “sí, chavales, seguro que la compilación continua es muy importante y todo eso, pero construyamos primero algo que nos permita facturar, ¿vale? Entonces podréis añadir vuestras chucherías técnicas más adelante, ¿OK?”

En algunos casos el Dueño de Producto tiene razón, pero a menudo no es así. Hemos concluido que el **Dueño de Producto no siempre está cualificado para manejar estos compromisos**. Así que esto es lo **que hacemos**:

- 1) Intentamos evitar las historias técnicas. Busca efusivamente formas de transformar las **historias técnicas en historias normales con valor de negocio mensurable**. Así el **Dueño de Producto tendrá mejores oportunidades para realizar decisiones correctas entre los pros y los contras**.

2) Si no podemos transformar una historia técnica en una historia normal, intentamos ver si es posible hacerla como una tarea dentro de otra historia. Por ejemplo, "refactorizar la capa de acceso a datos" podría ser una tarea dentro de la historia "editar usuario", ya que esto involucra utilizar la capa de acceso a datos.

3) Si lo anterior falla, definirla como historia técnica y mantener una lista separada con dichas historias. Permitimos al Dueño de Producto que vea dicha lista, pero no que la modifique. Usamos los factores de dedicación y la velocidad estimada para negociar con el Dueño de Producto y sacar algo de tiempo del Sprint para implementar estas historias.

Ejemplo (un diálogo muy similar a este ocurrió durante una de nuestras reuniones de planificación de Sprint):

- **Equipo:** "Tenemos algunos asuntos técnicos internos que deben hacerse. Nos gustaría presupuestar un 10% de nuestro tiempo para ello, es decir, reducir el factor de dedicación del 75% al 65%. ¿Os parece bien?"
- **Dueño de producto:** "¡Y una leche! ¡No tenemos tiempo!"
- **Equipo:** "Bueno, mira el último Sprint (todas las cabezas se giran hacia la velocidad que está apuntada en la pizarra). Nuestra velocidad estimada fue 80, y la velocidad real fue de 30, ¿correcto?"
- **Dueño de Producto:** "¡Exacto! ¡Así que no tenemos tiempo para asuntos técnicos internos! ¡Tenemos que añadir nuevas funcionalidades!"
- **Equipo:** "Bueno, la razón por la que nuestra velocidad fue tan mala fue que pasamos demasiado tiempo intentando conseguir versiones consistentes para probar".
- **Dueño de Producto:** "Sí, ¿Y?"
- **Equipo:** "Así que proponemos dedicar un 10% de nuestro tiempo del Sprint para implementar un servidor de compilación continua y otras cosas que nos ayudarán a hacer la integración menos penosa. Esto incrementará nuestra velocidad al menos un 20% para los próximos Sprints, ¡para siempre!."
- **Dueño de Producto:** "¿En serio? ¿Y por qué no lo hicimos en el último Sprint entonces?"
- **Equipo:** "Eh... porque no querías que lo hiciéramos..."
- **Dueño de Producto:** "Oh, um, bueno, vale, parece una buena idea hacerlo ahora entonces..."

Por supuesto, la otra opción es simplemente mantener al Dueño de Producto fuera del ciclo o darle un factor de dedicación no negociable. Pero no hay excusa para no intentar llegar a un consenso primero.

Si el Dueño de Producto es un tipo competente y razonable (y nosotros hemos tenido suerte en eso) yo sugiero mantenerle lo más informado posible y permitirle establecer las prioridades generales. La transparencia es uno de los valores principales de Scrum, ¿no es cierto?

Sistema de seguimiento de errores vs. Pila de Producto

He aquí un asunto complejo. Excel es un formato estupendo para la Pila de Producto. Pero aun así necesitas un sistema de seguimiento de errores, y Excel probablemente no funcione bien. Nosotros usamos Jira.

Así que, ¿cómo incluimos los elementos de Jira en la reunión de Planificación de Sprint? Quiero decir que no sirve simplemente ignorarlos y concentrarse solo en las historias.

Hemos intentado varias estrategias:

- 1) El Dueño de Producto imprime los elementos de Jira más importantes, los trae a la reunión de planificación de Sprint y los coloca en la pared junto al resto de historias (especificando así implícitamente la prioridad de estos elementos comparados con las otras historias).
- 2) El Dueño de Producto crea historias que se refieren a los elementos de Jira. Por ejemplo, "Arreglar los errores más críticos de informes de backoffice, Jira-124, Jira-126 y Jira-180".
- 3) La corrección de errores se considera algo fuera del Sprint. El equipo mantiene un factor de dedicación suficientemente bajo (por ejemplo, el 50%) para asegurar que tienen tiempo suficiente para arreglar errores. Entonces simplemente se asume que el equipo pasará parte de su tiempo arreglando los errores reportados en Jira.
- 4) Hacer la Pila de Producto en Jira (es decir, abandonar Excel). Tratar los errores como cualquier otra historia.

No hemos acabado de determinar qué estrategia es mejor para nosotros. De hecho, varía de equipo a equipo y de Sprint a Sprint. Tiendo no obstante a favorecer la primera estrategia. Es sencilla y agradable.

¡Por fin acabó la reunión de planificación de Sprint!

sprint planning

¡Wow, nunca creí que este capítulo sobre las reuniones de planificación de Sprint sería tan largo! Supongo que eso refleja mi opinión de que la reunión de planificación de Sprint es lo más importante que se hace en Scrum. Emplea una buena cantidad de esfuerzo en hacerla bien, y el resto será mucho más fácil.

La reunión de planificación de Sprint tiene éxito si todo el mundo (todos los miembros del equipo y el Dueño de Producto) sale de la reunión con una sonrisa y se levanta a la mañana siguiente con una sonrisa, y hacen su primer Scrum diario con una sonrisa. A partir de ahí, por supuesto, toda clase de cosas pueden ir horriblemente mal, pero al menos no puedes echarle la culpa al plan de Sprint :o)

5

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Como comunicamos los Sprints

Es importante mantener a toda la compañía informada sobre lo que está ocurriendo. De otra forma, la gente se queja constantemente o, incluso peor, hacen falsas presunciones sobre lo que está ocurriendo.

Utilizamos una “página de información de Sprint” para esto.

Equipo Jackass, Sprint 15	
Objetivo de Sprint:	- ¡Versión beta lista!
Pila de Sprint:	<ul style="list-style-type: none"> - Deposito (3) - Herramienta de migración (8) - Login en backoffice (5) - Administración de usuarios de backoffice (5)
Velocidad estimada: 21	
Calendario:	<ul style="list-style-type: none"> - Periodo de Sprint: 6/11/2006 a 24/11/2006 - Scrum diario: 9:30-9:45 en la sala del equipo - Demo de Sprint: 24/11/2006, 13:00 en la cafetería
Equipo:	<ul style="list-style-type: none"> - Jim - Erica (Scrum Master) - Tom (75%) - Eva - John

A veces incluimos también información sobre como se demostrará cada historia.

Tan pronto como sea posible **tras la reunión de planificación de Sprint, el Scrum Master crea esta página, la pone en el Wiki y manda un spam a toda la compañía:**

Asunto: Comienzo del Sprint 15 de Jackass

¡Hola a todos! El equipo Jackass ha comenzado con el Sprint 15. Nuestro objetivo es demostrar la versión el 24 de noviembre.

Podéis ver la página de información del Sprint en:
<http://wiki.miempresa.com/jackass/sprint15>

También tenemos un "panel de control" en el **Wiki**, que **enlaza con todos los Sprints** que estén teniendo lugar:

Panel de control corporativo:

Sprints en curso:

- [Equipo X – Sprint 11](#)
- [Equipo Y – Sprint 12](#)
- [Equipo Z – Sprint 1](#)

sprint review

Adicionalmente, el **Scrum Master imprime la página de información de Sprint y la pega en la pared fuera de la sala del equipo**. Así, todo el que **pasa cerca** puede **ver la página de información de Sprint y averiguar qué está haciendo el equipo**. Ya que esto incluye el **sitio y la hora** a la que se hace el **Scrum diario** y la **demo de Sprint**, ya sabe a donde ir para averiguar más.

Cuando el **Sprint se acerca a su final**, el **Scrum Master recuerda a todo el mundo que se acerca la demo**:

Asunto: Demo del Sprint 15 de Jackass mañana en la cafetería

¡Hola a todos! Os invitamos a asistir a la demo de Sprint que tendrá lugar mañana a las 13:00 en la cafetería.

Demostraremos una versión beta lista para liberar.

Podéis ver más en la página de Sprint:
<http://wiki.miempresa.com/jackass/sprint15>

Con todo esto, nadie tiene excusa para no saber lo que realmente está ocurriendo.

6

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

spring backlog

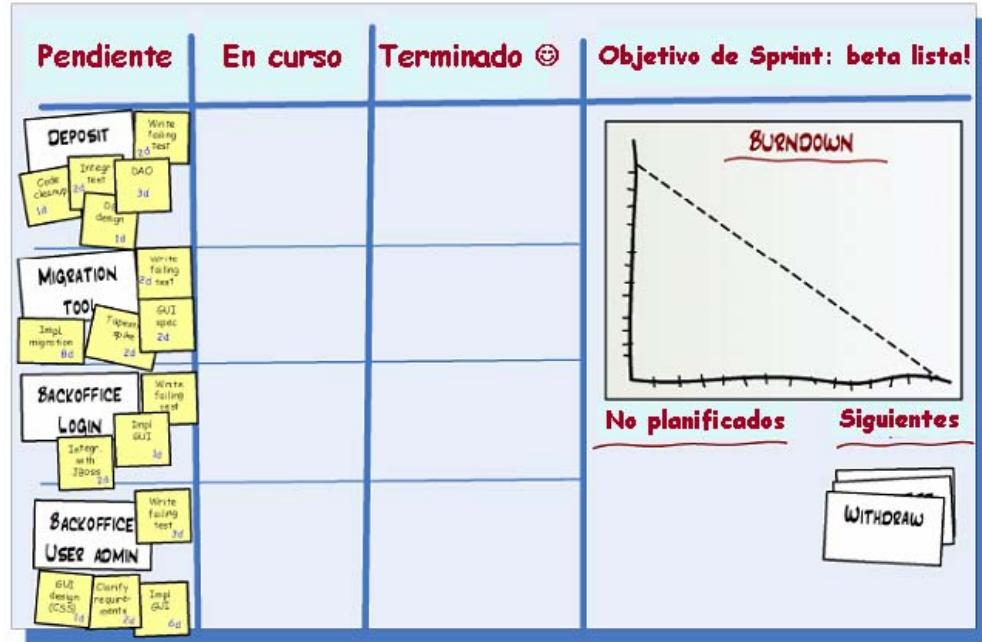
Como hacemos Pilas de Sprint

¿Has llegado hasta aquí? Wow, buen trabajo. Así que ahora que hemos completado la reunión de planificación de Sprint y hemos informado al mundo sobre nuestro nuevo y reluciente Sprint, es hora de que el Scrum Master cree una Pila de Sprint. Esto debe hacerse antes de la reunión de planificación de Sprint pero antes del primer Scrum diario. Despues

Formato de la Pila de Sprint

Hemos experimentado con diferentes formatos para la Pila de Sprint, incluyendo Jira, Excel y un tablón físico en la pared. Al principio usábamos Excel sobre todo, y hay muchas plantillas abiertamente disponibles para hacer Pilas de Sprint, incluyendo algunas que generan automáticamente los diagramas de burn-down y cosas así. Podría hablar mucho sobre cómo hemos refinado nuestras Pilas de Sprint basadas en Excel. Pero no lo haré. Ni siquiera voy a incluir ejemplos aquí. En lugar de ello voy a describir en detalle el formato que he encontrado más efectivo para la Pila de Sprint: ¡Una tabla de tareas en la pared!

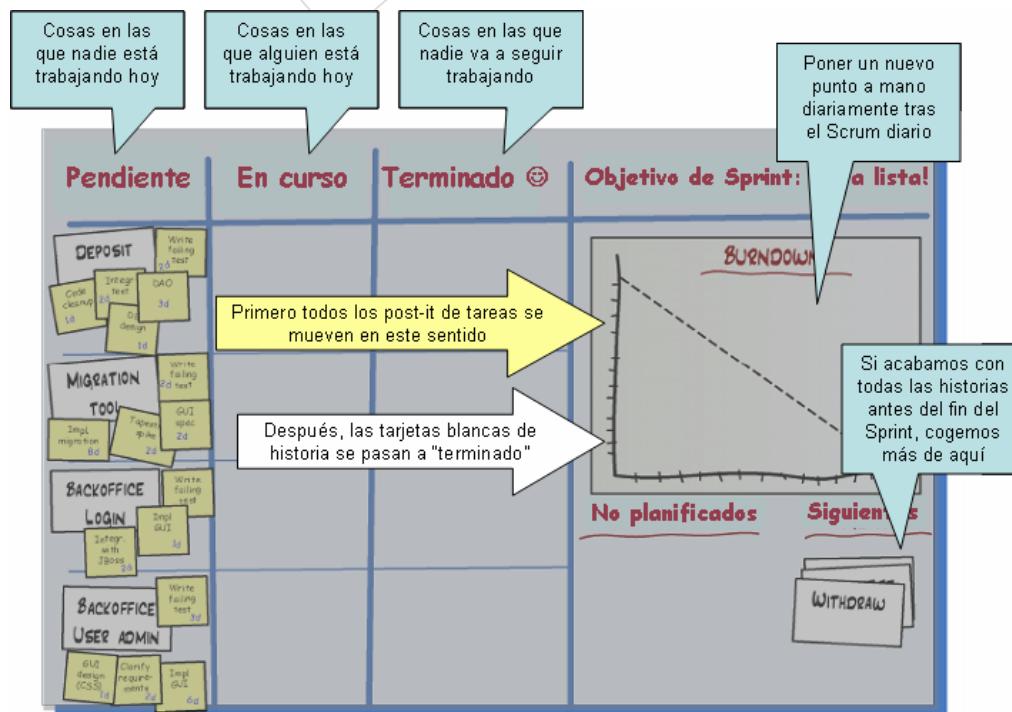
Busca una pared que no esté usada o contenga cosas inútiles como el logo de la compañía, viejos diagramas o cuadros feos. Vacía la pared (pide permiso solo si es absolutamente necesario). Pega una gran, gran hoja de papel (por lo menos de 2x2 metros, o 3x2 para un equipo grande). Y entonces haz esto:



Probablemente podrías usar una pizarra, pero es un desperdicio. Si es posible, guarda las **pizarras para garabatos de diseño** y usa las **paredes que no sean pizarras para los tablones de tareas**.

NOTA – Si usas **post-it** para las tareas, **no olvides pegarlos usando cinta adhesiva**, o encontrarás todos tus pos-it en un montoncito en el suelo cualquier día

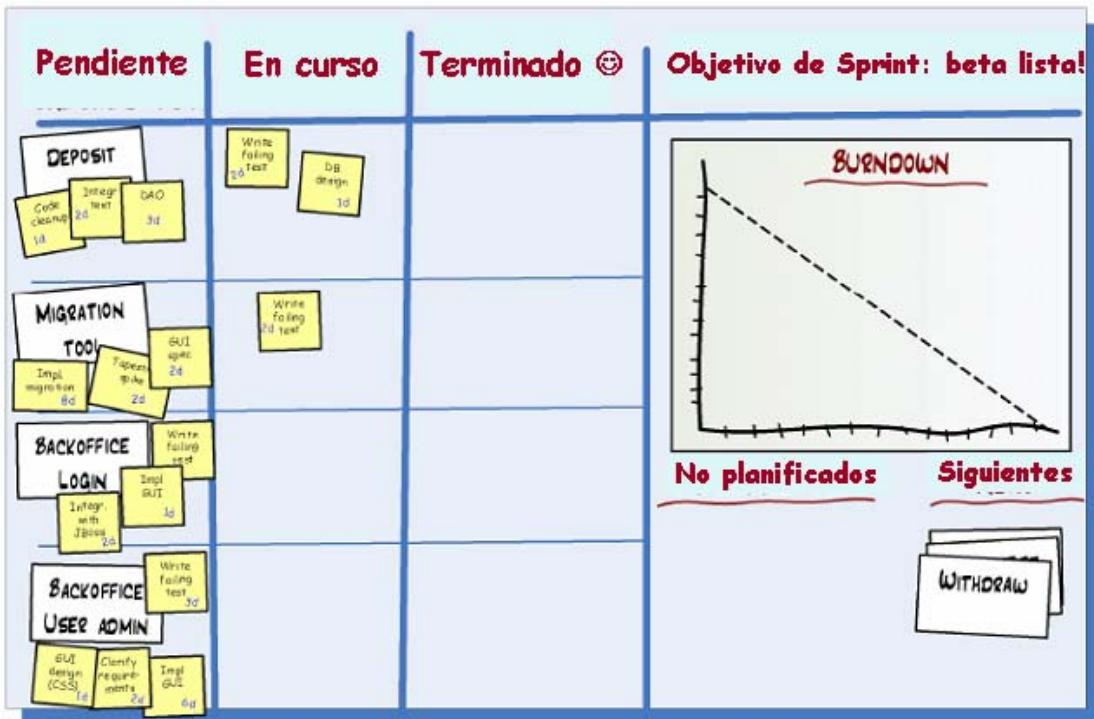
Cómo funciona el tablón de tareas



Por supuesto puedes añadir toda clase de columnas adicionales. “Esperando test de integración”, por ejemplo. O “cancelado”. De todas formas, antes de complicar las cosas, piénsalo profundamente. ¿Es esta extensión realmente, realmente necesaria? He descubierto que la simplicidad es extremadamente valiosa en estas cosas, así que sólo añado complicaciones adicionales si el coste de no hacerlo es demasiado grande.

Ejemplo 1 – tras el primer Scrum diario

Después del primer Scrum, el tablón puede aparecer así:



Como puedes ver, tres tareas están “en proceso”, es decir, el equipo estará trabajando en estos elementos hoy.

A veces, para equipos más grandes, una tarea queda atascada “en progreso” porque nadie recuerda quién estaba trabajando en ella. Si esto ocurre a menudo en un equipo, usualmente introducimos políticas como etiquetar cada tarea en progreso con el nombre de la persona que la ha emprendido.

Ejemplo 2 – tras unos cuantos días

Unos días más tarde el tablón puede aparecer de la siguiente manera:



Como puede observarse, se ha completado la historia "Depósito" (es decir, ha sido chequeada en el repositorio de código fuente, testeada, refactorizada, etcétera). La herramienta de migración (segunda historia) está parcialmente completada. La tercera historia ("backoffice login") ha comenzado, y la cuarta ("backoffice user admin.") no ha empezado aun.

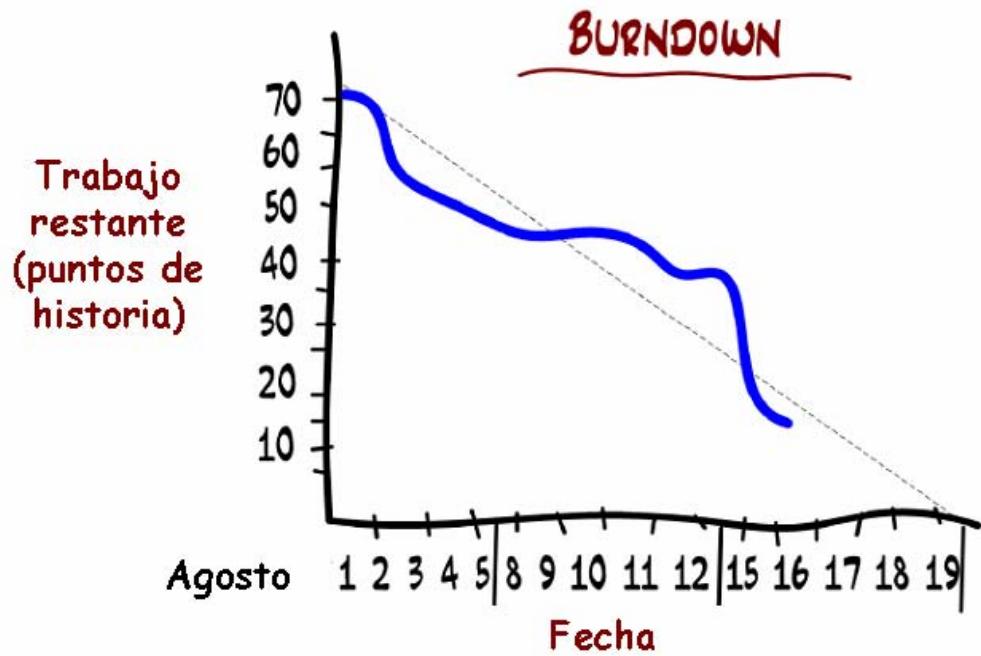
Tenemos tres elementos no planificados, como puede verse abajo a la derecha. Esto es útil para recordar cuando hagamos retrospectiva del Sprint.

Aquí tenemos un ejemplo de una Pila de Sprint real casi al final de un Sprint. Se vuelve bastante liosa conforme el Sprint progresá, pero no pasa nada, ya que tiene una vida muy corta. En cada nuevo Sprint, creamos una limpia, fresca y nueva Pila de Producto.



