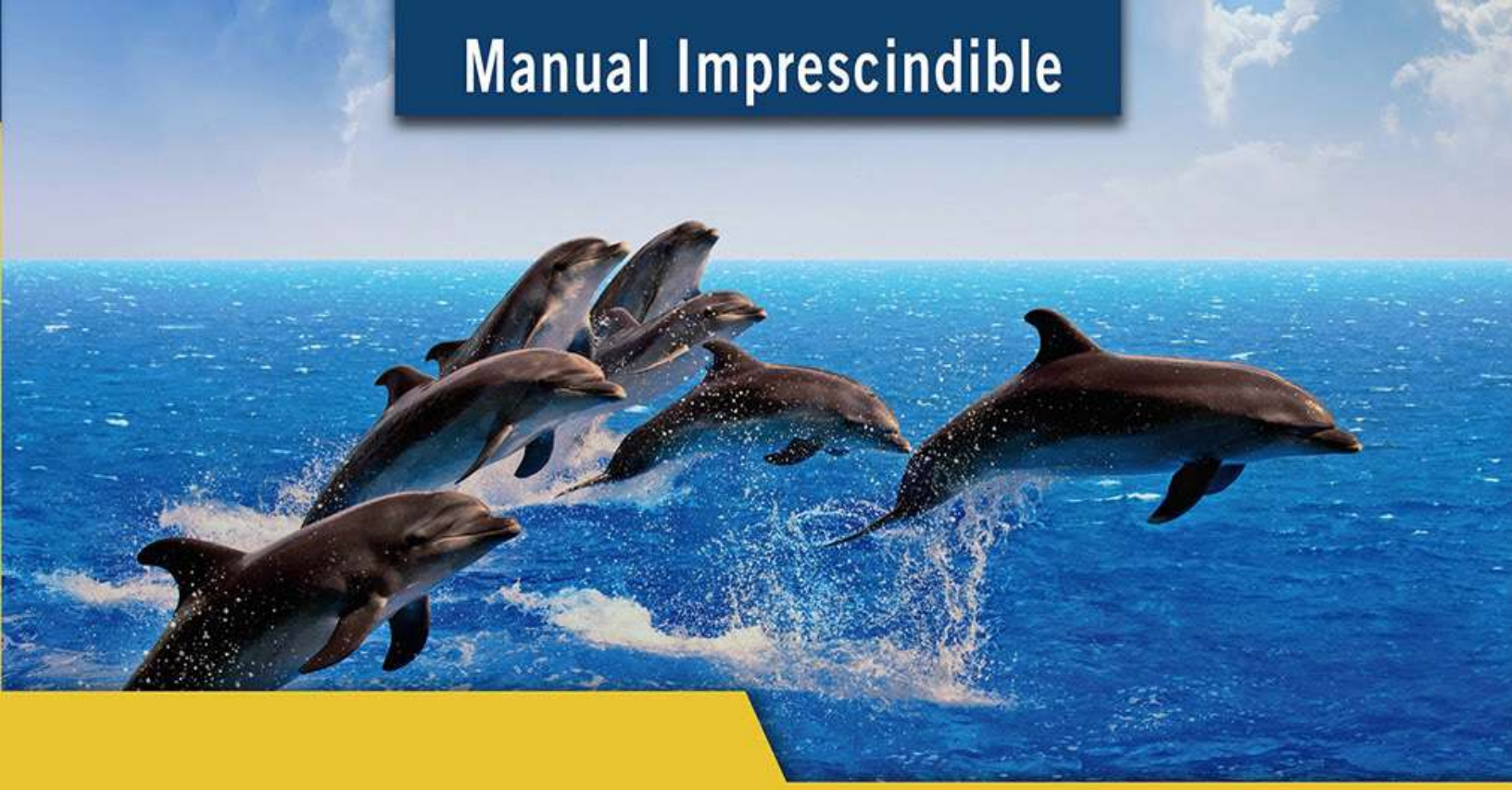
Manual Imprescindible



Métodos Ágiles
Scrum, Kanban, Lean

Prologado por Mario López de Ávila, promotor de la iniciativa "España Lean Startup" y Mike Beedle, coautor del *Manifiesto Ágil*

Carmen Lasa Gómez
Alonso Álvarez García
Rafael de las Heras del Dedo

ANAYA MULTIMEDIA

Manual Imprescindible

Métodos Ágiles
Scrum, Kanban, Lean

Carmen Lasa Gómez
Alonso Álvarez García
Rafael de las Heras del Dedo

Índice de contenidos

“A nuestras familias a las que debemos un par de Sprints”.

Agradecimientos

De Carmen Lasa

A mis “Pilares Lasa” porque contagian su alegría y porque siempre saben encontrar un rayo de luz aún en plena tormenta.