Como funciona el diagrama burn-down

Vamos a fijarnos en el **diagrama burn-down**:



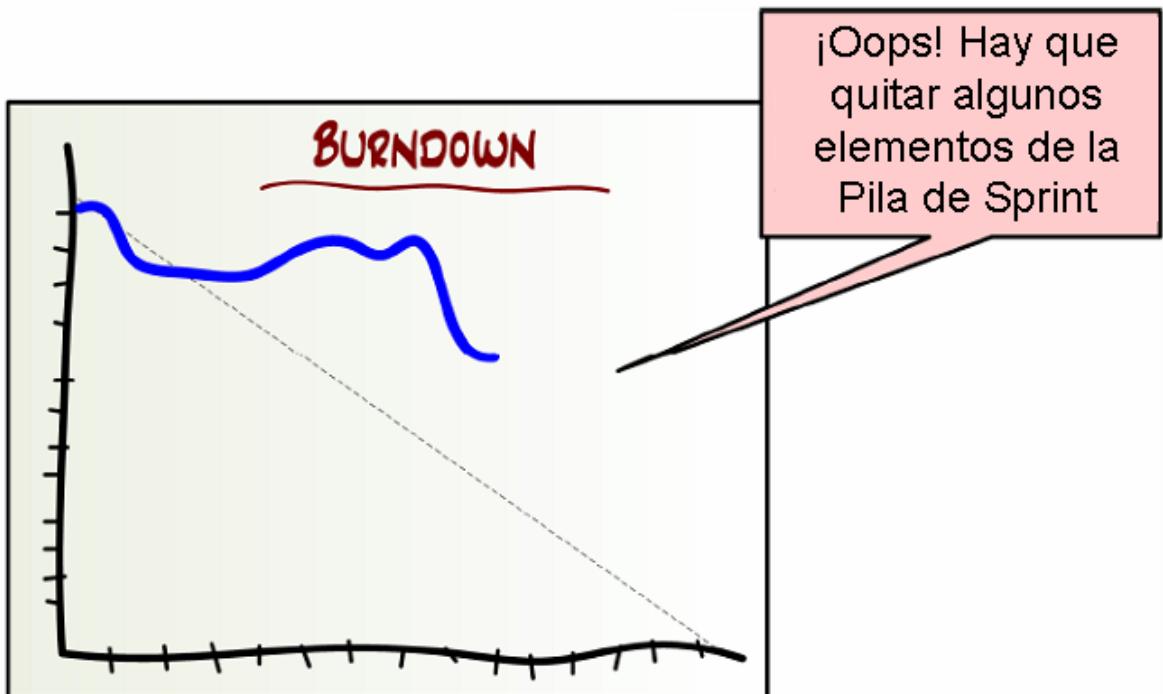
Este diagrama muestra que:

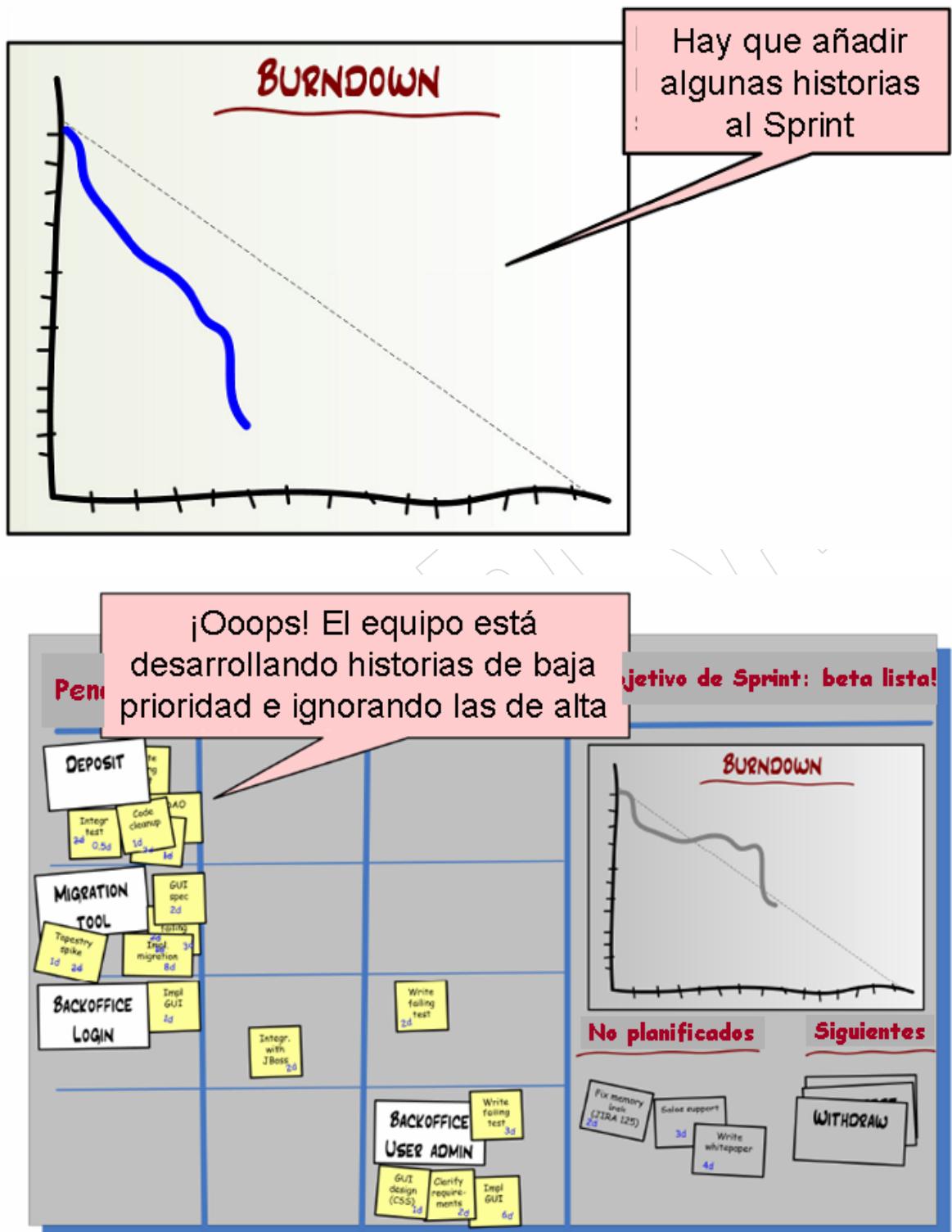
- En el primer día del Sprint, 1 de Agosto, el equipo estimó que habían aproximadamente 70 puntos de historia en los que trabajar. Esta era, consecuentemente, la velocidad estimada para todo el Sprint.
- El 16 de Agosto el equipo estima que quedan aproximadamente 15 puntos de historia por hacer. La línea de puntos nos muestra que estamos incluso algo avanzados respecto a la planificación, es decir, que a este paso completaríamos todo al final del Sprint.

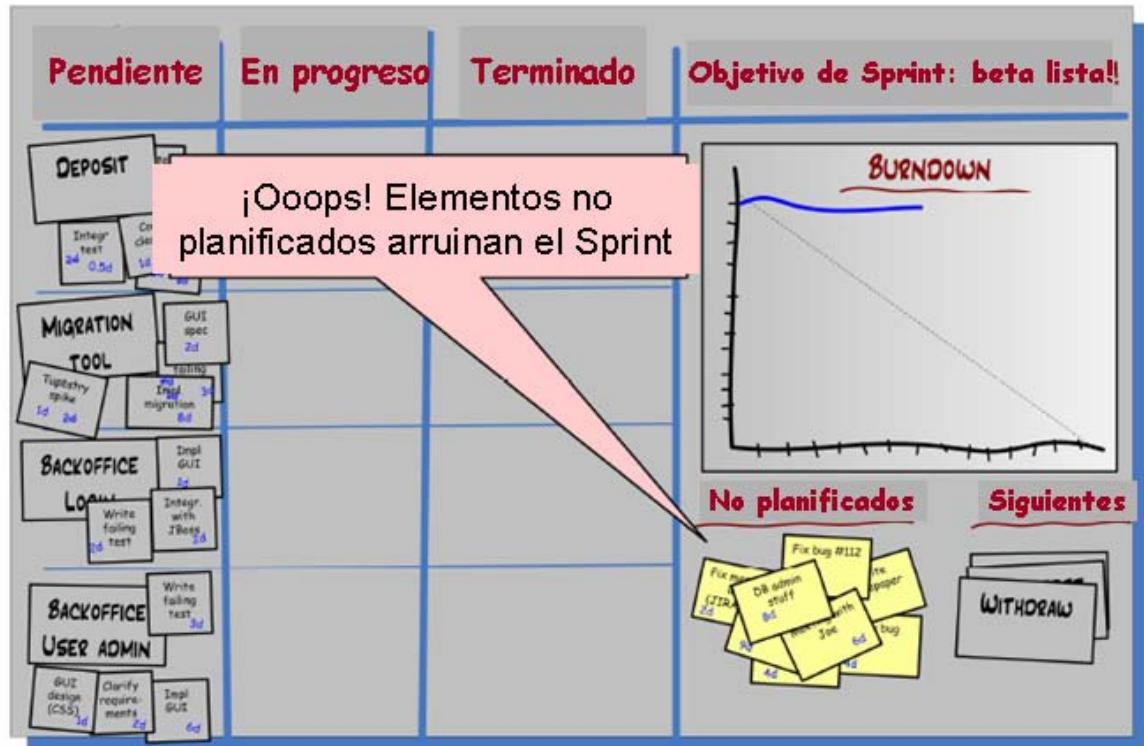
Nos saltamos los fines de semana en el eje X ya que rara vez se trabaja los fines de semana. Solíamos incluir los fines de semana, pero esto hacía los burn-down algo confusos ya que se “aplanaban” durante los fines de semana, lo que parecía una señal de peligro.

Señales de alarma en el burn-down

Un vistazo rápido a la tabla de tareas debería dar a cualquiera una indicación de cómo de bien está progresando el Sprint. El Scrum Master es responsable de asegurarse de que el equipo actúa ante señales de alarma como:







Hey, ¿Qué pasa con la trazabilidad?

La mejor trazabilidad que puedo ofreceros con este modelo es tomar una foto del tablón de tareas con una cámara digital todos los días. Sólo si es necesario. Yo lo hago a veces, pero luego nunca encuentro la necesidad de usar esas fotos.

Si la trazabilidad es muy importante para ti, entonces quizás el tablón de tareas no sea la mejor solución para ti.

Pero aun así te sugiero que intentes estimar el valor real de una trazabilidad detallada del Sprint. Una vez que el Sprint está terminado y se ha entregado código funcional y la documentación está lista, ¿realmente le importa a alguien cuántas historias se habían completado el día 5 del Sprint? ¿A alguien le importa cuál era la estimación de la tarea "escribir una prueba para Depósitos"?

Estimando en días vs horas

En la mayoría de los libros y artículos sobre Scrum encontrarás que las tareas se estiman en horas, no en días. Solíamos hacer eso. Nuestra fórmula general era: 1 día-hombre real = 6 horas-hombre reales.

Ahora hemos dejado de hacer eso, al menos en la mayoría de nuestros equipos, por las siguientes razones:

- Las estimaciones en horas-hombre eran demasiado granulares. Esto provocaba una tendencia a estimar muchas tareas en 1-2 horas, y con ello a la microgestión.
- De todas formas resultó que todo el mundo estaba pensando en términos de días-hombre, y simplemente multiplicaban por 6 para escribir las horas-hombre. "Hmmm, esta tarea debería llevar más o menos un día. Oh, tengo que escribirlo en horas, pues entonces escribo seis horas".
- Dos unidades diferentes causan confusión. ¿Esa es la estimación en días-hombre o en horas-hombre?".

Así que ahora usamos días-hombre como base para todas nuestras estimaciones (aunque lo seguimos llamando puntos de historia). Nuestro valor más bajo es 0.5, es decir, cualquier tarea que es menor de 0.5 se elimina o se combina con otras tareas, o se le deja una estimación de 0.5 (no causa mucho daño sobre-estimar un poco). Elegante y simple.

7

Versión gratuita on-line.

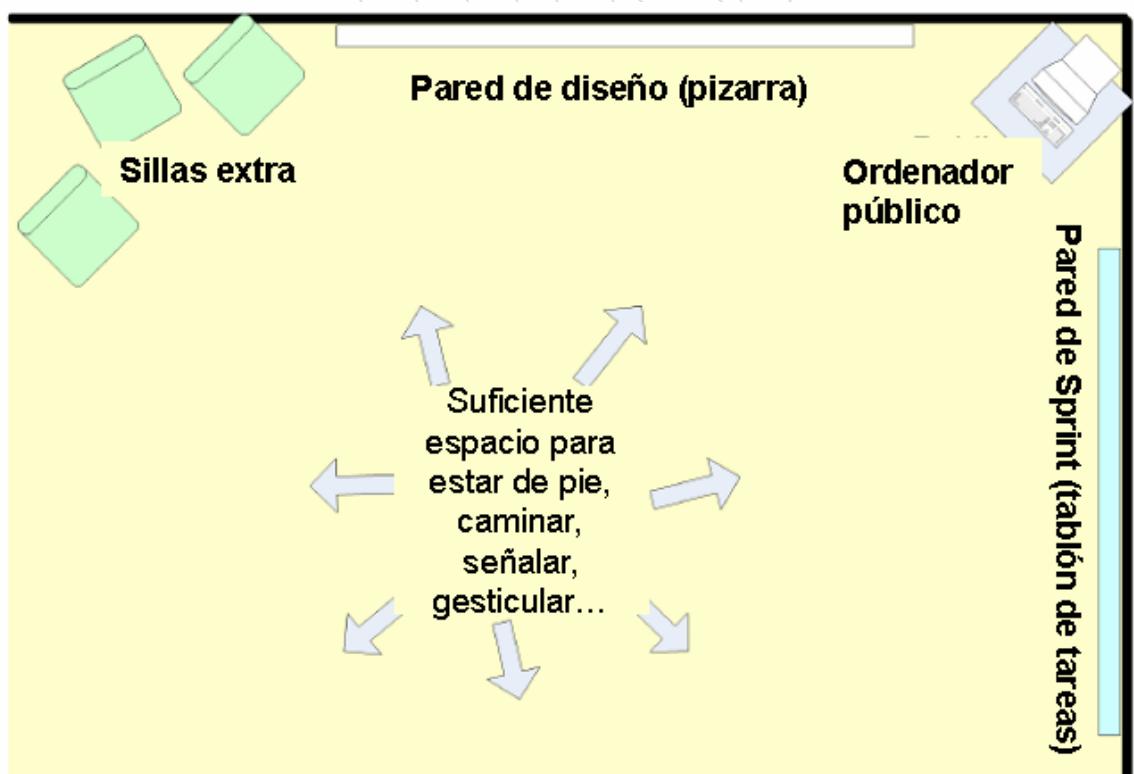
Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Como distribuimos la sala del equipo

La esquina de diseño

Me he dado cuenta de que la mayor parte de las discusiones interesantes y valiosas sobre el diseño tienen lugar espontáneamente frente al tablón de tareas. Por esta razón, tratamos de organizar esta área como una "esquina de diseño" explícitamente.



Esto es muy útil. No hay mejor manera para obtener una visión general del sistema que estar de pie en la esquina de diseño y observar ambas paredes, entonces echar un vistazo en el ordenador y probar la última compilación del sistema (si tienes suficiente suerte como para tener un sistema de compilación continua, mira el apartado de "Cómo combinamos Scrum con XP").

La "pared de diseño" es sólo una pizarra grande que contiene los garabateos más importantes sobre el diseño y la documentación impresa más importante

(diagramas secuenciales, prototipos de la interfaz de usuario, modelos de dominio, etc.).



Foto: un Scrum diario tiene lugar en la anteriormente mencionada esquina de diseño.

Hmmm... Ese burn-down parece sospechosamente bonito y recto, ¿no? Pero el equipo insiste en que es real :o)

¡Sienta al equipo junto!

Cuando se trata de asientos y disposición de mesas, hay una cosa en la que nunca puede hacerse demasiado énfasis:

¡Sienta al equipo junto!

Para clarificar esto un poco, lo que estoy diciendo es:

¡Sienta al equipo junto!

A la gente le cuesta cambiar de sitio. Al menos en los sitios en los que he trabajado. No quieren tener que coger todas sus cosas, desenchufar el ordenador, mover todos los trastos a una nueva mesa y volver a enchufarlo todo otra vez. Cuanto menor sea la distancia, más resistencia. “Venga YA, Jefe, ¿Qué necesidad hay de mudarse a sólo cinco metros?”.

Cuando se trata de construir equipos Scrum efectivos, no obstante, no hay alternativa. Incluso si tienes que amenazar personalmente a cada individuo, llevar sus cosas y limpiar sus viejas manchas de café. Si no hay espacio para el equipo, haz espacio. Donde sea. Incluso si tienes que colocar al equipo en el sótano. Mueve las mesas, soborna al encargado de la oficina, haz lo que sea necesario. Pero pon al equipo junto.

Una vez tengas al equipo junto, los resultados serán inmediatos. Tras un solo Sprint el equipo estará de acuerdo en que fue una buena idea sentarse juntos (hablo desde mi experiencia personal, no hay nada que garantice que tu equipo no será tan testarudo como para no querer admitirlo).

Ahora bien, ¿Qué significa "juntos"? ¿Cómo deberían distribuirse las mesas? Bueno, no tengo ninguna opinión sólida sobre la distribución óptima de las mesas. E incluso si la tuviera, asumo que la mayoría de los equipos no cuentan con el lujo de se les permita decidir exactamente cómo quieren que se coloquen sus mesas. Usualmente hay limitaciones físicas – el equipo vecino, la puerta del baño, la gran máquina tragaperras en medio de la habitación, lo que sea.

"Juntos" significa:

- **Audibilidad:** cualquier miembro del equipo puede hablar con cualquier otro sin tener que levantarse de su sitio.
- **Visibilidad:** todo el mundo puede ver a los demás. Todo el mundo puede ver el tablón de tareas. No necesariamente tan de cerca como para poder leerlo, pero si por lo menos verlo.
- **Aislamiento:** si todo el equipo necesitase levantarse y agruparse para una animada y espontánea discusión sobre el diseño, nadie fuera del equipo está tan cerca como para ser molestado. Y viceversa.

"Aislamiento" no significa que el equipo tenga que estar completamente aislado. En un entorno cubicular, puede ser suficiente que el equipo tenga su propio cubículo y paredes suficientemente gruesas como para filtrar la mayoría del ruido de los elementos fuera del equipo.

¿Y que pasa si tienes un equipo distribuido? Bueno, entonces mala suerte. Usa tantas técnicas como sea posible para minimizar el daño – videoconferencia, webcams, herramientas de compartición de escritorio, etc.

Mantén al Dueño de Producto a mano

El Dueño de Producto debería estar suficientemente cerca como para que el equipo pueda caminar hasta donde está y preguntarle algo, y para que él pueda acercarse al tablón de tareas. Pero no debería sentársele junto al equipo. ¿Por qué? Porque lo más probable es que no sea capaz de controlarse e interrumpa constantemente entrometiéndose en cualquier detalle, y el equipo no podrá "fluir" adecuadamente (es decir, alcanzar un estado centrado, autogestionado e hiper-productivo).

Para ser honesto, esto es una especulación. Nunca he visto un caso en que el Dueño de Producto estuviera sentado con el equipo, por lo que no tengo razones empíricas para asegurar que sea una mala idea. Sólo una impresión personal y lo que comentan otros Scrum Masters.

Mantén a los gerentes y coaches a mano

Esto es duro de escribir para mí, ya que he sido tanto gerente como coach...

Mi trabajo consistía en trabajar tan próximo al equipo como fuera posible. Yo formaba los equipos, me movía entre ellos, programaba por parejas con sus miembros, formaba a otros Scrum Masters, organizaba reuniones de Planificación de Sprint, etc. Mirando hacia atrás, la mayoría de la gente pensaba que era algo bueno, ya que yo tenía algo de experiencia en desarrollo Ágil de software.

Pero, además, yo también era (música de Darth Vader) el Jefe de Desarrollo, un perfil funcional gerencial. Lo que significa que al entrar en un equipo este se convertía automáticamente en menos auto-gestionado. “Leches, el Jefe está aquí, seguro que tiene un montón de opiniones sobre lo que tendríamos que estar haciendo y quién tendría que estar haciendo qué. Dejaré que sea él el que hable”.

Mi punto de vista es: Si eres un coach en Scrum (y quizás también un gerente), involúcrate lo máximo posible. Pero sólo por un periodo de tiempo limitado. Entonces quítate de en medio y deja que sea el equipo el que se autogestione. Chequea el equipo de vez en cuando (no demasiado a menudo) atendiendo a las demos de Sprint, mirando el tablón de tareas y escuchando en los Scrum diarios.

Si ves un área de mejora, reúnete con el Scrum Master y ayúdale. No en frente de todo el equipo. Otra Buena idea es atender a las retrospectivas de los Sprints (ver “cómo hacemos retrospectivas de Sprint”), si tu equipo confía suficiente en ti como para que tu presencia les haga cerrarse en banda.

En lo tocante a los equipos Scrum que funcionen bien, asegúrate de que tengan todo lo que necesiten y quítate de su puñetero camino (excepto durante las demos de Sprint).

8

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

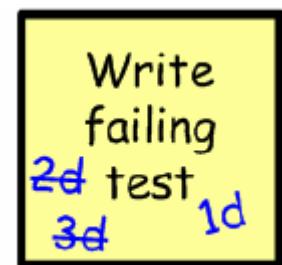
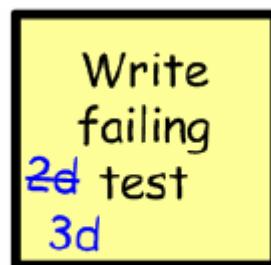
Cómo hacemos Scrum diarios

Nuestros Scrum diarios vienen a ser como describen las reglas. Empiezan exactamente a su hora, cada día en el mismo sitio. Al principio íbamos a alguna sala aparte para la planificación de Sprints (era cuando usábamos Pilas de Sprint electrónicas), pero en cualquier caso ahora hacemos Scrum diarios frente al tablón de tareas. No hay nada mejor que eso.

Normalmente hacemos las reuniones de pie, ya que eso reduce el riesgo de sobrepasar los 15 minutos.

Cómo actualizamos el tablón

Normalmente actualizamos el tablón de tareas durante los Scrum diarios. Conforme cada persona describe lo que hizo el día anterior y lo que hará hoy, mueve los post-it en el tablón. Conforme describe elementos no planificados, pone un pos-it nuevo para cada uno de ellos. Conforme actualiza sus estimaciones, escribe una nueva estimación en el post-it correspondiente y tacha la anterior estimación. A veces el Scrum Master hace todo esto mientras los demás hablan.



Algunos equipos tienen la política de que cada persona debe hacer la actualización del tablón que le corresponda antes del cada reunión. Eso también funciona bien. Simplemente decide qué política utilizar y cíñete a ella.

Sea cual sea el formato de tu Pila de Sprint, intenta involucrar a todo el equipo en la labor de mantener la Pila de Sprint actualizada. Hemos intentado hacer Sprints en los que el Scrum Master es el único que mantiene la Pila de Sprint y debe

hablar con todo el mundo todos los días y preguntarles por sus estimaciones de trabajo pendiente. Las desventajas de este enfoque son:

- El Scrum Master pasa demasiado tiempo administrando asuntos, en vez de dar soporte al equipo y eliminando impedimentos.
- Los miembros del equipo no están al corriente del estado del Sprint, ya que la Pila del Sprint es algo de lo que no necesitan preocuparse. Esta falta de feedback reduce la agilidad y el enfoque generales del equipo. Si la Pila de Sprint está bien diseñada debería ser igual de fácil para cada miembro del equipo actualizarla el mismo.

Inmediatamente tras el Scrum diario, alguien suma todas las estimaciones de tiempo (ignorando los de la columna “terminado”, por supuesto) y dibuja un nuevo punto en el burn-down de Sprint.

Tratando con tardones

Algunos equipos tienen una lata de monedas y billetes. Cuando llegas tarde, incluso aunque sea sólo por un minuto, añades una cantidad prefijada en la lata. Sin preguntas. Si llamas antes de la reunión y avisas de que vas a llegar tarde, aun así tienes que pagar. Solo te salvas de la multa si tienes una buena excusa como una cita con el médico, tu propia boda o algo similar.

El dinero de la lata puede usarse para eventos sociales. Para comprar hamburguesas cuando tenéis noche de juegos, por ejemplo :o)

Esto funciona bien. Pero solo es necesario en equipos donde es frecuente que la gente llegue tarde. Algunos equipos ni siquiera necesitan un esquema similar.

Tratando con “no se qué hacer hoy”

Ocurre a veces que alguien dice “Ayer hice bla bla bla, pero hoy no tengo la más remota idea de lo que hacer o lo que emprender” (hey, eso último ha rimado).

¿Y ahora qué?

Digamos que Joe y Lisa son los que no saben qué hacer hoy. Si yo soy el Scrum Master simplemente sigo con la reunión y dejo que el siguiente miembro del equipo hable, pero anoto quienes son las personas que no saben qué hacer. Después de que todo el mundo haya hablado, repaso el tablón de tareas con todo el equipo, de arriba a abajo, y me aseguro de que todo esté sincronizado, que todo el mundo sabe lo que significa cada elemento, etc. Invito a todo el mundo a añadir más post-its. Entonces vuelvo a las personas que no sabían qué hacer y les digo “ahora que hemos repasado el tablón de tareas, ¿tenéis idea de lo que podéis hacer hoy?”. Con algo de suerte será así.

Si no, considera si no tienes aquí una buena oportunidad para hacer programación por parejas. Digamos que Niklas va a implementar la interfaz de usuario de administración de usuarios del back-office hoy. En ese caso, sugiero educadamente si quizás Joe o Lisa podrían emparejarse con Niklas para ello. Eso suele funcionar.

Si eso tampoco funciona, este es el siguiente truco:

Scrum Master: "OK, ¿quién quiere demostrar la versión beta a todo el mundo?" (suponiendo que ese sea el objetivo del Sprint)

Equipo: (silencio confuso)

Scrum Master: "¿No hemos terminado?"

Equipo: "Um... No"

Scrum Master: "Oh, vaya, ¿Por qué no? ¿Qué falta por hacer?"

Equipo: "Bueno, ni siquiera tenemos un servidor de pruebas en el que ejecutarla, y el script de compilación no funciona"

Scrum Master: "Aham." (Añade dos post-it más a la pared de las tareas). "Joe y Lisa, ¿Cómo podéis ayudarnos hoy?"

Joe: "Um... Supongo que trataré de encontrar algún servidor de pruebas en algún sitio"

Lisa: "...Y yo trataré de arreglar ese script de compilación".

Si tienes suerte, alguien intentará demostrar la versión beta que pediste. ¡Estupendo! Habéis logrado vuestro objetivo de Sprint. Pero, ¿qué pasa si estas a mitad de Sprint? Fácil. Da la enhorabuena al equipo por el buen trabajo que han realizado, agarra una o dos historias de la sección "siguientes" del tablón y colócalas en la columna de "pendiente" a la izquierda. Entonces haz de nuevo el Scrum diario. Notifica al Dueño de Producto que habéis añadido algunos elementos más al Sprint.

Puede ocurrir que el equipo no haya alcanzado aun el objetivo del Sprint y a pesar de ello Joe y Lisa sigan negándose a especificar algo útil en lo que vayan a trabajar. Normalmente considero alguna de las siguientes estrategias. Ninguna de ellas es especialmente agradable, pero ten en cuenta que se trata de un último recurso:

- **Vergüenza:** "Bueno, si no tenéis idea de cómo podéis ayudar al equipo os sugiero que os vayáis a casa, o leáis un libro o algo. O simplemente sentaros por aquí hasta que alguien del equipo os pida ayuda"
- **Vieja escuela:** Simplemente asignales una tarea.
- **Presión de los compañeros:** Di "sentiros libres de tomaros el tiempo que haga falta, Joe y Lisa, todos los demás simplemente nos quedaremos aquí de pie hasta que se os ocurra algo que pueda ayudarnos a conseguir el objetivo del Sprint".
- **Servidumbre:** Di algo como "Bueno, podéis ayudar al equipo indirectamente siendo sus mayordomos hoy. Traedles café, dadles un masaje, limpiad sus mesas, preparadles algo de comida y cualquier cosa que os pidan durante el día". Os sorprenderá lo rápido que Joe y Lisa encontrarán algunas tareas técnicas de provecho :o)

Si una persona te fuerza frecuentemente a ir tan lejos, probablemente deberías coger a esa persona en un aparte y hacer algo de coaching en serio. Si el problema continua, necesitas evaluar si esa persona es importante para el equipo o no.

Si no es demasiado importante, intenta que lo aparten de tu equipo. Si es demasiado importante, entonces intenta emparejarlo con alguien que pueda actuar como su "guía". Joe puede ser un desarrollador y arquitecto estupendo, pero simplemente puede preferir que sean otros los que le digan lo que tiene que hacer. Dale a Niklas la tarea de ser su guía permanente. O hazte cargo tú mismo de esa labor. Si Joe es suficientemente importante para el equipo, merecerá la pena el esfuerzo. Hemos tenido casos como este y más o menos ha funcionado.

9

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo hacemos la demo de Sprint

La demo de Sprint (o revisión de Sprint, como la llaman algunos) es una parte importante de Scrum que la gente tiende a subestimar.

“Oh, ¿realmente tenemos que hacer una demo? ¡En realidad no hay mucho que podamos enseñar!”

“¡No tenemos tiempo para montar una &%\$# demo!”

“¡No tengo tiempo para asistir a las demo de los demás equipos!”

Por qué insistimos en que todos los Sprints acaben con una demo

Una demo de Sprint bien ejecutada, aunque parezca poco espectacular, tiene un efecto muy profundo:

- El equipo obtiene reconocimiento por sus logros. Se sienten bien.
- Otras personas se enteran de lo que está haciendo el equipo.
- La demo consigue feedback vital de los interesados.
- Las demos son (o deberían ser) un evento social donde diferentes equipos pueden interactuar unos con otros y discutir su trabajo. Esto es muy valioso.
- Hacer una demo fuerza al equipo a acabar realmente las cosas y entregarlas (incluso aunque sea sólo en entorno de pruebas). Sin las demos, seguimos consiguiendo enormes montones de cosas terminadas al 99%. Con las demos puede que consigamos menos cosas terminadas, pero estas están realmente terminadas, lo que (en nuestro caso) es mucho mejor que tener una enorme pila de cosas que están más o menos listas y que se pulirán en el próximo Sprint.
- Si un equipo se ve obligado a hacer una demo de Sprint, incluso aunque no tengan mucho que realmente esté funcionando, la demo será embarazosa. El equipo tartamudeará y tropezará mientras hace la demo y el aplauso de después será frío. La gente sentirá un poco de pena por el equipo, algunos se enfadarán con ellos por hacerles perder el tiempo con una demo pésima. Esto duele, pero el efecto es similar al de una

amarga medicina. En el próximo Sprint, el equipo intentará por todos los medios tener cosas terminadas. Sentirán algo como “bueno, puede que sólo podamos demostrar dos historias el próximo Sprint en vez de cinco, pero maldita sea, ¡esta vez VA A FUNCIONAR!”. El equipo sabe que tendrán que hacer una demo pase lo que pase, lo que incrementa significativamente las posibilidades de que haya algo útil que demostrar. He visto como esto ocurría varias veces.

Lista de comprobación para demos de Sprint

- Asegúrate de presentar claramente el objetivo del Sprint. Si hay personas en la demo que no saben nada sobre tu producto, tomate un par de minutos para describirlo.
- No pierdas mucho tiempo preparando la demo, especialmente en llamativas presentaciones. Déjate de milongas y concéntrate mostrar código funcionando.
- Mantén el paso rápido, es decir, concentra tu preparación en hacer que la demo sea rápida en lugar de bonita.
- Mantén la demo a nivel de negocio, deja los detalles técnicos aparte. Concéntrate en “qué hemos hecho” en lugar de “cómo lo hemos hecho”.
- En la medida de lo posible, deja que la audiencia pruebe el producto por si misma.
- No muestres un montón de pequeños errores solucionados y funcionalidades triviales. Mencionalos, pero no los muestres, ya que normalmente se tarda mucho y desvía la atención de las historias más importantes.

Tratando con historias “indemostrables”

Miembro del equipo: “No voy a demostrar esta historia porque no puede demostrarse. La historia es ‘mejorar la escalabilidad de forma que el sistema pueda aguantar 10.000 usuarios simultáneos’. A ver como leches invito a 10.000 usuarios simultáneos a la demo.”

Scrum Master: “¿Has terminado la historia?”

Miembro del equipo: “Sí, por supuesto”

Scrum Master: “¿Cómo lo sabes?”

Miembro del equipo: “Monté el sistema en un entorno de pruebas de rendimiento, arranqué ocho servidores de carga y le di caña al sistema con solicitudes simultáneas”.

Scrum Master: “Pero no tienes ninguna indicación de que el sistema pueda aguantar 10.000 usuarios simultáneos”

Miembro del equipo: “Sí. Las máquinas de pruebas están echas polvo, pero aun así pudieron aguantar 50.000 solicitudes simultáneas durante el test”.

Scrum Master: “¿Cómo lo sabes?”

Miembro del equipo: (frustrado) “¡Bueno, tengo este informe! ¡Puedes verlo por ti mismo, muestra cómo se montó la prueba y cuántas solicitudes se enviaron!”

Scrum Master: “¡Oh, excelente! Entonces ahí tienes tu ‘demo’. Simplemente muestra el informe y coméntaselo a la audiencia. Mejor que nada, ¿no?”

Miembro del equipo: “Ah, ¿basta con eso? Pero está muy feo, necesito ponerlo en bonito y...”

Scrum Master: “Vale, pero no pierdas mucho tiempo. No se trata de que sea bonito, simplemente informativo”.



10

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo hacemos retrospectivas de Sprint

Por qué insistimos en que todos los equipos hagan retrospectivas

Lo más importante de las retrospectivas es asegurarse de que tienen lugar.

Por alguna razón, los equipos no siempre parecen inclinados a hacer retrospectivas. Sin un empujoncito, la mayoría de nuestros equipos usualmente se saltarían la retrospectiva y continuarían con el próximo Sprint. Puede que sea algo cultural de los suecos, no estoy seguro.

Aun así, todo el mundo coincide en que las retrospectivas son extremadamente útiles. De hecho, yo diría que la retrospectiva es el segundo evento más importante de Scrum (siendo el primero la reunión de planificación de Sprint) ya que ¡es tu mejor oportunidad para mejorar!

Por supuesto, no necesitas una reunión de retrospectiva para conseguir buenas ideas, podrías tenerlas en la bañera de tu casa. Pero ¿aceptará el equipo tus ideas? Quizás, pero la probabilidad de conseguir que el equipo las acepte es drásticamente mayor si la idea viene "del equipo", es decir, surge durante la retrospectiva y a todo el mundo se le permite contribuir y discutir las ideas.

Sin las retrospectivas encontrarás que el equipo sigue cometiendo los mismos errores una y otra vez.

Cómo organizamos las retrospectivas

El formato general varía un poco, pero normalmente hacemos algo como esto:

- Reservamos 1-3 horas, dependiendo de cuánta discusión esperemos.
- Participantes: el Dueño de Producto, el equipo y yo mismo. Scrum master
- Nos vamos a una reunión cerrada, un rincón cómodo con sofás, el patio del tejado o algún sitio similar. Que podamos tener una discusión sin interrupciones.

- Normalmente no hacemos retrospectivas en la sala del equipo, ya que la atención de la gente suele diluirse.
- Alguien es designado secretario.
- El Scrum Master muestra la Pila de Sprint y, con ayuda del equipo, resume el Sprint, Eventos importantes, decisiones, etc.
- Hacemos "la ronda". Cada persona tiene una oportunidad de decir, sin ser interrumpida, qué piensan que ha ido bien, qué podría haber ido mejor y qué piensan que debería hacerse de forma diferente en el próximo Sprint.
- Observamos la velocidad estimada frente a la real. Si hay una gran diferencia, intentamos analizar por qué.
- Cuando el tiempo casi se ha acabado, el Scrum Master trata de resumir las sugerencias concretas sobre qué puede hacerse mejor el próximo Sprint.

Nuestras retrospectivas generalmente no están muy estructuradas. No obstante, el tema subyacente es siempre el mismo: "qué podemos hacer mejor el próximo Sprint".

He aquí un ejemplo de pizarra de una retrospectiva reciente:



Tres columnas:

- **Bien:** si hiciéramos el Sprint otra vez, volveríamos a hacer estas cosas igual.
- **Mejorable:** si hiciéramos otra vez el Sprint, haríamos estas cosas de forma diferente.
- **Mejoras:** ideas concretas sobre cómo podemos mejorar en el futuro.

Así que las columnas 1 y 2 son una mirada al pasado, mientras que la columna 3 mira al futuro.

Después de que el equipo genere todas estas ideas en post-its, utilizan "votación por puntos" para determinar en qué mejoras centrarse el próximo Sprint. Cada miembro del equipo tiene tres imanes y se les invita a votar sobre cualquier mejora en la que les gustaría trabajar en el próximo Sprint. Cada miembro del

equipo distribuye los imanes como quiera, incluso colocando los tres en el mismo elemento.

Basándonos en esto, seleccionamos 5 mejoras de procesos en los que concentrarse, y los evaluamos en la siguiente retrospectiva.

Es importante no ser demasiado ambicioso. Concéntrate en unas pocas mejoras en cada Sprint.

Difundiendo las lecciones entre los equipos

La información que surge durante una retrospectiva de Sprint es casi siempre tremadamente valiosa. ¿Tiene este equipo problemas de concentración porque los gerentes de ventas insisten en secuestrar programadores para participar como "expertos técnicos" en reuniones de ventas? Esta es una información importante. Quizás otros equipos tengan los mismos problemas. ¿Deberíamos estar formando más a los responsables de producto acerca de nuestros productos, de forma que puedan hacer el soporte a ventas ellos mismos?

Una retrospectiva de Sprint no trata sólo de cómo este equipo puede hacerlo mejor el próximo Sprint, tiene implicaciones más amplias que esa.

Nuestra estrategia para manejar este hecho es muy simple. Una persona (en este caso, yo) atiende a todas las reuniones de retrospectiva y actúa como puente de conocimiento. Muy informal.

Una alternativa sería que cada equipo Scrum publique un informe de la retrospectiva de Sprint. Lo hemos intentado, pero encontramos que no mucha gente lee esos informes, y muchos menos actúan basándose en ellos. Así que lo hacemos de la forma simple.

Reglas importantes para la persona que actúa como "puente de conocimiento":

- Debería ser bueno escuchando.
- Si la retrospectiva es poco activa, debería estar listo para realizar preguntas simples pero bien apuntadas para estimular la discusión dentro del grupo. Por ejemplo, "si pudierais rebobinar y hacer este Sprint otra vez desde el día 1, ¿qué haríais de forma diferente?"
- Debe estar dispuesto a pasar tiempo visitando todas las retrospectivas de todos los equipos.
- Debería tener algún tipo de autoridad, de forma que pueda actuar sobre las sugerencias que estén fuera del control del propio equipo.

Esto funciona bastante bien, pero puede que haya otras aproximaciones que funcionen mejor. En ese caso, por favor, iluminadme.

Cambiar o no cambiar

Digamos que el equipo concluye que "nos hemos comunicado muy poco entre los miembros del equipo, así que hemos estado pisándonos unos a otros y estropeando los diseños de los demás".

¿Qué deberías hacer al respecto? ¿Establecer reuniones diarias de diseño? ¿Implantar nuevas herramientas que faciliten la comunicación? ¿Añadir más páginas al Wiki? Bueno, quizás. Pero quizás no.

Hemos descubierto que, en muchos casos, simplemente identificar el problema es suficiente para que se resuelva por si mismo automáticamente en el próximo Sprint. Especialmente si pegas la retrospectiva del Sprint en la pared de la sala del equipo (cosa que nosotros siempre olvidamos hacer, ¡vergüenza debería darnos!). Cada cambio que introduces tiene algún tipo de coste, así que antes de hacer nuevos cambios considera no hacer nada y esperar a que el problema desaparezca (o se haga más pequeño) automáticamente.

El ejemplo anterior (“nos hemos comunicado poco...”) es un ejemplo típico de algo que se puede resolver mejor simplemente no haciendo nada.

Si introduces un nuevo cambio cada vez que alguien se queja de algo, la gente se resistirá a revelar pequeñas áreas problemáticas, lo cual sería terrible.

Ejemplo de cosas que pueden surgir en las retrospectivas

He aquí unos cuantos ejemplos de típicas cosas que surgen durante la retrospectiva y las típicas acciones a llevar a cabo:

“Deberíamos haber pasado más tiempo dividiendo historias en subhistorias y tareas”

Esta es muy habitual. Cada día en el Scrum diario, los miembros del equipo se encuentran a sí mismos diciendo “Realmente no sé qué hacer hoy”. Así que después de cada Scrum diario tienes que pasar un tiempo encontrando tareas concretas. Es generalmente más efectivo hacer este trabajo desde el principio.

Acciones típicas: ninguna. El equipo probablemente corregirá esto por si mismo en el próximo Sprint. Si esto ocurre repetidas veces, incrementa el tiempo para la planificación de Sprint.

sprint planning

“Demasiadas distracciones”

Acciones típicas:

- Pide al equipo que reduzca su factor de dedicación el próximo Sprint de forma que tengan una planificación más realista.
- Pide al equipo que lleven un registro de las interrupciones el próximo Sprint. Quién interrumpe, cuánto tiempo. Así será más fácil resolver el problema más adelante.
- Pide al equipo que canalicen todas las distracciones a través del Scrum Master o el Dueño de Producto.

- Pide al equipo que designe a una persona como "portero", y que todas las interrupciones se le envíen a él, de forma que el resto del equipo pueda concentrarse. Podría ser el Scrum Master o una posición rotatoria.

"Nos sobre-comprometimos y sólo hicimos la mitad"

Acciones típicas: ninguna. El equipo probablemente no se sobre-comprometerá en el próximo Sprint. O al menos no se sobre-comprometerá tanto.

"Nuestro entorno de oficina es demasiado ruidoso y desordenado"

Acciones típicas:

- Intenta crear un ambiente mejor, o llévate al equipo fuera de la oficina. Alquila una habitación de hotel. Lo que haga falta. Mira "cómo distribuimos la sala del equipo".
- Si no es posible, dile al equipo que reduzca su factor de concentración el próximo Sprint, y que declaren claramente que esto es así debido a lo ruidoso y desordenado del ambiente en el que trabajan. Con suerte esto causará que el Dueño de Producto comience a hostigar a la alta dirección sobre este tema.

Afortunadamente nunca he tenido que amenazar con llevarme al equipo fuera de la oficina. Pero lo haré si tengo que hacerlo :o)

11

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Entra

Descansos entre Sprints

En la vida real no puedes estar siempre esprintando. Si siempre estás esprintando, en realidad acabarás haciendo *footing*.

Lo mismo ocurre en Scrum y en el desarrollo de software en general. Los Sprints son muy intensos. Como desarrollador nunca puedes relajarte, todos los días tienes que estar de pie en esa puñetera reunión y decirle a todo el mundo lo que conseguiste ayer. Muy pocos dirán algo como “pasé la mayor parte del día con los pies sobre la mesa, leyendo blogs y sorbiendo un capuchino”.

Además del propio descanso, hay otra buena razón para dejar algo de espacio entre Sprints. Después de la demo y la retrospectiva tanto el Dueño de Producto como el Equipo estarán llenos de información y de ideas por digerir. Si se ponen de inmediato a planificar el próximo Sprint, es probable que nadie tenga la oportunidad de digerir ninguna información o lección aprendida, el Dueño de Producto no tendrá tiempo de ajustar sus prioridades después de la demo, etc.

Mal:

Lunes
9-10 Demo Sprint 1
10-11 Retrospectiva
13-16 Planificación Sprint 2

Intentamos introducir algo de descanso antes de empezar un nuevo Sprint (más específicamente, en el periodo después de terminar la retrospectiva de Sprint y antes de la siguiente reunión de planificación de Sprint). No siempre lo conseguimos, la verdad.

Como mínimo, intentamos que la retrospectiva y la subsiguiente reunión de planificación de Sprint no ocurran el mismo día. Todo el mundo debería tener al menos una buena noche de sueño sin Sprint antes de comenzar el siguiente Sprint.

Mejor:

Lunes	Martes
9-10 Demo Sprint 1 10-11 Retrospectiva	9-113 planificación Sprint 2

Mucho mejor:

Viernes	Sábado	Domingo	Lunes
9-10 Demo Sprint 1 10-11 Retrospectiva			9-113 planificación Sprint 2

Una forma de hacer esto son los “días de laboratorio” (o como quieras llamarlos). Es decir, días en los que a los desarrolladores se les permite hacer esencialmente cualquier cosa que deseen (OK, lo admito, está inspirado en Google). Por ejemplo, leer sobre las últimas herramientas y API's, estudiar para una certificación, discutir asuntos técnicos con colegas, programar un proyecto personal como hobby, etc.

Nuestro objetivo es tener un día de laboratorio entre cada Sprint. De esta forma obtenemos un descanso natural entre Sprints, y tendrás un equipo de desarrollo que tendrá una oportunidad realista de mantener su conocimiento al día. Además, es un estupendo beneficio laboral.

¿El mejor?

Jueves	Viernes	Sábado	Domingo	Lunes
9-10 Demo Sprint 1 10-11 Retrospectiva	LABORATORIO			9-113 planificación Sprint 2

Actualmente tenemos días de laboratorio una vez al mes. El primer viernes de cada mes, para ser más específicos. ¿Por qué no entre Sprints? Bueno, porque sentía que era importante que toda la compañía se tomase el día de laboratorio a la vez. De otra forma la gente tiende a no tomárselo en serio. Y dado que (hasta ahora) no hemos sincronizado los Sprints de todos los productos, tuve que seleccionar un día de laboratorio no dependiente de los Sprints.

Puede que algún día intentemos sincronizar los Sprints de todos los productos (es decir, misma fecha de inicio y fin de Sprint para todos los productos y equipos). En ese caso, definitivamente colocaremos un día de laboratorio entre Sprints.

12

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Como hacemos planificación de entregas y contratos a precio cerrado

A veces necesitamos planificar por adelantado más de un Sprint a la vez. Típicamente cuando tenemos un contrato a precio cerrado en el que tenemos que planificar por adelantado, o correr el riesgo de firmar algo que no podemos entregar a tiempo.

Típicamente, la planificación de entregas es para nosotros un intento de resolver la pregunta “cuándo, como muy tarde, seremos capaces de entregar la versión 1.0 de este nuevo sistema”.