A Rafa, Manu e Irene. Siempre.

De Rafael de las Heras

A mi familia que es mi mejor equipo.

Al lado bueno de las cosas.

De Alonso Álvarez

A mis chicas y mi chico, por su apoyo y paciencia. Al aire libre y al buen cine.

A mis padres.

Sobre los autores

Carmen Lasa es licenciada en ciencias físicas y está certificada como SAFe Agilist por la Scaled Agile Academy y como *Scrum Master* y *Product Owner* por la Scrum Alliance.

Durante más de 10 años ha trabajado como *Agile coach* y se ha dedicado a la investigación, adaptación e implementación de las metodologías ágiles para la gestión de la innovación.

Ha formado en estas metodologías a numerosos equipos de Telefónica I+D y también de otros sectores como banca, televisión, seguros, sanidad y máster en dirección de proyectos.

Rafael de las Heras es ingeniero de telecomunicación y en su trayectoria laboral ha trabajado en proyectos de muy diversa temática como son la bioingeniería, ocio digital, servicios financieros e identidad. En estos campos ha desarrollado su conocimiento de las metodologías ágiles y su aplicación a diferentes entornos de trabajo, empresas y productos desempeñando diferentes roles.

Alonso Álvarez es ingeniero informático. Certificado como *Scrum Master*, *Product Owner* y *Product Manager*. Su variada experiencia va de la calidad del software a la inteligencia ambiental, o de la prospectiva tecnológica a los servicios de Video.

Creyente en la flexibilidad, la autonomía, la visión crítica, la calidad y todo lo relacionado con el agilismo, Con éste, llegan a catorce los libros de los que es autor.

Prólogos

- Prólogo de la primera edición por Mike Beedle, coautor del *Manifiesto Ágil.*
- Prólogo de la segunda edición por Mario López de Ávila, promotor de la iniciativa "España Lean Startup".

Prólogo de la primera edición

Es realmente un placer para mí escribir sobre este libro que Carmen, Rafael y Alonso han escrito sobre Scrum. Después de leer el primer capítulo, realmente no tuve otra opción que seguir leyéndolo hasta el final, pues me cautivó su profundidad que contrastaba exquisitamente con su sencillez, ¿qué mejor cumplido se le puede dar a un libro?

Este libro es necesario, no solo para nosotros los hispanoparlantes, para los que seguro este libro indudablemente será uno de nuestros preferidos en el ámbito de Scrum y Agilidad, pero también para otros que quizá quisieran algún día futuro leerlo en otros idiomas. No me extrañaría que algún día pudiéramos leer los sabios consejos que se explican y exponen en este libro en inglés o en otros idiomas. Veamos por qué.

Me provocó su franqueza a veces muy sutil, y la autoridad con la que los autores manejan los conocimientos expuestos sobre Scrum, Ágil y Lean, explicando sus relaciones, dando ejemplos y proveyendo con prácticas explicitas. Los autores usan un estilo didáctico directo y accesible que unifica el proceso de Scrum con la necesidad de conocimiento en cada uno de sus pasos. Es decir, los autores narran el libro como si fuera la historia de un proyecto, y nos dan la información necesaria para “seguir por buen camino”, en cada jornada del viaje. Esto hace que al lector se le facilite aplicar los conocimientos adquiridos “paso a paso”, en el proceso de Scrum, pues el lector aprenderá de la misma forma, un proceso idéntico o muy similar a Scrum.

En sus páginas no solo se demuestra un amplio conocimiento de Scrum, también se expone una alta experiencia en su práctica. Esto le da a este libro un valor único al lector como una herramienta práctica y concisa, para que él o ella también implementen Scrum lo mejor posible.

Los autores no solo explican “cuál es el camino correcto”, también introducen señales precaucionarias para que el lector sepa si se ha desviado inapropiadamente, y pueda hacer correcciones en su curso hacia un “mejor Scrum”.

Los autores han escogido desde un principio introducir y explicar las historias de usuario, y esto es de gran beneficio para el lector también. Yo, de hecho en mis clases públicas y privadas de CSM y de Enterprise Scrum, también he escogido introducir las historias de usuario desde un principio, pues pienso que aunque oficialmente las historias de usuario no son parte de Scrum, sí lo son en la práctica, es así como los implementadores modernos trabajamos. Es así nuevamente, como este libro hace una vez más otra contribución importante: nos introduce a prácticas modernas actuales. Esta inclusión es importante, pues hace una contribución directa a la parte estratégica de Scrum, el *Product Backlog*, que es lo que se le entrega al cliente.