Si realmente quieres aprender sobre planificación de entregas te sugiero que te saltes este capítulo y en su lugar te compres el libro de Mike Cohn “Agile Estimating and Planning”. Ojala hubiera leído ese libro antes (lo leí después de que nos figurásemos todo este asunto por nuestra cuenta). Mi versión de planificación de entregas es un poco simplista, pero puede servir como un punto de partida.

Define tus umbrales de aceptación

Además de la Pila de Producto habitual, el Dueño de Producto define una lista de umbrales de aceptación, que son una simple clasificación de qué significan los niveles de importancia de la Pila de Producto en términos del contrato.

He aquí un ejemplo de umbrales de aceptación:

- Todos los elementos con importancia ≥ 100 deben estar incluidos en la versión 1.0, o seremos penalizados hasta la muerte.
- Todos los elementos de importancia 50-99 deberían estar incluidos en la versión 1.0, pero podríamos pasar sin ellos si los incluyésemos en otra entrega poco después.
- Los elementos con importancias 25-49 son requisitos, pero podemos incluirlos en una versión 1.1.
- Los elementos con importancia <25 son puramente especulativos y puede que ni siquiera hagan falta.

Y he aquí un ejemplo de Pila de Producto con un código de colores basado en las reglas anteriores:

Importancia	Historia
130	Plátano
120	Manzana
115	Naranja
110	Guayaba
100	Pera
95	Pasa
80	Cacahuete
70	Donut
60	Cebolla
40	Uva
35	Papaya
10	Arándano
10	Melocotón

Rojo = debe incluirse en la versión 1.0 (plátano – pera)

Amarillo = debería incluirse en la versión 1.0 (pasa – cebolla)

Verde = puede hacerse más tarde (uva – melocotón)

Así que si lo entregamos todo desde plátano a cebolla en la fecha límite, estamos a salvo. Si el tiempo se nos acaba, podríamos salir adelante abandonando pasa, cacahuete, donut o cebolla. Cualquier cosa por debajo de cebolla es un plus.

Estimación de los elementos más importantes

Para poder hacer la planificación de entregas el dueño de producto necesita estimaciones, al menos para todas las historias incluidas en el contrato. Al igual que en la planificación de Sprint, se trata de un esfuerzo cooperativo entre el Dueño de Producto y el equipo – el equipo estima, el Dueño de Producto describe los elementos y responde a las preguntas.

Una estimación de tiempos es valiosa si resulta ser casi correcta, menos valiosa si resulta que falla por, digamos, un 30% y completamente inútil si no tiene ninguna conexión con la realidad.

He aquí mi percepción del valor de una estimación de tiempos en función de quién la calcula y cuánto tiempo invierten realizando la estimación:



Todo esto no ha sido más que una forma complicada de decir:

- Deja que el **equipo** haga las estimaciones.
- **No** dejes que le **dediquen demasiado tiempo**.
- Asegúrate de que entiendan que **se trata de estimaciones, no compromisos**.

Usualmente el **Dueño de Producto** **reúne** a todo **el equipo** en una sala, proporciona algunos refrescos y les dice que el **objetivo de la reunión es estimar las principales 20** (o las que sean) **historias de la Pila de Producto**. Enuncia cada historia una vez y deja que el equipo se ponga manos a la obra. El **Dueño de Producto** **se queda en la sala para responder preguntas y aclarar el alcance de cada historia si es necesario**. Igual que en la planificación de Sprint, el campo **“como probarlo”** es muy útil para reducir el riesgo de malentendidos.

Esta **reunión** debe de ser **limitada en tiempo**, o si no el equipo tiende a perder demasiado tiempo estimando muy pocas historias.

Si el **Dueño de Producto quiere** que se **dedique más tiempo** simplemente **organiza otra reunión para más adelante**. El **equipo debe asegurarse de que el impacto de estas reuniones** en sus **actuales Sprints** **sea claramente visible** para el **Dueño de Producto**, de forma que entienda que su **trabajo de estimación no es gratuito**.

He aquí un ejemplo de cómo podrían acabar las estimaciones (en puntos de historia):

Importancia	Historia	Estimación
130	Plátano	12
120	Manzana	9
115	Naranja	20
110	Guayaba	8
100	Pera	20
95	Pasa	12
80	Cacahuete	10
70	Donut	8
60	Cebolla	10
40	Uva	14
35	Papaya	4
10	Arándano	
10	Melocotón	

Estimar la velocidad

OK, así que ahora tenemos algunas **estimaciones rudimentarias** para las **historias más importantes**.

El **siguiente paso es estimar nuestra velocidad media por Sprint**

Esto significa que debemos decidir nuestro factor de dedicación. Lee "Como decide el equipo qué historias incluir en el Sprint".

El factor de dedicación significa, básicamente, "cuanto del tiempo del equipo se emplea en las historias a las que se ha comprometido". Nunca es el 100%, ya que el equipo gasta tiempo en elementos no planificados, haciendo cambios de contexto, ayudando a otros equipos, chequeando el correo, arreglando sus ordenadores rotos, discutiendo de política en la cocina, etc.

Digamos que determinamos que el factor de dedicación del equipo es del 50% (bastante bajo, normalmente oscilamos en torno al 70%). Y digamos que la duración del Sprint es de 3 semanas (15 días) y el tamaño del equipo es 6 personas.

Así que cada Sprint tendría 90 días-hombre ideales, pero solo podemos pretender producir el equivalente a 45 días-hombre ideales (debido al factor del 50%).

Así que nuestra velocidad estimada es 45 puntos de historia.

Si cada historia tiene una estimación de tiempo de 5 días (que no es así), entonces este equipo podría producir aproximadamente 9 historias por Sprint.

Uniéndolo todo en un plan de entregas (*release plan*)

Ahora que tenemos estimaciones de tiempo y una velocidad (45) podemos dividir fácilmente la Pila de Producto en Sprints.

Importancia	Historia	Estimación
Sprint 1		
130	Plátano	12
120	Manzana	9
115	Naranja	20
Sprint 2		
110	Guayaba	8
100	Pera	20
95	Pasa	12
Sprint 3		
80	Cacahuete	10
70	Donut	8
60	Cebolla	10
40	Uva	14
Sprint 4		
35	Papaya	4
10	Arándano	
10	Melocotón	

Cada Sprint incluye tantas historias como sea posible sin exceder la velocidad estimada de 45.

Así, podemos ver que probablemente necesitamos 3 Sprints para finalizar todos los "debe" y los "debería".

3 Sprints = 9 semanas de calendario = 2 meses. Así que esa es nuestra fecha de entrega. Ahora bien, ¿es la fecha que se prometió al cliente? Depende enteramente de la naturaleza del contrato: como de fijo es el alcance, etc. Usualmente añadimos un buffer o colchón significativo para protegernos contra las malas estimaciones, problemas inesperados, etc. Así que en este caso podríamos acordar fijar la fecha de entrega dentro de 3 meses, dándonos así un mes de "reserva".

Lo bonito es que podemos mostrar algo usable al cliente cada tres semanas e invitarle a introducir cambios en los requisitos conforme avanzamos (dependiendo por supuesto de lo que permita el contrato).

Adaptando el plan de entregas

La realidad no se adaptará ella sola al plan, así que tendremos que hacerlo al revés.

Después de cada Sprint comprobamos la velocidad real de dicho Sprint. Si la velocidad real ha sido muy diferente de la estimada, revisamos la velocidad estimada para próximos Sprints y actualizamos el plan de entregas. Si esto nos coloca en una situación problemática, puede que el Dueño de Producto empiece a negociar con el cliente o se dedique a averiguar cómo podemos reducir el alcance sin romper el contrato. O quizás él y el equipo encuentren una forma de aumentar la velocidad eliminando algún impedimento severo que se haya identificado durante el Sprint.

El Dueño de Producto podría llamar al cliente y decirle "hola, vamos un poco retrasados respecto a la planificación, pero creo que podríamos cumplir con la fecha de entrega si eliminásemos la funcionalidad "comecocos embebido" que nos llevaría un montón de tiempo construir. Podríamos añadirla en la siguiente entrega, 3 semanas después de la primera versión, si así lo quieras".

Quizás no sean buenas noticias para el cliente, pero al menos estamos siendo honestos y le estamos dando al cliente opciones muy pronto: podemos entregar las funcionalidades más importantes en fecha o entregarlo todo pero tarde. Normalmente no suele ser una decisión muy difícil :o)

13

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo combinamos Scrum con XP

Decir que Scrum y XP (Programación eXtrema) pueden combinarse de forma fructífera no es una afirmación muy controvertida. La mayoría del material que encuentro en la red apoya esta hipótesis, así que no voy a perder mucho tiempo argumentando por qué.

Bueno, mencionaré una sola cosa. Scrum se enfoca en las prácticas de organización y gestión, mientras que XP se centra más en las prácticas de programación. Esa es la razón de que funcionen tan bien juntas: tratan de áreas diferentes y se complementan entre ellas.

¡Desde estas líneas, añado mi voz a los que afirman que existe evidencia empírica de que Scrum y XP se pueden combinar fructíferamente!

Voy a subrayar algunas de las prácticas de XP más valiosas y cómo se aplican a nuestro trabajo cotidiano. No todos nuestros equipos han logrado adoptar todas las prácticas, pero en conjunto hemos experimentado con casi todos los aspectos de la combinación Scrum/XP. Algunas de las prácticas de XP son tratadas directamente por Scrum y se podrían ver como una intersección entre ambas, como por ejemplo “Todo el equipo”, “Sentarse juntos”, “Historias” y “Juego de Planificación”. En esos casos simplemente nos hemos ceñido a Scrum.

Pair Programming

Programación por parejas

Empezamos a hacer esto hace poco con uno de nuestros equipos. En realidad, funciona bastante bien. La mayoría de los demás equipos no hacen mucha programación por parejas, pero ahora que lo he probado en un equipo durante algunos Sprints estoy inspirado para animar a más equipos a probarlo.

Algunas conclusiones hasta ahora respecto a la programación por parejas:

- La programación por parejas realmente mejora la calidad del código.
- La programación por parejas realmente mejora la concentración del equipo (por ejemplo cuando el tipo que está sentado contigo dice “hey, ¿realmente necesitamos eso para este Sprint?”)
- Sorprendentemente, muchos de los desarrolladores que están en totalmente en contra de la programación por parejas ni siquiera la han probado, y aprenden rápidamente a apreciarla una vez que la prueban.

- La programación por parejas es agotadora y no debería hacerse durante todo el día.
- Es bueno cambiar de parejas frecuentemente.
- La programación por parejas realmente mejora la distribución de conocimiento entre el equipo. Sorprendentemente rápido, además.
- Algunas personas simplemente no se sienten a gusto haciendo programación por parejas. No prescindas de un programador excelente simplemente porque no se sienta a gusto programando por parejas.
- La revisión de código es una alternativa aceptable a la programación por parejas.
- El “navegante” (el tipo que no usa el teclado) debería tener un ordenador propio también. No para desarrollar, pero sí para hacer pequeñas tareas cuando sea necesario, consultar documentación cuando el “piloto” (el tipo que usa el teclado) se atasque, etc.
- No fuerces a la gente a hacer programación por parejas. Anima a las personas y proporcionales las herramientas adecuadas, pero permíteles experimentar con ella a su propio ritmo.

Desarrollo guiado por pruebas (TDD)

¡Amen! Esto, para mi, es más importante que Scrum y XP juntos. Puedes quitarme la casa, la tela y el perro, ¡pero no intentes impedirme hacer *Test Driven Development* (TDD)! Si no te gusta TDD, entonces es mejor que no me dejes entrar en el edificio, ya que trataré de colarlo de una forma u otra :o)

He aquí un resumen en diez segundos de TDD:

Desarrollo guiado por pruebas significa que escribes un test automático y, a continuación, escribes el código suficiente para pasar dicho test y después refactorizas el código, principalmente para mejorar la legibilidad y eliminar duplicaciones. Aclarar y repetir.

Algunas reflexiones sobre el desarrollo guiado por pruebas:

- TDD es duro. Los programadores tardan un tiempo en pillarlo. De hecho, en muchos casos no importa cuanto lo expliques, lo demuestres y los animes: en muchos casos la única forma de que un programador lo pille es emparejarlo con otro programador que sea bueno en TDD. Una vez que un programador lo pilla, sin embargo, normalmente será infectado severamente y nunca más querrá trabajar de otra forma.
- TDD tiene un efecto profundamente positivo en del diseño del sistema.
- Se tarda un tiempo en conseguir que TDD funciona en un nuevo producto, especialmente con pruebas de integración tipo “caja negra”, pero el retorno de la inversión
- Asegúrate de que inviertes suficiente tiempo en hacer que a la gente le resulte fácil escribir pruebas. Esto significa conseguir las herramientas adecuadas, educar a las personas, proporcionarles las clases de utilidad o clases básicas adecuadas, etc.

Nosotros empleamos las siguientes utilidades para el desarrollo guiado por pruebas:

- jUnit / httpUnit / jWebUnit. Estamos considerando TestNG y Selenium.
- HSQLDB como una base de datos embebida en memoria para pruebas.

- **Jetty** como un contenedor Web embebido en memoria para pruebas.
- Para las métricas de cobertura de pruebas usamos Cobertura.
- El framework **Spring** para montar diferentes tipos de instalaciones de pruebas (con simulacros, sin simulacros, con base de datos externa, con base de datos en memoria, etc.)

En nuestros productos más sofisticados (desde una perspectiva **TDD**) tenemos **pruebas de aceptación tipo “caja negra” automatizadas**. Estas pruebas **arrancan todo el sistema** en memoria, incluyendo las **bases de datos y servidores Web**, y acceden al sistema utilizando únicamente sus interfaces públicos (por ejemplo **HTTP**).

Esto hace extremadamente **rápidos los ciclos de desarrollo-compilación-pruebas**. También actúa como **una red de seguridad**, proporcionando a los desarrolladores suficiente confianza como **para refactorizar a menudo**, lo que significa que el **diseño** se mantiene **limpio y simple** incluso mientras el sistema crece.

TDD en código nuevo

Hacemos TDD para todos los nuevos desarrollos, incluso si significa que el arranque del proyecto tarde más tiempo (ya que necesitamos más herramientas y soporte para pruebas de seguridad, etc.). Esto no tiene vuelta de hoja, ya que los beneficios son tan grandes que realmente no hay excusas para *no hacer TDD*:

TDD en código antiguo

TDD es duro, pero intentar hacer TDD sobre una base de código que no se construyó utilizando TDD desde el principio... Eso *si que es duro*. ¿Por qué? Bueno, realmente podría escribir muchas páginas sobre el tema, así que creo que lo dejaré aquí. Lo guardaré para mi libro “TDD desde las trincheras” :o)

Pasamos bastante tiempo intentando automatizar las pruebas de integración en uno de nuestros sistemas más complejos, una base de código que habíamos empleado durante un tiempo y estaba en un estado severo de desorden y completamente desprovista de pruebas.

Para **cada entrega del sistema teníamos a un equipo dedicado de pruebas de dos personas** que realizarían un montón de complejísimas pruebas de regresión y rendimiento. Las **pruebas de regresión** eran **fundamentalmente manuales**. Esto ralentizaba significativamente nuestro ciclo de desarrollo y entrega. Nuestra meta era automatizar estas pruebas. Sin embargo, después de golpearnos la cabeza contra la pared durante unos meses, no habíamos progresado mucho.

Después de aquello cambiamos la aproximación al problema. Aceptamos el hecho de que estábamos condenados a hacer pruebas de regresión de forma manual, y empezamos a preguntarnos “**¿cómo podríamos hacer que las pruebas manuales no tardasen tanto tiempo?**”. Se trataba de un sistema de juegos, y nos dimos cuenta que gran parte del tiempo del equipo de pruebas se estaba consumiendo en tareas triviales de puesta en funcionamiento, como navegando por el interfaz de administración para configurar torneos de prueba, o esperando a que comenzase un torneo programado. Así que fuimos creando utilidades para

ello. Pequeños y fácilmente accesibles atajos que hacían todo el trabajo rutinario y permitían a los encargados de pruebas concentrarse en las pruebas.

¡El esfuerzo mereció la pena! De hecho, es probablemente lo que deberíamos haber hecho desde el principio. Estábamos tan deseosos de automatizar las pruebas que nos olvidamos de hacerlo paso a paso, siendo el primer paso construir herramientas que hiciesen las pruebas *manuales* más eficientes.

Lecciones aprendidas: si estás **atascado con pruebas de regresión manuales** y **quieres automatizarlas, no lo hagas** (a menos que sea realmente fácil). En lugar de eso, **construye herramientas que hagan la regresión manual más sencilla**. **Entonces considera automatizar las pruebas.**

Diseño incremental

Esto significa **mantener el diseño simple desde el principio** y **mejorarlo continuamente**, en lugar de conseguir que todo funcione desde el principio y entonces congelarlo.

Esto lo estamos haciendo bastante bien, es decir, empleamos una cantidad razonable de tiempo refactorizando y mejorando los diseños existentes, y rara vez empleamos tiempo haciendo grandes diseños desde el principio. A veces metemos la pata, por supuesto, por ejemplo permitiendo que un diseño inestable se “enquiste” fuertemente de tal forma que la refactorización se convierta en un proyecto grande. Pero en términos generales estamos satisfechos.

La **mejora continua del diseño** es sobre todo un efecto secundario **automática** de hacer TDD.

Integración continua

La mayoría de nuestros productos cuentan con un sofisticado **entorno de integración continua basado en Maven y QuickBuild**. Esto es extremadamente valioso y ahorra muchísimo tiempo. Es la solución definitiva al viejo problema de “hey, pero sí que funciona en *mí* máquina”. Nuestro **servidor de compilación continua** **actúa como el “juez”** o punto de referencia desde el que determinar la salud de todas nuestras bases de código.

Cada vez que **alguien chequea** algo en el sistema de control de versiones, el **servidor de compilación continua** **arranca, compila todo desde cero en un servidor compartido, y corre todas las pruebas**. Si algo va mal, **manda un correo notificando a todo el equipo** que la compilación ha fallado, incluyendo información sobre qué parte del código falló la compilación exactamente, enlaces a los informes de pruebas, etc.

Todas las noches el **servidor de compilación** continúa **reconstruye el producto desde cero y publica los binarios (ear's, war's, etc.)**, documentación, informes de pruebas, informes de cobertura de pruebas, informes de dependencias, etc., a nuestro portal interno de documentación. Algunos productos también se instalarán automáticamente en un entorno de pruebas.

Montar todo este entorno nos costó un montón de trabajo, pero cada minuto mereció la pena.

Propiedad colectiva del código

Apoyamos el concepto de propiedad colectiva del código, pero no todos los equipos lo han adoptado aun. Hemos encontrado que la programación por parejas con una rotación frecuente de las parejas conduce a un nivel elevado de propiedad colectiva del código.

Los equipos que tienen un alto nivel de propiedad colectiva del código han probado ser muy robustos. Por ejemplo, sus Sprints no fallan simplemente porque una persona clave esté enferma.

Espacio informativo

Todos los equipos tienen acceso a pizarras y espacios vacíos en las paredes, y hacen buen uso de ellos. En la mayoría de las salas encontrarás las paredes empapeladas de toda clase de información sobre el producto y el proyecto. El principal problema es que se va acumulando porquería vieja en las paredes, por lo que puede que introduzcamos un rol de "responsable de limpieza" en cada equipo.

Apoyamos en uso de paneles de tareas, pero no todos los equipos lo han adoptado aun. Ver "Cómo distribuimos la sala del equipo"

Estandarización de código

Últimamente hemos comenzado a definir un estándar de código. Muy útil, ojala lo hubiéramos hecho antes. Prácticamente no cuesta nada de tiempo: comienza con algo simple y deja que vaya creciendo con el tiempo. Escribe únicamente material que no sea obvio para todo el mundo y enlaza a material ya existente siempre que sea posible.

La mayoría de los programadores tienen su propio y distintivo estilo de programación. Pequeños detalles como la forma en la que tratan las excepciones, cómo comentan el código, cuándo devuelven un valor *null*, etc. En algunos casos esta diferencia no importa, pero en otros puede conducir a una severa inconsistencia del diseño del sistema y a un código difícil de leer. Un estándar de código ayuda a que esto no ocurra, siempre que te concentres en las cosas que importan.

He aquí algunos ejemplos de nuestro estándar de código:

- Puedes romper estas reglas, pero asegúrate de que hay una buena razón para ello y documéntala.
- Usa las convenciones de código de Sun por defecto:
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>
- Nunca, nunca, nunca captures excepciones sin registrar la traza de la pila (*stack trace*) o relanzar. *log.debug()* esta bien, pero no pierdas esa traza de la pila.

- Utiliza inyección de dependencias basada en métodos set para escindir unas clases de otras y eliminar dependencias (excepto, por supuesto, cuando una herencia o acompañamiento estricto de clases es deseable)
- Evita usar abreviaturas. Abreviaturas muy conocidas como DAO son admisibles.
- Los métodos que devuelven colecciones o vectores no deberían devolver null. Es mejor devolver colecciones o vectores vacíos.

Ritmo sostenible / trabajo enérgico

Muchos libros sobre desarrollo Ágil de software afirman que las jornadas extendidas son contraproducentes para el desarrollo de software.

Tras algunos experimentos poco deseables en el tema, ¡sólo puedo estar de acuerdo con todo mi corazón!

Hace cosa de un año uno de nuestros equipos (el más grande) estaba trabajando un número insalubre de horas extra. La calidad de la base de código era pésima y habían pasado la mayor parte del tiempo apagando fuegos. El equipo de pruebas (que también estaba haciendo horas extra) no tenía ninguna posibilidad de hacer aseguramiento de la calidad en condiciones. Nuestros usuarios estaban enfadados y la prensa nos estaba devorando vivos.

Después de unos meses conseguimos disminuir las horas de trabajo a un nivel decente. La gente comenzó a trabajar en horarios normales (excepto durante algunas crisis de proyecto, a veces). Y, oh sorpresa, la productividad y la calidad mejoraron notablemente.

Por supuesto, reducir las horas de trabajo no fue en absoluto el único aspecto que condujo a la mejora, pero todos estamos convencidos de que tuvo mucho que ver.

14

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo hacemos pruebas

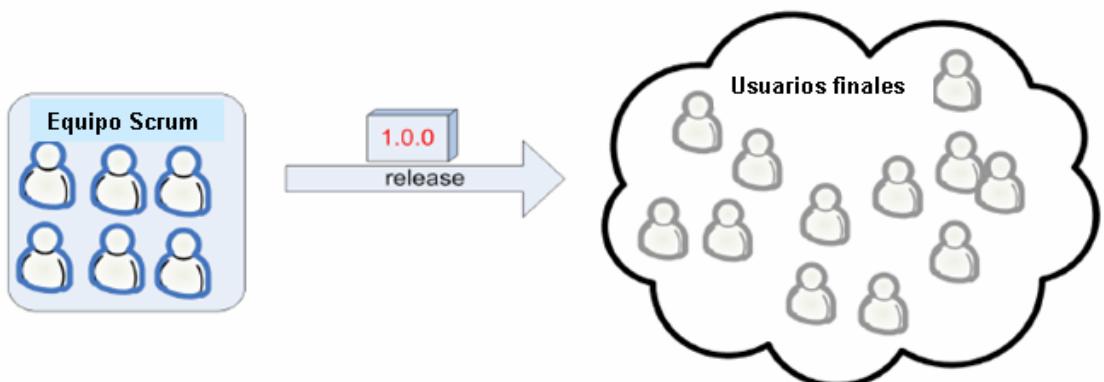
Esta es la parte más dura. No estoy seguro de si la más dura de Scrum, o la más dura del desarrollo de software en general.

Hacer pruebas es la parte probablemente más variará entre diferentes organizaciones. Dependiendo de cuántos encargados de pruebas tengas, cuanta automatización tengas, qué tipo de sistema tengas (¿implemente servidor + aplicación Web, o de hecho vendes software en cajas?), tamaño de los ciclos de entrega, como de crítico sea el software (servidor de blogs vs sistema de control de vuelos), etc.

Hemos experimentado muchísimo sobre cómo hacer pruebas en Scrum. Intentaré describir lo que hemos estado haciendo y lo que hemos aprendido hasta ahora.

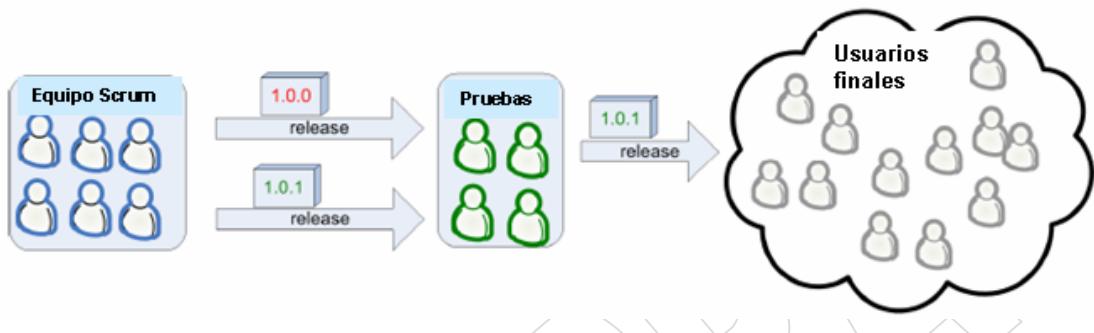
Probablemente no puedes renunciar a la fase de pruebas

En el mundo Scrum ideal, un Sprint produce una versión potencialmente instalable de nuestro sistema. Así que la instalamos, ¿no?



Error.

Nuestra experiencia nos dice que eso **rara vez funciona**. Habrán errores graves. Si la calidad tiene algún valor para ti, es necesario algún tipo de fase de pruebas de aceptación manuales. Se trata de que encargados de **pruebas dedicados** que **no son parte del equipo** **machaquen el sistema** con ese tipo de **pruebas** que el **equipo de Scrum** no pudo imaginar, o no tuvo tiempo de hacer o no contaban con el hardware necesario para implementar. Los **encargados de pruebas acceden al sistema** en la **forma exacta** en la que los **usuarios finales lo harán**, lo que significa que debe hacerse **manualmente** (asumiendo que tu sistema sea para usuarios humanos).



El **equipo de pruebas encontrará errores**, el equipo Scrum tendrá que hacer las correcciones necesarias, y tarde o temprano (esperemos que temprano) será posible lanzar una versión **1.0.1** a los **usuarios finales** en lugar de la inestable **1.0.0**.

Cuando digo **"fase de pruebas"** me refiero a todo el **periodo de pruebas, correcciones y relanzamiento** hasta que haya una versión suficientemente buena como para entrar en producción.

Minimiza la fase de pruebas

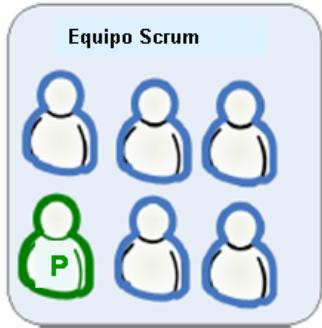
La **fase de pruebas es dura**. La sensación es distintivamente no-Ágil. Aunque no podemos librarnos de ella, sí que podemos minimizarla (y lo hacemos). Más específicamente, **minimizamos la cantidad de tiempo** necesario para la **fase de pruebas**. Esto lo conseguimos:

- **Maximizando la calidad del código desarrollado por el equipo Scrum.**
- **Maximizando la eficiencia del trabajo manual de pruebas** (es decir, encontrar los **beta-testers**, darles las mejores **herramientas** y asegurarnos de que **informan de las tareas que pueden automatizarse**).

Así que ¿cómo maximizamos la calidad del código desarrollado por el equipo Scrum? Bueno, hay muchas maneras. He aquí dos que nos funcionan muy bien:

- **Incluir encargados de pruebas en el equipo Scrum**
- **Hacer menos cosas en cada Sprint**

Incrementar la calidad incluyendo encargados de pruebas en el equipo



Si, ya oigo las dos quejas:

- “¡Pero eso es obvio! ¡Se supone que los **equipos Scrum deben ser multifuncionales!**”
- “¡Se supone que en los **equipos Scrum no debe haber roles!** ¡No podemos tener a una persona que sea **sólo un encargado de pruebas!**”

Permitidme clarificarlo. Lo que quiero decir con “encargado de pruebas” en este caso es “**una persona cuya principal habilidad es hacer pruebas**” y no “**un tipo cuya única responsabilidad es hacer pruebas**”.

Los **desarrolladores** son **frecuentemente muy malos encargados de pruebas**. **Especialmente** los desarrolladores que **deben probar su propio código**.

El encargado de pruebas es quien da el visto bueno

Además de ser “solo” un miembro del equipo, **el encargado de pruebas tiene una labor importante**. Es **el que da el visto bueno**. Nada se considera terminado hasta que **él** dice que está terminado. He encontrado a muchos desarrolladores que dicen que algo está terminado cuando en realidad no lo estaba. Incluso si tienes una definición muy clara de lo que significa “terminado” (algo que deberías tener, ver “Definición de terminado”), los desarrolladores frecuentemente la olvidarán. Los programadores somos gente impaciente y queremos dedicarnos al próximo elemento lo antes posible.

Así que, ¿cómo **sabe** el Señor P. (nuestro encargado de pruebas) que **algo está terminado**? Bueno, antes que nada **debería** (sorpresa) **probarlo**. En muchas ocasiones ocurre que algo que el programador consideraba terminado ni siquiera era **possible de probar**. Porque no se había registrado, o no se había instalado en el servidor de pruebas, o no podía arrancarse o lo que sea. **Una vez** que el Señor P. **ha probado la funcionalidad**, debería **revisar la lista de comprobación de “terminado”** (si tenéis una) **con el desarrollador**. Por ejemplo, si la **definición de “terminado”** establece que **debería haber una nota de versión**, entonces el Señor P. comprueba que **haya una nota de versión**. Si hay algún tipo de especificación más formal para esta funcionalidad (algo raro en nuestro caso) entonces el Señor P. la comprueba también. Etcétera.

Un bonito efecto secundario de esta práctica es que el equipo tiene ahora una persona que está perfectamente preparada para organizar la Demo del Sprint.

¿Qué hace el encargado de pruebas cuando no hay nada que probar?

Esta cuestión siempre está surgiendo. Señor P: "Oye, Scrum Master, no hay nada que probar por el momento, así que ¿qué podría hacer yo?". Puede que lleve una semana completar la primera historia, así que ¿qué debería hacer el encargado de pruebas durante ese tiempo?

Bueno, antes que nada, debería ir *preparando las pruebas*. Esto es, escribir *las especificaciones de las pruebas*, *preparando el entorno de pruebas*, etc. Así, cuando un desarrollador tenga algo que probar, no habrá esperas, el Señor P. podrá zambullirse en ello y comenzar las pruebas.

Si el equipo está *haciendo TDD* entonces todo el mundo pasa un tiempo escribiendo código de pruebas desde el día 1. El encargado de pruebas debería *emparejarse para programar* con los desarrolladores que están programando pruebas. Si el encargado de pruebas no sabe programar, aun así debería *emparejarse para programar* con los desarrolladores, excepto que sólo debería navegar y *dejar al programador el trabajo de teclado*. A un buen encargado de pruebas normalmente *se le ocurren muchos más tipos de pruebas* de las que se le ocurren a un programador, así que se complementan el uno al otro.

Si el equipo no está haciendo TDD, o si no hay mucho trabajo de escritura de pruebas como para ocupar todo el tiempo del encargado de pruebas, este simplemente *debería hacer todo lo que pueda para ayudar al equipo a alcanzar el objetivo del Sprint*. Al igual que todos los miembros del equipo. Si el encargado de pruebas *sabe programar, perfecto*. Si no, el equipo deberá *identificar todas las tareas que no sean de programación y deban hacerse en el Sprint*.

Cuando se *dividen las historias en tareas durante la reunión de planificación* de Sprint, el equipo tiende a *centrarse en tareas de programación*. Sin embargo, usualmente hay montones de tareas no de programación que deben efectuarse en el Sprint. Si *dedicas tiempo a identificarlas durante la fase de planificación* de Sprint, hay muchas probabilidades de que el Señor P. *pueda contribuir mucho*, incluso si no sabe programar y no hay muchas pruebas que hacer ahora mismo.

Algunos ejemplos de tareas no de programación que deben hacerse durante el Sprint:

- Montar un entorno de pruebas.
- Clarificar requisitos.
- Discutir los detalles de instalación con operaciones.
- Escribir documentos de instalación (notas de versión, RFC's, o lo que sea que haga tu organización).
- Contactar con recursos externos (diseñadores de interfaz de usuario, por ejemplo)
- Mejorar los scripts de compilación
- Identificar las preguntas clave de los desarrolladores y conseguir respuestas.

En otro sentido, ¿qué hacemos si el Señor P. se convierte en un cuello de botella? Digamos que estamos en el último día del Sprint y de repente hay montones de cosas terminadas y es imposible para el Señor P. probarlo todo. ¿Qué hacemos? Bueno, podríamos convertir a todo el equipo en asistentes del Señor P. Él decide qué cosas debe hacer por sí mismo y delega las pruebas más rutinarias al resto del equipo. ¡De eso es de lo que va lo de los equipos multifuncionales!

Así que sí, el Señor P. sí que tiene un rol especial en el equipo, pero se le permite hacer otro tipo de trabajo, y otros miembros del equipo también pueden hacer el suyo.

Incrementar la calidad haciendo menos en cada Sprint

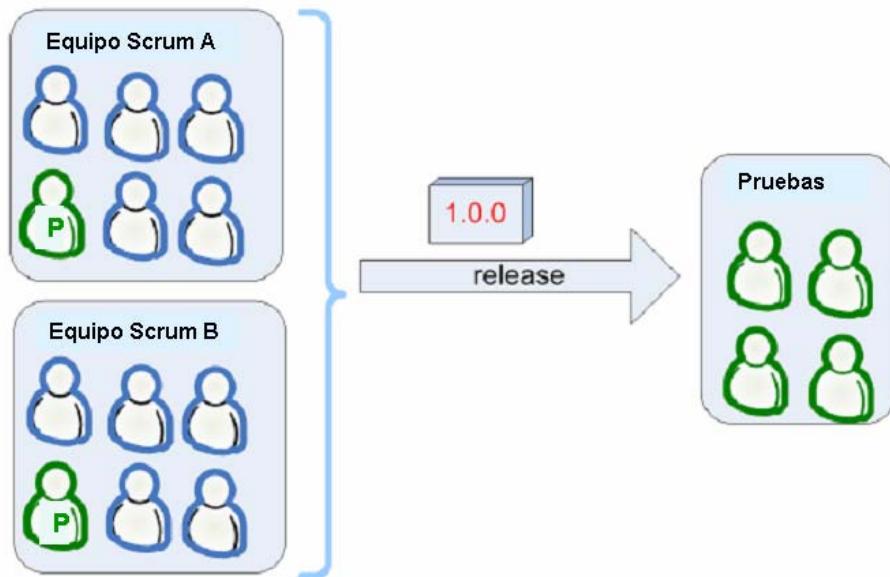
Esto aplica en la reunión de planificación de Sprint. En pocas palabras, ¡no metas demasiadas historias en el Sprint! Si tienes problemas de calidad, o ciclos de pruebas muy largos, ¡haz menos en cada Sprint! Esto conducirá casi automáticamente a una mayor calidad, ciclos de aceptación más cortos, menos errores que afecten al usuario final y mayor productividad a largo plazo ya que el equipo podrá concentrarse en producir cosas nuevas en lugar de arreglar cosas antiguas que siguen rompiéndose.

Casi siempre es más barato producir menos, pero hacerlo estable, en lugar de producir muchísimo y luego tener que hacer parches de emergencia.

¿Deberían las pruebas de aceptación ser parte del Sprint?

Aquí oscilamos bastante. Algunos de nuestros equipos incluyen las pruebas en el Sprint. La mayoría, sin embargo, no lo hacen por dos razones:

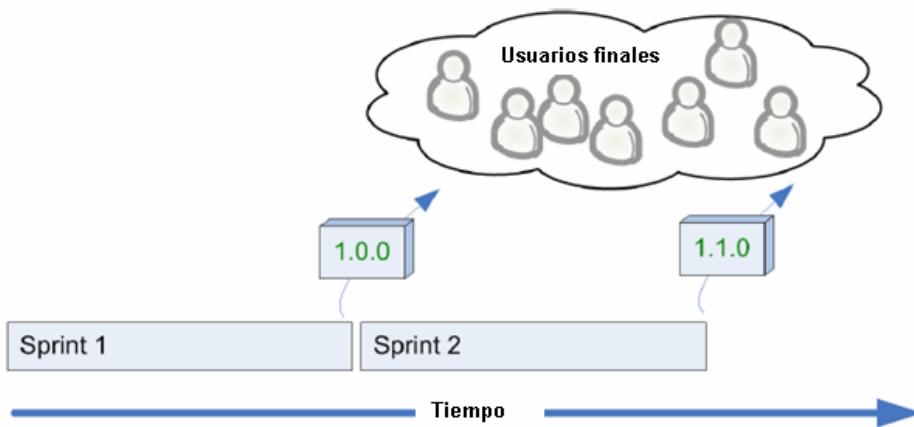
- Un Sprint tiene un tiempo limitado. Las pruebas de aceptación (utilizando mi definición, que incluye corregir los errores y volver a liberar el código) es muy difícil de limitar en el tiempo. ¿Qué ocurre si se acaba el tiempo y todavía hay un error crítico? ¿Vas a sacar una versión en producción con un error crítico? ¿Vas a esperar hasta el próximo Sprint? En la mayoría de los casos ambas soluciones son inaceptables. Así que dejamos las pruebas manuales fuera del Sprint.
- Si tienes múltiples equipos Scrum trabajando en el mismo producto, las pruebas manuales de aceptación deben hacerse sobre el resultado combinado de todos los equipos. Si todos los equipos hicieran pruebas manuales dentro del Sprint, aun necesitarías un equipo externo que hiciera pruebas sobre el resultado final, que sería la combinación del trabajo de ambos equipos.



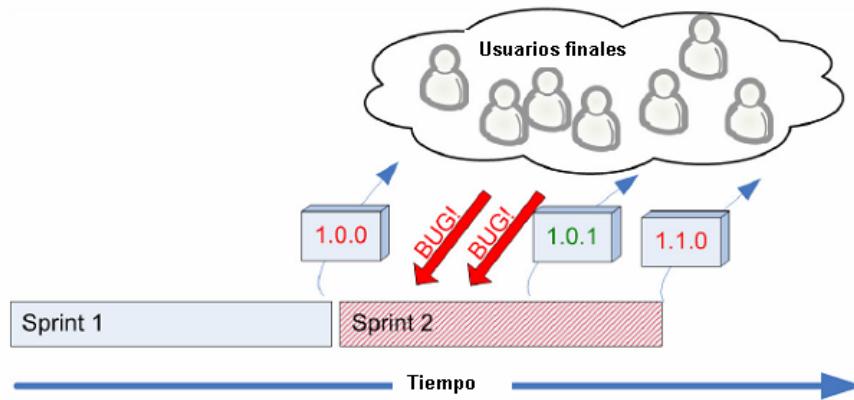
Esta no es en absoluto una solución perfecta, pero es suficientemente Buena para la mayoría de los casos.

Ciclos de Sprint vs. ciclos de pruebas

En un mundo **McScrum perfecto** no necesitarías las fases de pruebas ya que cada **equipo Scrum** entregaría una nueva versión de tu sistema **lista para producción** tras cada **Sprint**.



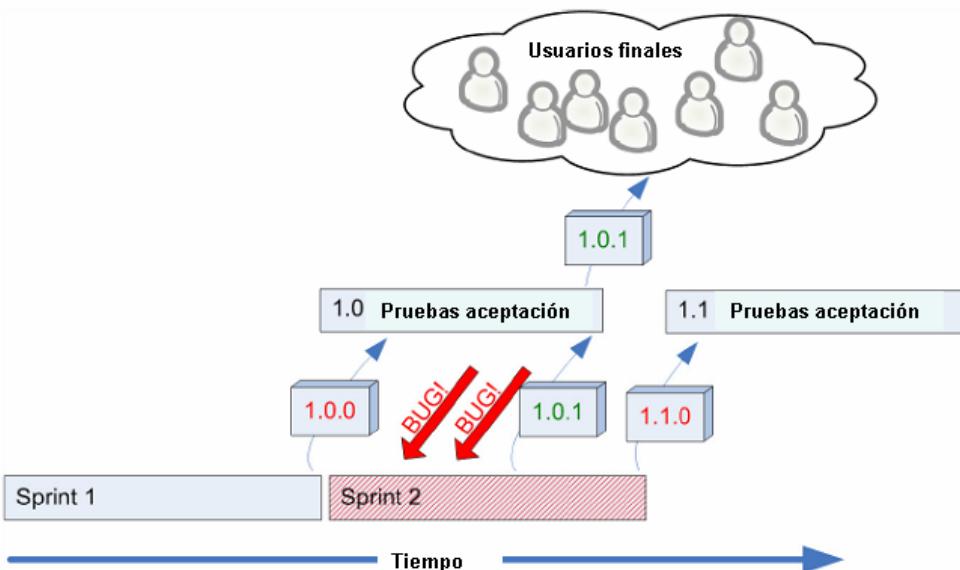
Bueno, he aquí una imagen más realista:



Tras el Sprint 1, se libera una versión 1.0.0 con errores (*bugs*). Durante el Sprint 2, los informes de errores comienzan a llegar, el equipo pasa la mayor parte del tiempo corrigiendo y se le fuerza a liberar una versión 1.0.1 libre de errores a mitad de Sprint. Entonces al final del Sprint se entrega una versión 1.1.0 con nuevas funcionalidades, que por supuesto tiene incluso más errores ya que han tenido menos tiempo para hacerlo bien debido a todas las interrupciones recibidas a raíz de la primera versión. Etc., etc.

Las líneas diagonales rojas en el Sprint 2 simbolizan el caos.

No pinta muy bien, ¿verdad? Bueno, lo triste es que el problema permanece incluso aunque tengas un equipo de pruebas. La única diferencia es que la mayoría de los informes de errores vendrán del equipo de pruebas en lugar de proceder de clientes enfadados. Es una gran diferencia desde el punto de vista del negocio, pero para los desarrolladores es mas o menos lo mismo. Excepto que los encargados de pruebas normalmente son menos agresivos que los clientes finales. Normalmente.



No hemos encontrado una solución simple para este problema. Pero hemos experimentado con varios modelos.

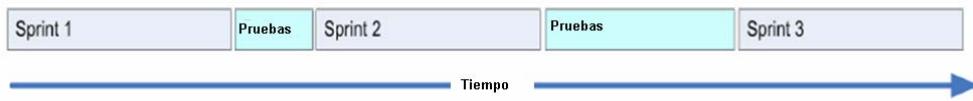
Lo primero de todo, maximiza la calidad del código que el equipo Scrum libera. El coste de encontrar y corregir errores pronto, durante el Sprint, es extremadamente bajo comparado con encontrarlos y corregirlos más tarde.

Pero el problema permanece, incluso aunque minimicemos el número de errores, aun seguirán apareciendo errores después de que se complete el Sprint. ¿Cómo nos enfrentamos a ello?

Enfoque 1: "No empieces a desarrollar nada nuevo hasta que esté en producción lo que has desarrollado"

Suena bien, ¿verdad? ¿Tu también has tenido esa sensación calida?

Hemos estado cerca de adoptar este enfoque varias veces, y hemos diseñado elaborados modelos de cómo podríamos hacerlo. Y sin embargo siempre hemos cambiado de parecer cuando nos damos cuenta de los contratiempos. Tendríamos que añadir un periodo de pruebas no acotado en tiempo entre los Sprints, en el que sólo haríamos pruebas y correcciones hasta que tuviéramos una versión lista para producción.



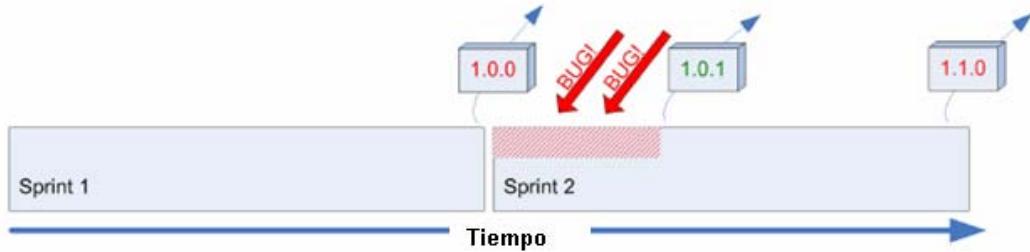
No nos gusta el concepto de tener períodos de pruebas no acotados en tiempo entre Sprints, principalmente porque rompería el ritmo regular de los Sprints. No podríamos decir "cada 3 semanas comenzamos un Sprint". Además, esto no soluciona completamente el problema. Incluso si tenemos un periodo de pruebas, habrían informes urgentes de error apareciendo de cuando en cuando y tendríamos que estar preparados para tratar con ellos.

Enfoque 2: "OK, desarrollamos cosas nuevas, pero priorizamos el poner en producción lo que ya está desarrollado"

Este es nuestro enfoque favorito, al menos hasta ahora.

Básicamente, cuando acabamos un Sprint comenzamos con el siguiente. Pero esperamos pasar parte del tiempo del próximo Sprint encontrando errores del último Sprint. Si el próximo Sprint queda seriamente dañado porque hemos tenido que pasar demasiado tiempo arreglando errores del Sprint anterior, evaluamos por qué ha ocurrido esto y cómo podemos mejorar la calidad. Nos aseguramos de que los Sprints son suficientemente largos como para soportar una cantidad aceptable de correcciones del último Sprint.

Gradualmente, durante un periodo de muchos meses, la cantidad de tiempo que pasamos solucionando errores de Sprints anteriores fue disminuyendo. Adicionalmente, fuimos capaces de dedicar menos personas involucradas cuando se producían errores, así que todo el equipo no tenía que ser molestado en cada ocasión. Ahora estamos a un nivel mucho más aceptable.



Durante las reuniones de planificación de Sprint establecemos el factor de dedicación suficientemente bajo como para tener en cuenta el tiempo que esperamos dedicar a encontrar y corregir errores del último Sprint. Con el tiempo, los equipos se han vuelto muy buenos estimando este tiempo. La métrica de velocidad ayuda mucho (ver “Cómo decide el equipo qué historias incluir en el Sprint”).

Mal enfoque – “centraos en producir cosas nuevas”

Esto, a todos los efectos, significa “centraos en producir cosas nuevas *en lugar de conseguir poner producción las cosas antiguas*”. ¿Quién querría hacer eso? Y aun así cometimos ese error muy a menudo al principio, y estoy seguro de que muchas otras compañías lo hacen también. Es una enfermedad relacionada con el estrés. Muchos gerentes realmente no lo entienden, y cuando toda la programación termina aun se encuentran muy lejos de poner algo en producción. Al menos para sistemas complejos. Así que el gerente (o el Dueño de Producto) pide al equipo que siga incluyendo nuevas funcionalidades mientras la bolsa de código-casi-listo-para-producción va volviéndose más y más pesada, ralentizándolo todo.

No sobrecargues el eslabón más débil de tu cadena

Digamos que las pruebas de aceptación son tu eslabón más lento. Tienes muy pocos encargados de pruebas, o las pruebas de aceptación tardan demasiado por la baja calidad del código. Digamos que tu equipo de pruebas puede probar como mucho 3 funcionalidades a la semana (no, no usamos “funcionalidades a la semana” como métrica; lo estoy usando únicamente como ejemplo). Y digamos que tus desarrolladores pueden desarrollar 6 nuevas funcionalidades por semana.

Sería tentador para los gerentes o Dueños de Producto (o quizás incluso para el equipo) planificar el desarrollo de 6 nuevas funcionalidades a la semana. ¡No lo hagáis! La realidad os alcanzará de una u otra manera, y será doloroso.

En lugar de eso, planificad 3 nuevas funcionalidades a la semana y pasad el resto del tiempo aligerando el cuello de botella de las pruebas. Por ejemplo:

- Haz que unos cuantos desarrolladores trabajen como encargados de pruebas (oh, te adorarán por ello...).
- Implementa herramientas y scripts que hagan las pruebas más fáciles.
- Añade más código de pruebas automatizadas
- Incrementa la longitud de los Sprints y haz que se incluyan pruebas de aceptación en los Sprints.
- Define algunos Sprints como “Sprint de pruebas” en los que todo el equipo trabaje como un equipo de pruebas.
- Contrata a más encargados de pruebas (incluso aunque signifique eliminar desarrolladores).

Hemos probado todas estas soluciones (excepto la última). Las mejores soluciones a largo plazo son por supuesto la segunda y la tercera, es decir, mejores herramientas y script y automatización de las pruebas.

Las retrospectivas son un buen foro en el que identificar el eslabón más débil de la cadena.

De vuelta a la realidad

Probablemente te he dado la impresión de que tenemos encargados de pruebas en todos los equipos Scrum, que tenemos enormes equipos de pruebas para cada producto, que tenemos versiones listas después de cada Sprint, etc., etc.

Bueno, pues no.

En ocasiones hemos conseguido todo esto, y hemos visto los efectos positivos que producen. Pero aun estamos lejos de un proceso aceptable de aseguramiento de la calidad, y aun tenemos mucho que aprender al respecto.

15

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo manejar múltiples equipos Scrum

Un montón de cosas son más complicadas cuando tienes varios equipos Scrum trabajando en el mismo producto. Este problema es universal y realmente no tiene mucho que ver con Scrum. Más desarrolladores = más complicaciones. Nosotros (como es habitual) hemos experimentado con ello. El máximo que hemos tenido han sido 40 personas trabajando en el mismo producto.

Las preguntas clave son:

- Cuántos equipos crear
- Cómo distribuir a la gente en equipos

Cuántos equipos crear

Si tratar con múltiples equipos Scrum es tan duro, ¿por qué nos preocupamos? ¿Por qué no poner a todo el mundo en el mismo equipo?

El mayor equipo Scrum que hemos tenido tuvo en torno a 11 personas. Funcionó, pero no demasiado bien. Los Scrum diarios tendían a alargarse más de los 15 minutos. Los miembros del equipo no sabían lo que otros miembros estaban haciendo, así que había bastante confusión. Era difícil para el Scrum Master mantener a todo el mundo alineado con el objetivo y encontrar tiempo para tratar con todos los obstáculos que se encontraban.

La alternativa es dividir en dos equipos. Pero, ¿es esto mejor? No necesariamente.

Si el equipo tiene experiencia y se siente a gusto con Scrum, y hay una manera lógica de dividir el plan de producto en dos caminos distintos, y esos dos caminos no involucran el mismo código, entonces diría que es una buena idea dividir el equipo. En otro caso, consideraría mantener un solo equipo a pesar de las desventajas de un equipo grande.

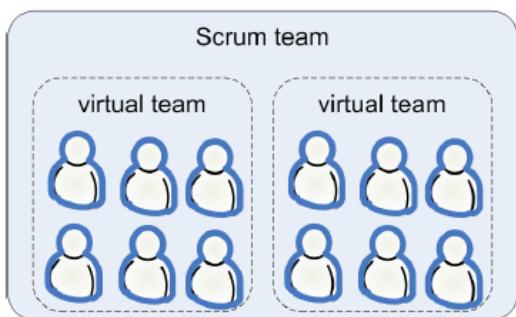
Mi experiencia me dicta que es mejor tener menos equipos que sean demasiado grandes que tener muchos equipos pequeños que están todo el rato interfiriendo unos con otros. ¡Haz equipos pequeños sólo cuando no necesiten interferir unos con otros!

Equipos virtuales

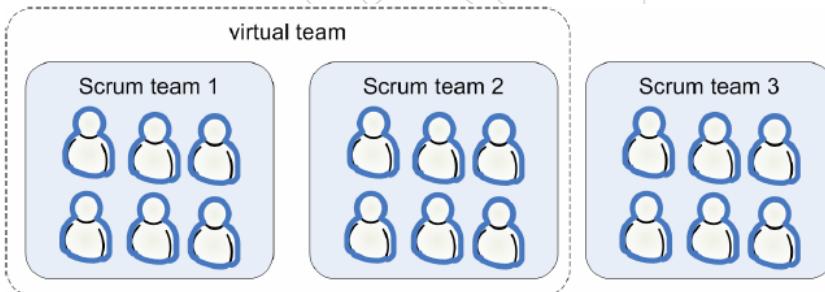
¿Como sabes si has tomado una Buena decisión respecto a los pros y los contras de “equipo grande” frente a “equipo pequeño”?

Si mantienes tus ojos y tus oídos abiertos puede que notes como se forman “equipos virtuales”.

Ejemplo 1: Escoges tener un equipo grande. Pero cuando observas quién habla con quién durante el Sprint notas que el equipo se ha separado a todos los efectos en dos sub-equipos.



Ejemplo 2: Escoges tener tres equipos más pequeños. Pero cuando empiezas a ver quién habla con quién durante el Sprint notas que el equipo 1 y el equipo 2 hablan todo el rato, mientras que el equipo 3 trabaja por su cuenta.



Así que ¿qué significa todo esto? ¿Que tu estrategia de división era errónea?

Sí, si el equipo virtual parece permanente. No, si el equipo virtual parece algo eventual.

Observa otra vez el ejemplo 1. Si los dos sub-equipos virtuales tienden a cambiar cada cierto tiempo (por ejemplo, la gente cambiar de sitio entre los diferentes sub-equipos virtuales) entonces probablemente habrás hecho la decisión correcta al mantenerlos como un solo equipo Scrum. Si los dos sub-equipos se mantienen igual durante todo el Sprint probablemente prefieras dividirlos en dos equipos Scrum para el próximo Sprint.

Ahora observa otra vez el ejemplo 2. Si el equipo 1 y el equipo 2 hablan entre ellos (y no con el equipo 3) durante todo el Sprint, probablemente quieras combinar los equipos 1 y 2 en un solo equipo Scrum el próximo Sprint. Si el equipo 1 y el equipo 2 hablan entre ellos durante la primera mitad del Sprint, y

entonces el equipo 1 y el equipo 3 hablan entre ellos durante la segunda mitad del Sprint, entonces quizás deberías considerar combinar los tres equipos en uno, o simplemente dejarlos como tres equipos Scrum. Comenta el tema durante la retrospectiva de Sprint y deja que los equipos decidan por si mismos.

La división en equipos es una de las partes realmente duras de Scrum. No pienses demasiado u optimices en exceso. Experimenta, mantente alerta ante la formación de equipos virtuales y asegúrate de que tienes tiempo de sobra para discutir estos aspectos durante las retrospectivas. Tarde o temprano encontrarás la solución correcta para tu situación concreta. Lo importante es que los equipos se sientan a gusto y no tropiecen unos con otros demasiado a menudo.

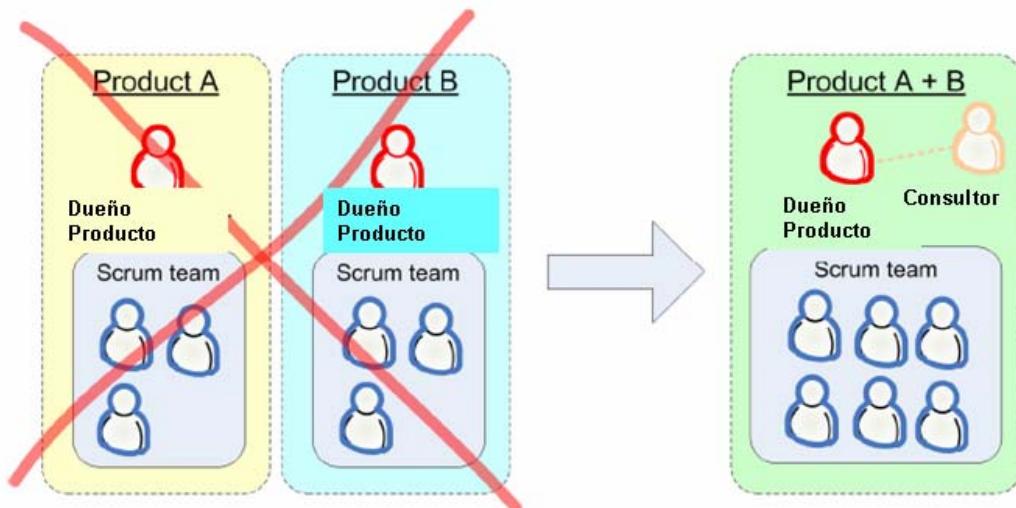
Tamaño óptimo del equipo

La mayoría de los libros que he leído afirman que el tamaño óptimo oscila entre 5-9 personas.

Por lo que he visto hasta ahora solo puedo estar de acuerdo. Aunque yo diría 3-8 personas. De hecho, creo que merece la pena pasar algunos malos ratos con tal de conseguir equipos de ese tamaño.

Digamos que tienes un solo equipo Scrum de 10 personas. Considera deshacerte de los dos miembros mas débiles del equipo. Oooops, ¿realmente acabo de decir eso?

Digamos que tienes dos productos diferentes, con equipos de 3 personas por producto y ambos avanzando muy lentamente. Podría ser una buena idea combinarlos en un solo equipo de 6 personas responsable del desarrollo de ambos productos. En este caso, deshazte de uno de los dos Dueños de Producto (o dale un puesto de consultor o similar).



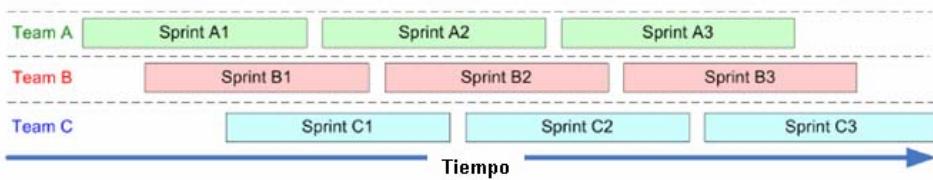
Digamos que tienes un solo equipo de 12 personas, porque la base de código se encuentra en un estado tan precario que no hay manera de que dos equipos trabajen en ella de forma independiente. Haz un serio esfuerzo en arreglar la base de código (en lugar de desarrollar nuevas funcionalidades) hasta que

llegues a un punto en el que puedes dividir el equipo. La inversión probablemente produzca sus frutos muy pronto.

¿Sprints sincronizados, o no?

Digamos que tienes a tres equipos Scrum trabajando en el mismo producto. ¿Deberían los Sprints sincronizarse, es decir, empezar y acabar al mismo tiempo, o deberían solaparse?

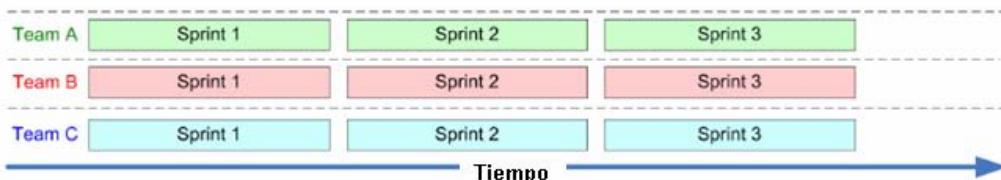
Nuestro primer enfoque fue tener Sprints solapados en el tiempo:



Esto sonaba muy bien. En cualquier momento habría un Sprint a punto de acabar y otro recién comenzado. La carga de trabajo del Dueño de Producto podría distribuirse equitativamente en el tiempo. Habría un flujo constante de entregas. Demos cada semana. Aleluya.

Sí, lo sé, pero realmente sonaba tan convincente al principio!

Habíamos comenzado a hacer esto cuando un día tuve la oportunidad de hablar con Ken Schwaber (cuando conseguí mi certificación en Scrum). Él opinó que era una mala idea, que era mucho mejor sincronizar los Sprints. No recuerdo exactamente sus argumentos, pero después de un poco de discusión me había convencido.

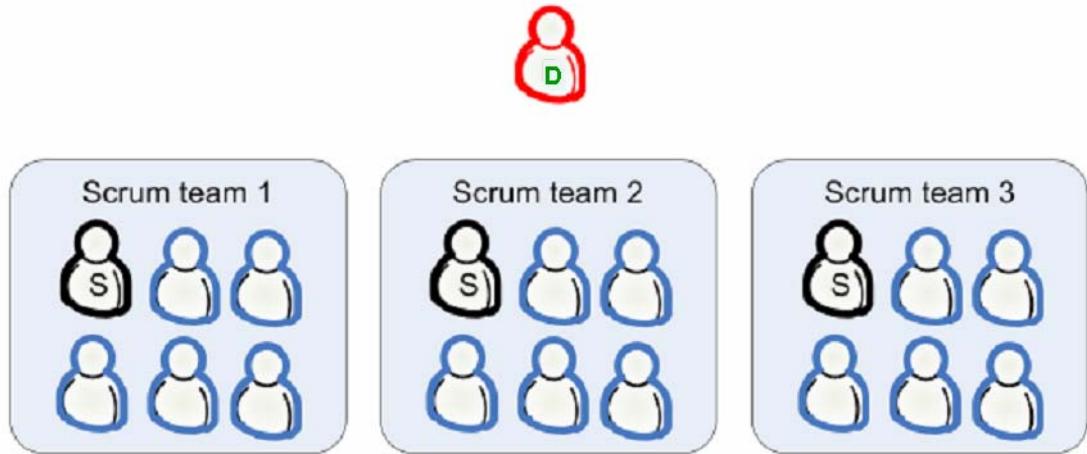


Esta es la solución que usamos desde entonces y nunca nos hemos arrepentido. Nunca sabremos si la estrategia de Sprints solapados habría fallado, pero yo pienso que sí. Las ventajas de los Sprints sincronizados son:

- Hay un momento natural en el que redistribuir los equipos - ¡entre los Sprints! Son Sprints solapados, no hay manera de reorganizar los equipos sin entorpecer al menos a un equipo a mitad de Sprint.
- Todos los equipos pueden trabajar juntos en el mismo objetivo de Sprint y hacer reuniones de planificación de Sprint juntos, lo que conduce a una mejor colaboración entre equipos.
- Menos sobrecarga administrativa, es decir, menos reuniones de planificación de Sprint, demos y versiones.

Por qué introdujimos un rol de “guía de equipo”

Digamos que tenemos un solo producto con tres equipos:



El tipo de rojo con la D es el Dueño de Producto. Los tipos de negro con la S son Scrum Masters. El resto son esbirros...er...respetables miembros del equipo.

En esta constelación, ¿Quién decide qué personas deberían estar en qué equipos? ¿El Dueño de Producto? ¿Los tres Scrum Masters juntos? ¿O debería cada persona seleccionar su equipo? Pero ¿qué pasa si todo el mundo quiere estar en el equipo 1 (porque el Scrum Master es tan guapo)?

¿Y si luego resulta que realmente no es posible tener más de dos equipos trabajando en paralelo en esta base de código, así que necesitamos transformar esta situación en 2 equipos de 9 personas en lugar de 3 equipos de 6? Eso significa 2 Scrum Masters. Así que ¿Cuál de los 3 Scrum Masters actuales será despojado de su título?

En muchas empresas estas pueden ser cuestiones muy delicadas.

Es muy tentador dejar que sea el Dueño de Producto el que haga la distribución y reorganización de las personas. Pero ¿es realmente labor del Dueño de Producto? El Dueño de Producto es el experto que dice al equipo en qué dirección debería correr. No debería involucrarse en los detalles de bajo nivel. Especialmente si es una “gallina” (si es que has oído la anécdota de las gallinas y los cerdos, y si no busca en Google “scrum gallinas y cerdos”).

Nosotros solucionamos esto introduciendo el rol de “guía de equipo”. Esto correspondería a lo que podríamos llamar el “Scrum de Scrum Masters” o “el Jefe” o “Scrum Master Jefe”, etc. No tiene que liderar ningún equipo, pero es responsable de los asuntos entre equipos como quién debería ser Scrum Master en cada equipo, cómo debería dividirse la gente entre equipos, etc.

Pasamos un tiempo complicado escogiendo un nombre para este rol. “Guía de equipo” fue el nombre menos penoso que pudimos encontrar.

Esta solución ha funcionado para nosotros y puedo recomendarla (independientemente de cómo quieras llamar al puesto).

Como asignamos personas a los equipos

Hay dos estrategias generales para asignar personas a los equipos cuando tienes múltiples equipos para el mismo producto.

- Dejar que una persona designada haga la distribución, por ejemplo el "guía de equipo" que mencioné antes, el Dueño de Producto o el Gerente funcional (si es que está suficientemente involucrado como para hacer buenas decisiones al respecto).
- Dejar que sean las propias personas las que decidan de alguna forma.

Nosotros hemos experimentado con las tres. ¿Tres? Sí. Estrategia 1, estrategia 2 y una combinación de ambas.

Hemos descubierto que la combinación de ambas es lo que mejor funciona.

Antes de la reunión de planificación de Sprint, el guía de equipo convoca una reunión de distribución de personal con el Dueño de Producto y todos los Scrum Masters. Hablamos sobre el último Sprint y decidimos si es necesaria una redistribución del personal. Quizás queramos combinar dos equipos, o cambiar algunas personas de un equipo a otro. Decidimos algo y lo ponemos por escrito como propuesta de distribución de equipos, y lo llevamos a la reunión de planificación de Sprint. Lo primero que hacemos en la reunión de planificación de Sprint es repasar los elementos más prioritarios de la Pila de Producto. El guía de equipos dice algo así como:

"Hola a todos. Sugerimos la siguiente distribución de personas para el próximo Sprint."

Distribución preliminar de equipos		
Team 1	Team 2	Team 3
- tom - jerry - donald - mickey	- goofy - daffy - humpty - dumpty	- minnie - scrooge - winnie - roo

"Como podéis ver, esto significa una reducción de 4 a 3 equipos. Hemos listado a los miembros de cada equipo. Por favor, agrupaos y elegid una sección de la pared."

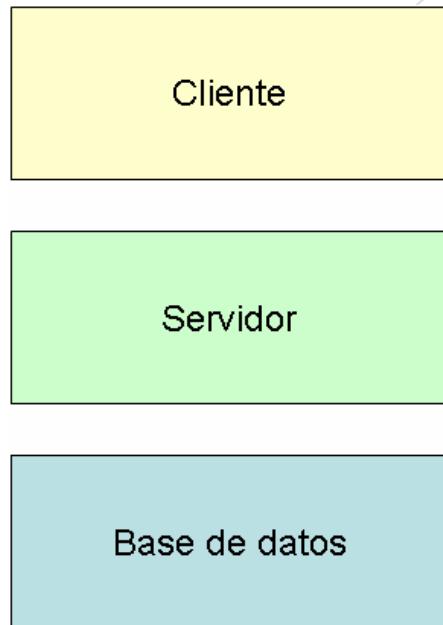
(el guía de equipo espera mientras la gente pasea por la sala, y poco después hay tres grupos de personas, cada uno de pie frente a una sección vacía de la pared).

"Bueno, esta es una distribución *preliminar*. Es sólo un punto de partida, para ahorrar tiempo. Conforme la reunión de planificación de Sprint prospere, **sentíos libres** de **cambiar de equipo, dividir vuestro equipo en dos, combinarlos con otro equipo o lo que sea**. Usad el sentido común en función de las prioridades marcadas por el Dueño de Producto."

Esto es lo que nos ha funcionado mejor. Un cierto nivel de control centralizado al principio, seguido de un cierto nivel de optimización descentralizada después.

¿Equipos especializados – o no?

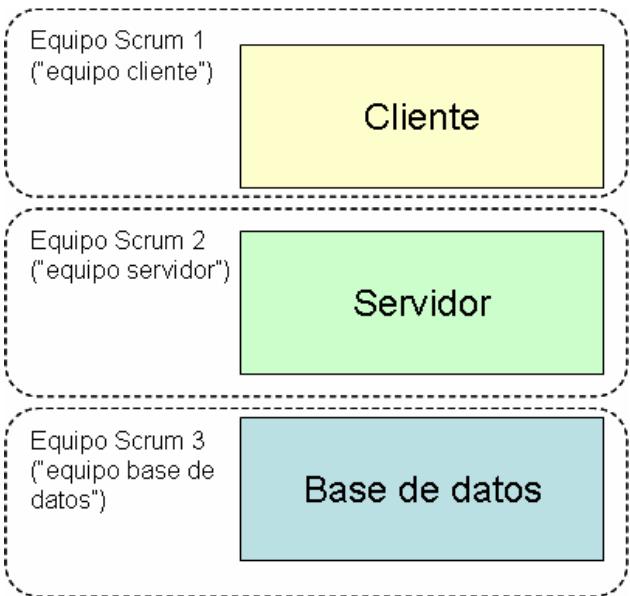
Digamos que tu tecnología consiste en tres componentes principales:



Y digamos que tienes 15 personas trabajando en este producto, así que no quieres manejarlos como un solo equipo Scrum. ¿Qué equipos crearías?

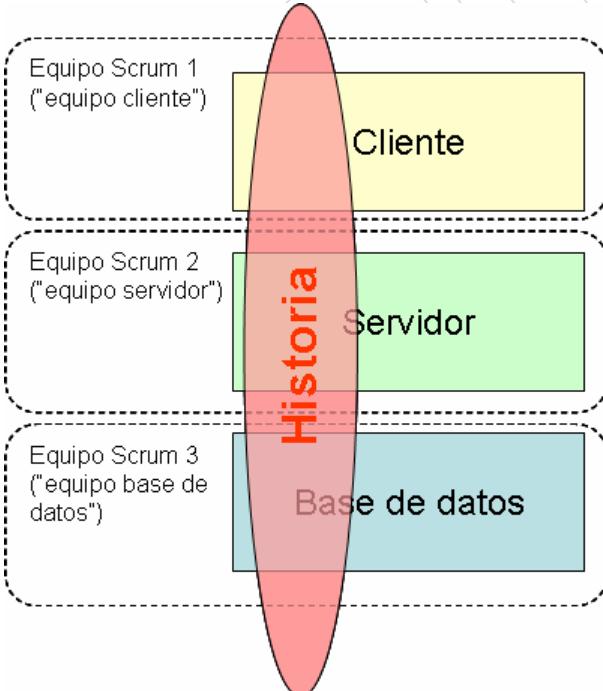
Enfoque 1: equipos especializados

Una aproximación es crear equipos especializados en componentes, como el "equipo cliente", "equipo servidor" y "equipo base de datos".



Es así como empezamos nosotros. **No funciona muy bien, al menos no cuando las historias involucran múltiples componentes.**

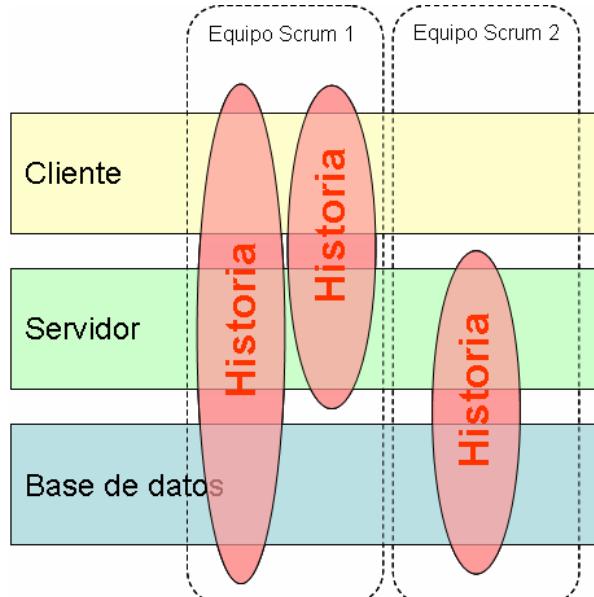
Por ejemplo, digamos que hay una historia llamada “tablón de anuncios en el que los usuarios pueden colgar mensajes para los demás”. Esta funcionalidad de tablón de anuncios incluiría actualizar el interfaz de usuario en el cliente, añadir lógica al servidor y añadir algunas tablas a la base de datos.



Esto significa que los tres equipos – el equipo cliente, el equipo servidor y el equipo base de datos – tienen que colaborar para conseguir terminar la historia. No parece buena idea.

Enfoque 2: equipos multi-funcionales

Una segunda aproximación es crear **equipos multifuncionales**, es decir, **equipos que no están limitados a un componente específico**.



Si **muchas de vuestras historias implican múltiples componentes, este tipo de estrategia de división funcionará mejor**. Cada equipo puede implementar una historia completa incluyendo las partes de cliente, servidor y base de datos. Los equipos pueden así **trabajar de forma más independiente**, lo cuál es bueno.

Una de las primeras cosas que hicimos al comenzar con Scrum fue **disolver los equipos especializados por componentes** y crear equipos multi-funcionales en su lugar (enfoque 2). Esto **diminuyó el número de casos de "no podemos completar este elemento** porque estamos **esperando a que los tíos del equipo servidor hagan su parte".**

Sin embargo, de vez en cuando creamos equipos temporales especializados en algún componente cuando hay una fuerte necesidad.

¿Redistribuir equipos entre Sprints – o no?

Cada Sprint suele ser diferente de los anteriores, dependiendo de qué tipo de historias son la **máxima prioridad** en cada momento. Como consecuencia, la distribución óptima de cada equipo puede ser **diferente en cada Sprint**.

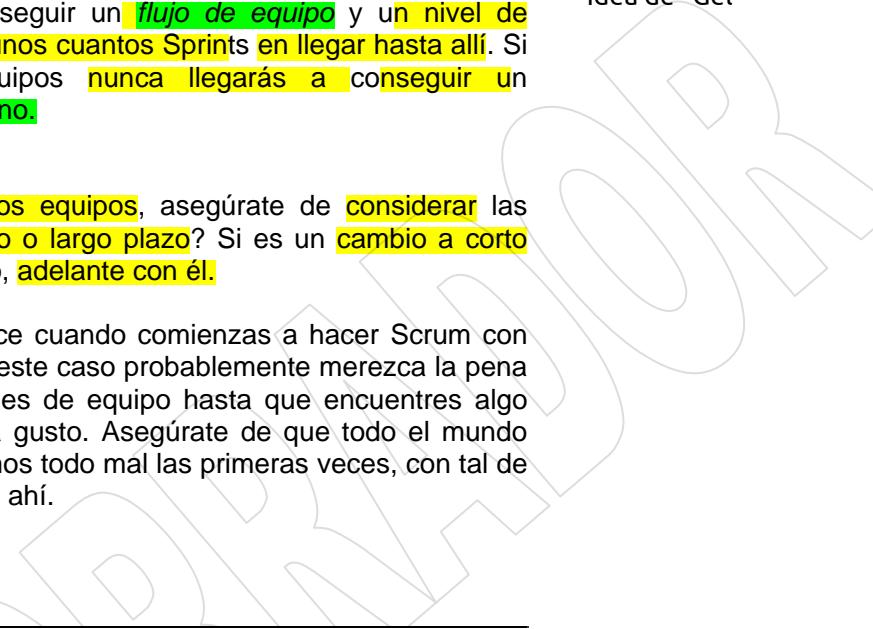
De hecho, en casi todos los Sprints empezamos diciendo algo como “**este Sprint no es un Sprint normal ya que (bla bla bla)...**”. Tras un tiempo simplemente hemos abandonado la idea de un Sprint “normal”. **No hay Sprints normales**. Al igual que **no hay familias “normales” o personas “normales”**.

Puede ser Buena idea **durante un Sprint tener un equipo de sólo-cliente**, formado por todos los que conocen la base de código de cliente realmente bien,

y durante el próximo Sprint puede ser buena idea tener dos equipos multifuncionales y dividir a la gente del equipo cliente entre los dos.

Uno de los aspectos clave de Scrum es el “acoplamiento de equipo”, es decir, si un equipo llega a trabajar unido durante muchos Sprints, usualmente se volverá muy cohesionado. Aprenderán a conseguir un flujo de equipo y un nivel de productividad increíble. Pero se tarda unos cuantos Sprints en llegar hasta allí. Si persistes en ir cambiando los equipos nunca llegarás a conseguir un acoplamiento de equipo realmente bueno.

Idea de "Gel"



Así que cuando quieras redistribuir los equipos, asegúrate de considerar las consecuencias. ¿Es un cambio a corto o largo plazo? Si es un cambio a corto plazo considera saltártelo. Si es a largo, adelante con él.

Una excepción a considerar se produce cuando comienzas a hacer Scrum con un equipo grande por primera vez. En este caso probablemente merezca la pena experimentar un poco con subdivisiones de equipo hasta que encuentres algo con lo que todo el mundo se sienta a gusto. Asegúrate de que todo el mundo entienda que no pasa nada si lo hacemos todo mal las primeras veces, con tal de que continuemos mejorando a partir de ahí.

Miembros a tiempo parcial

Solo puedo confirmar lo que dicen los libros sobre Scrum: tener miembros a tiempo parcial no es buena idea.

Digamos que estás a punto de incorporar a Joe como miembro a tiempo parcial de tu equipo Scrum. Piénsatelo bien primero. ¿Realmente necesitas a Joe en el equipo? ¿Estás seguro de que no puedes incorporar a Joe a tiempo completo? ¿Cuáles son sus otros compromisos? ¿Puede otra persona encargarse de los compromisos de Joe y permitir que Joe tome un papel menos activo, de soporte, en dichos compromisos? ¿Puede Joe unirse a tiempo completo el próximo Sprint y mientras tanto ir transfiriendo sus otras responsabilidades a otra persona?

A veces simplemente no hay otra solución. Necesitas a Joe desesperadamente porque es el único administrador de base de datos en el edificio, pero el resto de equipos también le necesitan así que no puede comprometerse a tiempo completo, y la compañía no puede contratar a más administradores de base de datos. Estupendo. Ese es un caso válido para incorporarle a tiempo parcial (que por cierto es exactamente lo que nos pasa a nosotros). Pero asegúrate de realizar esta evaluación cada vez.

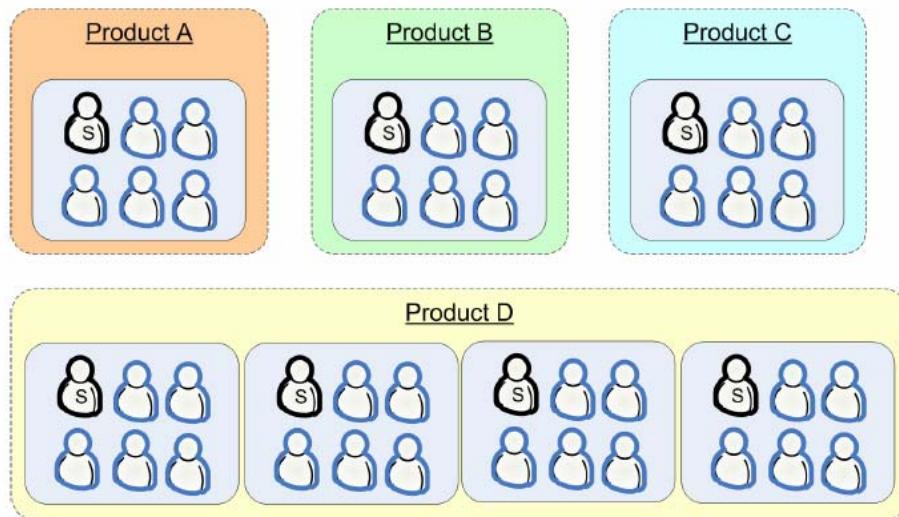
En general prefiero tener un equipo de 3 a tiempo completo que de 8 a tiempo parcial.

Si tienes a una persona que divide su tiempo entre diferentes equipos, como el administrador de bases de datos de antes, es buena idea que de todas formas esté asignado principalmente a un equipo. Estima cuál es el equipo que le necesitará más y conviértelo en su “equipo matriz”. Cuando nadie esté tirando de él, atenderá a los Scrum diarios, planificaciones de Sprint, retrospectivas, etc., de ese equipo.

Como hacemos Scrum de Scrums

Scrum de Scrums es básicamente una reunión periódica a la que los Scrum Masters acuden para hablar.

En un momento dado tuvimos cuatro productos, con tres de los productos asignados a un equipo cada uno y el último producto asignado a un grupo de 25 personas divididas en varios equipos Scrum. Algo como esto:



Esto significa que teníamos dos niveles de Scrum de Scrums. Teníamos un "nivel de producto" que consistía en todos los equipos asignados al producto D, y un "nivel de empresa" que consistía en todos los productos.

Scrum de Scrums a nivel producto

Esta reunión es muy importante. La hacíamos una vez a la semana, a veces incluso más a menudo. Discutíamos aspectos de integración, equilibrio entre equipos, preparación de la próxima reunión de planificación de Sprint, etc. Disponíamos de 30 minutos, aunque frecuentemente nos pasábamos. Una alternativa hubiera sido tener Scrum de Scrums todos los días, pero nunca conseguimos llevarla adelante.

Nuestra agenda Scrum de Scrums era:

- 1) Todo el mundo en la mesa describe qué ha conseguido su equipo la última semana, qué planean hacer esta semana y qué impedimentos tienen.
- 2) Cualquier otro asunto inter-equipo que necesite ser discutido, por ejemplo aspectos relativos a la integración.

La agenda del Scrum de Scrums no es realmente importante para mí, lo importante es que se celebren reuniones Scrum de Scrums regularmente.

Scrum de Scrums a nivel empresa

Bautizamos estas reuniones como "El Pulso". Las hemos celebrado en diferentes formatos, con distintos participantes. Últimamente hemos descartado el concepto y lo hemos sustituido por una reunión semanal de todas las personas involucradas en el desarrollo. 15 minutos.

¿Cómo? ¿15 minutos? ¿Todo el mundo? ¿Todos los miembros de todos los productos? ¿Y eso funciona?

Pues sí, funciona si la persona que dirige la reunión es estricta respecto a la duración.

El formato de la reunión es:

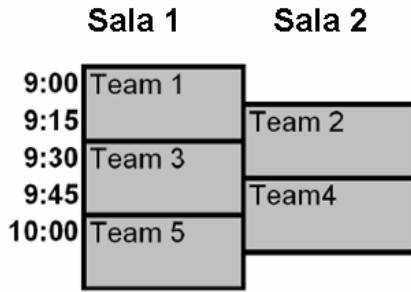
- 1) Noticias y actualizaciones del Jefe de Desarrollo. Información sobre eventos próximos, por ejemplo.
- 2) Un turno por producto. Una persona de cada grupo de producto informa sobre lo que consiguieron la semana pasada, lo que piensan conseguir esta semana y cualquier problema que haya surgido. Otras personas también informan (responsable de calidad, responsable de gestión de configuraciones, etc.).
- 3) Cualquier persona es libre de añadir información o formular preguntas.

Este es un foro de información rápida, no de discusión o reflexión. Mantenerlo en 15 minutos normalmente funciona. A veces nos pasamos, pero rara vez llegamos a más de 30 minutos en total. Si aparecen discusiones interesantes, hago una pausa e invito a las personas interesadas en dicha discusión a que se queden después de la reunión y continúen con ella.

¿Por qué hacemos una reunión de pulso con todo el mundo? Porque nos dimos cuenta de que el Scrum de Scrums a nivel empresa trataba sobre todo de informar. Rara vez teníamos discusiones en él. Adicionalmente, muchas otras personas fuera de esta reunión estaban realmente necesitadas de este tipo de información. Básicamente, los equipos quieren saber qué están haciendo los demás equipos. Así que decidimos que si íbamos a reunirnos y dedicar un tiempo a informarnos unos a otros sobre qué estaba haciendo cada equipo, por qué no dejar que todo el mundo asistiera.

Intercalando los Scrums diarios

Si tienes muchos equipos Scrum en un solo producto y todos ellos hacen el Scrum diario al mismo tiempo, tienes un problema. El Dueño de Producto (y los fisgones como yo) sólo pueden atender a una de las reuniones de Scrum diarias. Así que le pedimos a los equipos que intenten no tener las reuniones al mismo tiempo.



El ejemplo del esquema anterior data de cuando realizábamos los Scrum diarios en salas aparte en lugar de en la sala del equipo. Las reuniones tardan normalmente 15 minutos cada una, pero cada equipo tiene una reserva de 30 minutos por si necesitasen pasarse un poco de la hora.

Esto es *extremadamente útil* por dos razones:

1. Personas como el Dueño de Producto o yo mismo podemos visitar todos los Scrum diarios en una sola mañana. No hay mejor manera de obtener una visión exacta de cómo está funcionando el Sprint y cuáles son las amenazas clave.
2. Los equipos pueden visitar los Scrum diarios de otros equipos. No ocurre muy a menudo, pero cada cierto tiempo dos equipos pueden acabar trabajando en áreas similares, así que algunos miembros de un equipo se dejan caer por el Scrum diario del otro equipo para mantenerse sincronizados.

La desventaja es una menor libertad para el equipo – no pueden escoger la hora que quieran para el Scrum diario. Esto, en todo caso, no nos ha causado ningún problema por ahora.

Equipos apagafuegos

Tuvimos una situación en la que un producto muy grande no podía adoptar Scrum porque pasaban demasiado tiempo apagando fuegos, es decir, liberando parches de emergencia para su sistema, que había sido entregado en un estado muy prematuro de desarrollo. Este era realmente un círculo vicioso, ya que estaban tan ocupados apagando fuegos que no tenían tiempo para trabajar proactivamente en prevenir fuegos (es decir, mejorar el diseño, automatizar las pruebas, crear herramientas de monitorización, herramientas de alarmas, etc.).

Solucionamos este problema creando un equipo apagafuegos dedicado y un equipo Scrum.

La labor del equipo Scrum era (con las bendiciones del Dueño de Producto) intentar estabilizar el sistema y, efectivamente, prevenir fuegos.

El equipo apagafuegos (al que realmente llamábamos “soporte”) tenía dos labores:

- 1) Apagar fuegos
- 2) Proteger al equipo Scrum de todo tipo de distracciones, incluyendo cosas como defenderse de peticiones de funcionalidades ad-hoc que aparecían de la nada.

El equipo apagafuegos se colocó cerca de la puerta, mientras que el equipo Scrum fue colocado en la zona más lejana de la habitación. Así que el equipo de soporte podía proteger *físicamente* al equipo Scrum de distracciones como un vendedor impaciente o un cliente enfadado.

Se **destinaron programadores senior a ambos equipos**, de forma que **un equipo no fuera demasiado dependiente de las competencias clave del otro**.

Este fue, básicamente, un **intento de resolver un problema de implementación de Scrum desde cero**. ¿**Cómo podemos comenzar a usar Scrum si el equipo no tiene oportunidad de planificar su trabajo para más allá de un día?** Nuestra estrategia fue, como se ha mencionado, **dividir el grupo**.

Esto funcionó muy bien. Ya que al **equipo Scrum se le dio más tiempo para trabajar proactivamente** fueron capaces al fin de **estabilizar el sistema**. Mientras tanto, el **equipo apagafuegos se olvidó por completo del concepto de ser capaces de planificar su trabajo, funcionaban de forma completamente reactiva**, arreglando cualquier emergencia que surgiera a continuación.

Por supuesto, el **equipo Scrum no estuvo completamente aislado**. Frecuentemente, el **equipo apagafuegos tuvo** que involucrar a personas clave del **equipo Scrum**, o en el peor de los casos a todo el equipo.

En cualquier caso, **después de un par de meses el sistema era suficientemente estable como para eliminar el equipo apagafuegos y crear nuevos equipos Scrum en su lugar**. Los apagafuegos se sintieron felices de poder colgar sus aporreos cascós y unirse a equipos Scrum en su lugar.

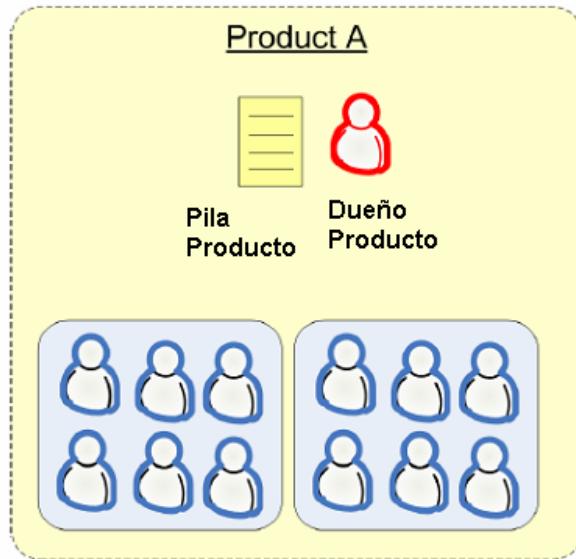
¿Dividir la Pila de Producto – o no?

Digamos que **tenemos un producto y dos equipos Scrum**. ¿**Cuántas Pilas de Producto deberías tener?** ¿**Cuántos Dueños de Producto?** Hemos evaluado **tres modelos** para ello. La elección que hagas tendrá un gran efecto en cómo se desarrollan las reuniones de Planificación de Sprint.

Estrategia 1: un Dueño de Producto, una Pila de Producto

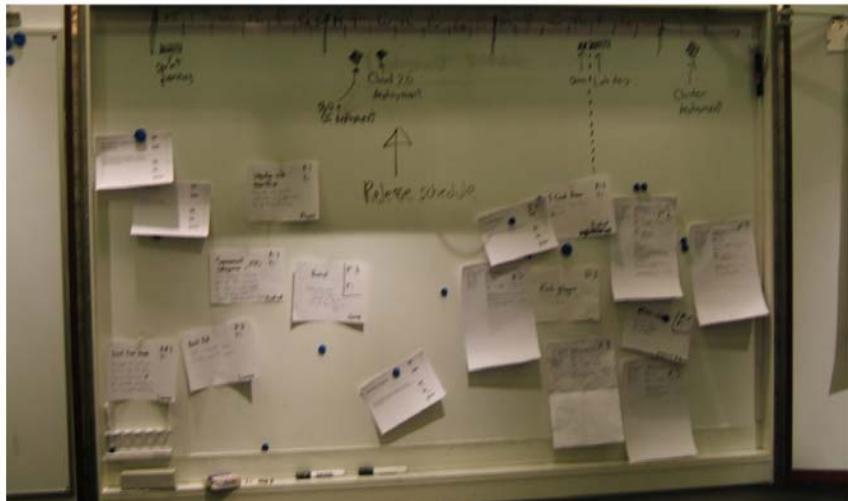
Este es el modelo “solo puede quedar uno”. Nuestro **modelo preferido**.

Lo bueno de este modelo es que puedes **permitir** a los **equipos formarse ellos mismos en función de las prioridades actuales** del Dueño de Producto. El Dueño de Producto **puede centrarse en lo que él necesita**, y dejar a los **equipos que decidan como dividir el trabajo**.



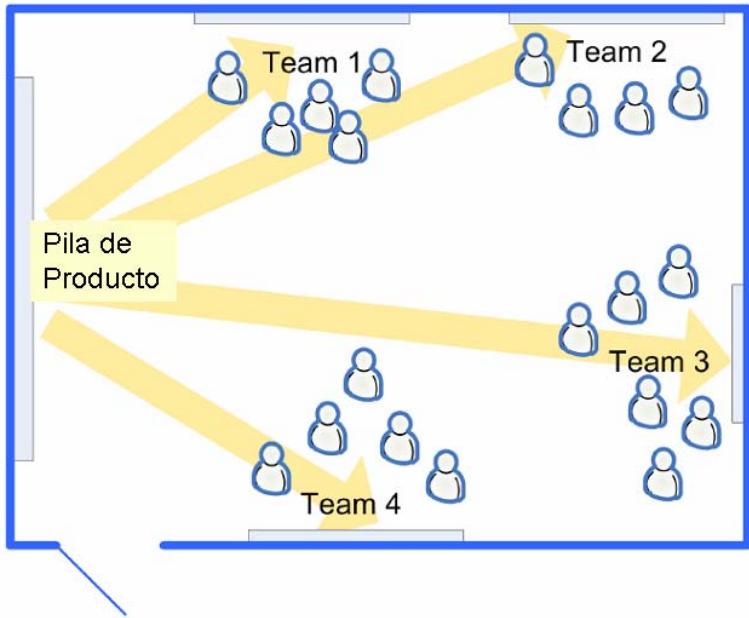
Para ser más concretos, he aquí como funcionaría la reunión de planificación de Sprint para este equipo:

Esta reunión de planificación de Sprint tiene lugar en un centro de conferencias externo. Justo antes de la reunión, el Dueño de Producto designa una pared como la "pared de Pila de Producto" y pone las historias en ella (tarjetas de cartulina), ordenadas por prioridad relativa. Continua poniéndolas hasta que la pared está llena, lo cuál usualmente es más que suficiente para un Sprint.



Cada equipo Scrum selecciona una sección de pared vacía para sí y colocan su nombre en ella. Esta será su "pared de equipo". Cada equipo, entonces, va cogiendo historias de la pared de Pila de Producto comenzando por las de máxima prioridad y las coloca en su propia pared.

Esto queda reflejado en el siguiente gráfico, en el que las flechas amarillas simbolizan el flujo de historias de la pared de Pila de Producto a las paredes de equipo.



Conforme la reunión progresaba, el Dueño de Producto y los equipos negocian con las tarjetas de cartulina, rotándolas de equipo en equipo, moviéndolas arriba y abajo para cambiar su prioridad, dividiéndolas en historias más pequeñas, etc.

Después de una hora más o menos, cada equipo tiene una primera versión de lo que podría ser su Pila de Sprint en su pared. A partir de ahí los equipos trabajan de forma independiente, estimando tiempos y dividiendo en tareas.

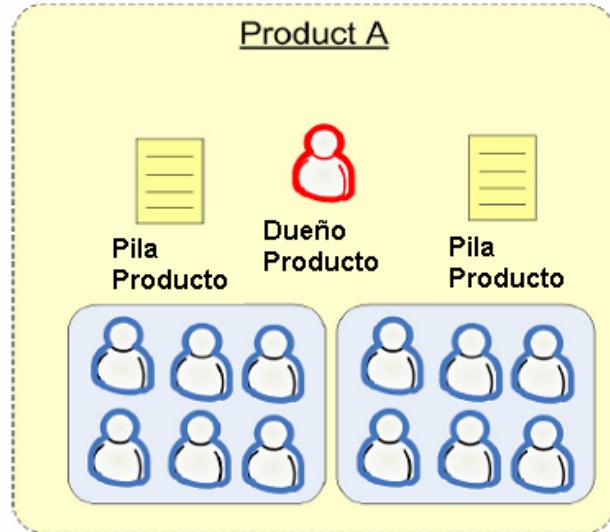


Es un follón, es caótico y agotador, pero también es muy efectivo, divertido y social. Cuando se acaba el tiempo, todos los equipos tienen usualmente suficiente información como para comenzar con su Sprint.

Estrategia 2: un Dueño de Producto, múltiples Pilas de Producto

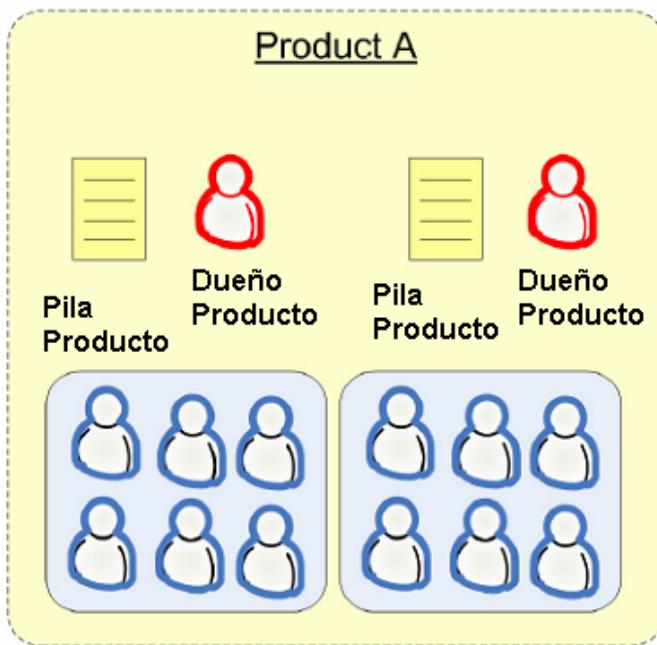
En esta estrategia, el Dueño de Producto mantiene múltiples Pilas de Producto, una por equipo. Realmente no hemos probado esta aproximación, pero hemos estado cerca. Es básicamente nuestro plan de repuesto si el primer enfoque falla.

La **debilidad** de esta estrategia es que el **Dueño de Producto** se dedica a asignar historias a los equipos, una tarea que probablemente harían mejor ellos mismos.



Estrategia 3: múltiples Dueños de Producto, una Pila de Producto por equipo.

Esta es equivalente a la segunda estrategia, una **Pila de Producto por equipo**, pero con un **Dueño de Producto** por equipo también.



No lo hemos probado, y probablemente no lo haremos nunca.

Si las dos **Pilas de Producto** emplean la misma base de código, esto probablemente causará serios conflictos de interés entre ambos **Dueños de Producto**.

Si las dos Pilas de Producto emplean distintas bases de código, sería esencialmente lo mismo que dividir el producto en dos sub-productos y gestionarlos independientemente. Eso significa que volveríamos al enfoque de un-equipo-por-producto, lo cuál es agradable y sencillo.

Ramificación del código

Con múltiples equipos trabajando en la misma base de código, inevitablemente debemos enfrentarnos a ramificaciones del código en el SCM (sistema de gestión de configuración del software). Hay muchos libros y artículos sobre cómo enfrentarse a múltiples personas trabajando en la misma base de código, así que no entrará en detalles al respecto. No tengo nada nuevo o revolucionario que añadir. Lo que sí haré, en todo caso, es resumir algunas de las lecciones más importantes aprendidas por nuestros equipos hasta ahora.

- Se estricto en mantener la línea principal (o tronco) en un estado consistente. Esto significa, como mínimo, que todo debería compilar y que todas las pruebas unitarias deberían pasarse. Debería ser posible crear una versión funcional *en cualquier momento*. Preferiblemente, el sistema de compilación continua debería compilar y auto-instalar *en un entorno de pruebas* cada noche.
- Etiqueta cada versión. Cada vez que liberes una versión para pruebas o para producción, asegúrate de que hay una etiqueta en tu línea principal identificando *exactamente qué se ha liberado*. Esto significa que, en cualquier momento del futuro, podrás volver atrás y crear *una nueva ramificación desde ese punto*.
- Crea nuevas ramas sólo cuando sea necesario. Una buena regla es ramificar nuevas líneas de código *sólo cuando no puedas utilizar una existente sin romper la política* de dicha línea. Ante la duda, no ramifiques. ¿Por qué? Porque cada nueva rama incrementa el coste de administración y la complejidad.
- Usa ramificaciones principalmente para separar *diferentes ciclos de vida*. Puede que hayas decidido que cada equipo Scrum programe su propia línea de código o no. Pero si mezclas correcciones a corto plazo con cambios a largo en la misma línea de código, encontrarás *muy difícil liberar las correcciones* a corto plazo.
- Sincroniza frecuentemente. Si estás trabajando en una rama, sincroniza *con la línea principal* cada vez que tengas algo que compile. Cada día, cuando *llegues al trabajo*, sincroniza *desde la línea principal* a tu rama, de forma que tu rama esté *al día* respecto a *los cambios de otros equipos*. Si esto te causa un infierno de fusión de código, simplemente acepta que habría sido muchísimo peor esperar.

Retrospectivas multi-equipo

¿Cómo hacemos las retrospectivas cuando tenemos múltiples equipos trabajando en el mismo producto?

Inmediatamente *después de la demo de Sprint, tras el aplauso y la charla*, cada equipo va a su sala, o a algún sitio cómodo fuera de la oficina. Hacen su *retrospectiva* más o menos como describí en el apartado “cómo hacemos retrospectivas de Sprint”.

Durante la reunión de planificación de Sprint (a la que todos los equipos asisten, ya que hacemos Sprints sincronizados para cada producto), la primera cosa que hacemos es dejar que un portavoz de cada equipo se levante y resuma los puntos clave de su retrospectiva. Lleva unos cinco minutos por equipo. Entonces tenemos una discusión abierta durante unos 10 – 20 minutos. A continuación hacemos un descanso. Y después comenzamos con la planificación del Sprint propiamente dicha.

No hemos intentado otras cosas para múltiples equipos, esto funciona suficientemente bien. La principal desventaja es que no hay un tiempo de descanso justo después de la retrospectiva y antes de la planificación de Sprint (ver “descansos entre Sprints”).

Para productos de un solo equipo, no hacemos resumen de retrospectiva durante la planificación de Sprint. No hace falta, ya que todo el mundo estuvo en la reunión de retrospectiva.

16

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Cómo gestionamos equipos distribuidos geográficamente

¿Qué pasa cuando los miembros de los equipos están en localizaciones geográficas diferentes?

Mucha de la “magia” de Scrum y XP se basa en tener miembros de equipos co-alocados, colaborando estrechamente, programando por parejas y encontrándose cara a cara todos los días.

Tenemos algunos equipos que están geográficamente separados, y también tenemos a miembros de equipo que trabajan desde casa de vez en cuando.

Nuestra estrategia es muy simple. Usamos todos los trucos que podemos para maximizar el “ancho de banda” de la comunicación entre los miembros de equipo separados. No me refiero solo al ancho de banda en Megabits por segundo (aunque esto también es muy importante). Me refiero al ancho de banda de la comunicación en un sentido más amplio:

- La posibilidad de programar por parejas juntos
- La posibilidad de encontrarse cara a cara en el Scrum diario
- La posibilidad de tener discusiones cara a cara en cualquier momento
- La posibilidad de encontrarse físicamente y socializar
- La posibilidad de tener reuniones espontáneas de todo el equipo
- La posibilidad de tener la misma visión de la Pila de Sprint, burn-down de Sprint, Pila de Producto y otras fuentes de información.

Algunas de estas medidas que hemos implementado (o estamos implementando, todavía no las hemos hecho todas) son:

- Webcam y auriculares en cada puesto de trabajo
- Salas de reuniones “habilitadas para trabajo remoto” con webcams, micrófonos de conferencia, ordenadores siempre encendidos / siempre listos, software de compartición de escritorio, etc.
- “Ventanas remotas”. Grandes pantallas en cada localización, mostrando una vista permanente de la otra. Una especie de ventana virtual entre dos departamentos. Puedes estar de pie frente a ellas y saludar con la mano. Puedes ver quién está en su mesa y quién habla con quién. Esto crea una sensación de “hey, estamos en esto juntos”.
- Programas de intercambio, donde personas de cada localización viajan y se visitan unos a otros regularmente.

Utilizando estas técnicas y algunas otras, estamos cogiendo el tranquillo, lenta pero seguramente, de cómo hacer reuniones de planificación de Sprint, demos, retrospectivas, Scrum diarios, etc., con miembros de equipos distribuidos geográficamente.

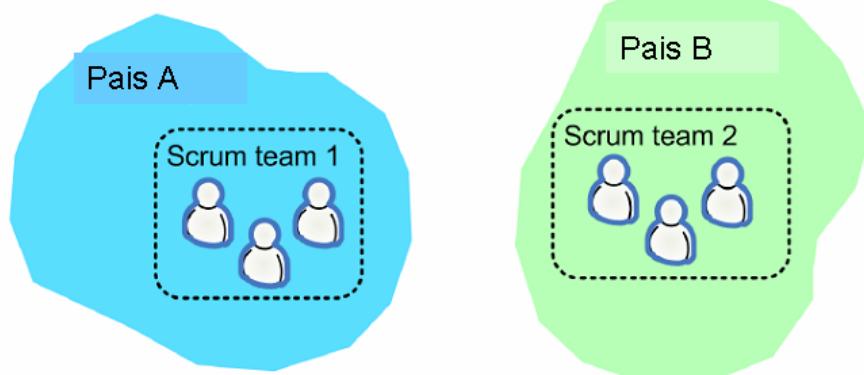
Como siempre, todo consiste en experimentar. Inspeccionar => adaptar => inspeccionar => adaptar => inspeccionar => adaptar => inspeccionar => adaptar => inspeccionar => adaptar

Offshoring

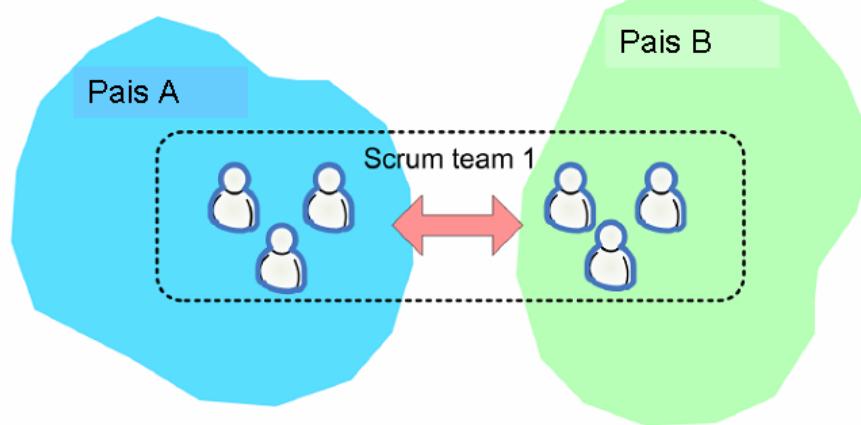
Tenemos varios equipos offshore (en el extranjero) y hemos estado experimentando cómo gestionar esto eficientemente utilizando Scrum

Hay un par de estrategias principales: **equipos separados** o **miembros de equipo separados**.

Equipos Separados



Miembros de equipo separados



La primera estrategia, **equipos separados**, es una opción convincente. Sin embargo, hemos comenzado con la **segunda opción, miembros de equipo separados**. Hay **varias razones para ello**.

1. Queremos que los miembros de cada equipo se conozcan bien entre sí.
2. Queremos tener una infraestructura de comunicaciones excelente entre las dos localizaciones, y queremos dar a los equipos un fuerte incentivo para conseguirlo.

3. Al principio, el equipo offshore es demasiado pequeño como para formar un equipo Scrum por si solo.
4. Queremos un periodo de intercambio intensivo de conocimiento antes de que los equipos independientes offshore sean una opción viable.

A largo plazo puede que pasemos al enfoque "equipos separados".

Miembros de equipo que trabajan desde casa

Trabajar desde casa puede ser realmente bueno de vez en cuando. A veces puedes hacer más programación en un día desde casa que en toda una semana en la oficina. Al menos si no tienes niños :o)

Y aun así uno de los fundamentos de Scrum es que todo el equipo debería ser colocado físicamente juntos. Así que ¿qué hacemos?

Básicamente dejamos que sean los equipos los que decidan cuándo y con qué frecuencia es bueno trabajar desde casa. Algunos miembros de equipos trabajan desde casa regularmente debido a que pierden mucho tiempo en el itinerario al trabajo. No obstante, animamos a los equipos a que estén juntos "la mayor parte" del tiempo.

Cuando los miembros de equipo trabajan desde casa, se unen al Scrum diario utilizando una llamada de voz con Skype (a veces incluso con video). Se mantienen conectados a la mensajería instantánea todo el día. No es lo mismo que estar en la misma sala, pero es suficientemente bueno.

Una vez probamos el concepto de utilizar los miércoles como *día de concentración*. Esto básicamente significaba "si quieres trabajar desde casa, perfecto, pero hazlo los miércoles. Y acuérdate con tu equipo". Esto funcionó muy bien con el equipo en que lo probamos. Normalmente la mayor parte del equipo se quedaba en casa los miércoles y conseguían terminar muchísimas cosas, y aun así colaboraban juntos bastante bien. Ya que duraba un solo día, no llegaban a perder demasiada sincronía entre ellos. Por alguna razón, sin embargo, esto nunca llegó a prender en otros equipos.

En general, tener a gente trabajando desde casa no ha sido un problema para nosotros.

17

Versión gratuita on-line.

Apoya este trabajo, compra la copia impresa:
<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Lista de comprobación del Scrum Master

En este capítulo final mostraré nuestra “lista de comprobación”, que enumera la mayoría de las **rutinas administrativas de nuestros Scrum Masters**. Cosas que es **fácil olvidar**. Nos saltaremos las **evidentes** como “**eliminar impedimentos del equipo**”.

Comienzo del Sprint

- Tras la reunión de planificación de Sprint, crea una **página de información**. Añade un enlace a la página desde la página principal del Wiki. Imprime la página y ponla en la **pared por la que pase la gente cerca de tu equipo**.
- Manda un correo a todo el mundo anunciando que ha comenzado un **nuevo Sprint**. Incluye el **objetivo del Sprint** y un enlace a la **página de información del Sprint**.
- Actualiza el **documento de estadísticas de Sprint**. Añade **tu velocidad estimada**, tamaño del equipo, longitud del Sprint, etc.

Todos los días

- Asegúrate de que el **Scrum diario comienza y termina a su hora**.
- Asegúrate de que **todas las historias son añadidas o eliminadas de la Pila de Sprint cuando sea necesario para mantener el Sprint en fecha**. Asegúrate de que se notifica al **Dueño de Producto** sobre estos cambios.
- Asegúrate de que el **equipo mantiene actualizados la Pila de Producto y el burn-down**.
- Asegúrate de que los **problemas o impedimentos son solucionados o reportados al Dueño de Producto y/o el Jefe de Desarrollo**.

Final de Sprint

- Haz una **demo de Sprint abierta**.
- Todo el mundo **debería ser notificado acerca de la demo uno o dos días antes**.
- Haz una **retrospectiva de Sprint con todo el equipo** y el **Dueño de Producto**. Invita al **Jefe de Desarrollo** también, de forma que pueda **ayudar a difundir las lecciones aprendidas**.
- Actualiza el **documento de estadísticas de Sprint**. Añade **la velocidad real y los puntos clave** de la retrospectiva.

18

Version gratuita on-line.

Apoya este trabajo, compra la copia impresa:

<http://infoq.com/minibooks/scrum-xp-from-the-trenches>

Epílogo

¡Buf! Nunca pensé que sería tan largo.

Espero que este documento te de algunas ideas útiles, ya seas nuevo en Scrum o un curtido veterano.

Ya que Scrum debe ser adaptado a las necesidades específicas de cada entorno, es duro discutir de forma constructiva sobre las mejores prácticas a nivel general. Aun así, estoy interesado en vuestro feedback. Contadme cómo difiere vuestro enfoque del mío. ¡Dadme ideas para mejorar!

Sentíos invitados a escribirme a henrik.kniberg@crisp.se.

También suelo atender la dirección scrumdevelopment@yahooroups.com.

Si te gusta este libro puede que quieras echar un vistazo a mi blog de vez en cuando. Espero ir añadiendo artículos sobre Java y desarrollo Ágil de software.

<http://blog.crisp.se/henrikkniberg/>

Oh, y no lo olvides...

Es sólo un trabajo, ¿verdad?

Lecturas recomendadas

He aquí unos cuantos libros que me han proporcionado montones de ideas e inspiración. ¡Altamente recomendados!



Sobre el autor

Henrik Kniberg (henrik.kniberg@crisp.se) es un consultor de la empresa Crisp en Estocolmo (www.crisp.se), especializado en Java y desarrollo Ágil de software.

Desde que aparecieron los primeros libros sobre XP y el manifiesto Ágil, Henrik ha adoptado los principios Ágiles y ha intentado aprender cómo aplicarlos eficientemente en diferentes tipos de organizaciones. Como co-fundador y Director Técnico de Goyada (1998-2003) ha tenido múltiples oportunidades de experimentar con desarrollo orientado a pruebas y otras prácticas Ágiles mientras desarrollaba y gestionaba una plataforma técnica y un equipo de desarrollo de 30 personas.

A finales de 2005 Henrik fue contratado como Jefe de Desarrollo en una compañía Sueca del negocio de los juegos. La compañía se encontraba en una situación de crisis con problemas organizativos y técnicos urgentes. Utilizando Scrum y XP como herramientas, Henrik ayudó a la compañía a salir de la crisis mediante la implementación de los principios Ágiles y Lean a todos los niveles en la compañía.

Un viernes de Noviembre de 2006 Henrik estaba en casa, en la cama con fiebre, y decidió apuntar una serie de notas para si mismo sobre lo que había aprendido en el último año. Sin embargo, cuando empezó a teclear ya no pudo parar y, después de tres días de tecleo y dibujo frenéticos, las notas iniciales se convirtieron en un artículo de 80 páginas titulado "Scrum y XP desde las trincheras" que finalmente se convirtieron en este libro.

Henrik adopta un enfoque holístico y disfruta desempeñando diferentes roles como gerente, desarrollador, Scrum Master, maestro y coach. Le apasiona ayudar a compañías a desarrollar software excelente y formar equipos excelentes, tomando para ello el papel que sea necesario.

Henrik creció en Tokio y ahora vive en Estocolmo con su mujer Sophia y sus dos hijos. Es un músico activo en su tiempo libre, compone y toca el bajo y el teclado con bandas locales.

Para más información ver <http://www.crisp.se/henrik.kniberg>