Otra razón importante para leer este libro, es que fue realmente delicioso encontrar buenas referencias a otros autores de Scrum, Agilidad, y Lean que los autores seleccionaron apropiadamente, pero sin ser demasiadas para no confundir al lector. Con estas referencias, el lector podrá extender aún más sus conocimientos y entendimiento de Scrum selectivamente.

Como uno de los autores del Manifiesto Ágil, también me es muy grato leer este libro, pues verifica aún más, que Scrum y la Agilidad representan una revolución de una magnitud global, y que todos en el mundo podemos y debemos hacer contribuciones, independientemente de nuestros antecedentes culturales. Este año he viajado por unos 25 países, dando clases de Scrum y haciendo consultoría generalmente para empresas grandes, y mi experiencia corrobora que las prácticas, retos, y problemas por resolver son de índole global, pues son compartidas por todos los practicantes de Scrum en el mundo. Así, la sabiduría en este libro puede ser usada por cualquier practicante en cualquier lugar del mundo. Ojalá muchos tengan la oportunidad de leerlo y empaparse de los conocimientos y experiencias que aquí se comparten.

Por último, Scrum y Ágil, se aplican más que al desarrollo de sistemas, a la mercadotecnia, estrategia corporativa, desarrollo de productos, desarrollo básico, rediseño de procesos, etc. cualquier proceso que necesite un proceso de manejo empírico como lo es Scrum; y este libro nos dota del conocimiento necesario tanto en el proceso en general, como en las prácticas y experiencias, para su posible aplicación a otros procesos.

Realmente, les felicito: han escrito una excelente introducción de Scrum, que sería práctica, sencilla, útil pero profunda en cualquier idioma. Enhorabuena.

Mike Beedle
Lake Forest, IL

Prólogo de la segunda edición

Es un mundo VUCA

El 26 de octubre de 2015 el presidente de la tercera empresa de procesados cárnicos de este país comenzó el día con una mala noticia. La Organización Mundial de la Salud (WHO), haciendo propias las recomendaciones de la Agencia Internacional para la Investigación sobre el Cáncer (IARC), comunicaba al mundo que las carnes procesadas, como las hamburguesas, los embutidos o el bacon, de acuerdo con la evidencia científica disponible, causaban cáncer.

Póngase por un momento el lector en la situación. ¿Cómo se siente? ¿Sorprendido? Seguramente el término se quede corto. Haga un repaso de algunos de los principales hitos de los dos últimos años: la crisis de las emisiones en Volkswagen; el Brexit o las elecciones USA de 2016… el mundo es cada vez más volátil, pareciera que cualquier cosa pudiese pasar de un momento a otro.

El mundo es, también, cada vez más incierto. Disponemos de más datos y de más capacidad de cálculo y de inteligencia que en toda la historia de la humanidad, pero tratar de anticipar qué va a ocurrir es cada vez más difícil. En parte es así porque el mundo es, también, cada vez más complejo. Más agentes, a menudo con intereses enfrentados, conviven en un mundo hiperconectado en el que cada vez es más difícil distinguir una oportunidad de una amenaza, al “amigo” del enemigo”. Ante este panorama, no es de extrañar que la vida de las organizaciones sea cada vez más corta. Se dice que una empresa fundada en los años 60 podía esperar mantenerse en el mercado durante más de medio siglo. Una empresa fundada hoy tiene una ‘esperanza de vida’ de algo más de una década.

Preguntados por este hecho, un tercio de los lectores participantes en una encuesta del prestigioso semanario británico **The Economist** realizada en 2009 reconocían ser conscientes de que sus organizaciones se encontraban en grave riesgo como consecuencia de su incapacidad para manejarse en un mundo volátil, incierto, complejo y ambiguo como éste. Un mundo VUCA, de acuerdo con el acrónimo en inglés, de origen militar pero rápidamente adoptado por gurús del Management, sociólogos y tertulianos en los últimos años.

¿Cómo se sobrevive en un mundo así? Y, dado que la supervivencia no es suficiente, ¿Cómo prosperar? La mitad de los CEOs participantes en la encuesta consideraban imprescindible para la competitividad de su empresa ganar una mayor flexibilidad en la toma de decisiones y en la ejecución de las iniciativas estratégicas. En otras palabras, el 90% de los encuestados concluían que el factor clave para supervivencia era ser más ágiles.

Aprender más, más rápido

¿Cómo debemos entender “Agilidad” en este contexto? A menudo tiende a confundirse con rapidez, pero la velocidad ‘cruda’ no es la característica definitoria de la Agilidad, sino más bien la capacidad de adaptarse al cambio o, si lo prefieren, la velocidad, sí, pero a la hora de responder a las demandas del entorno. Ser capaz de identificar más y mejores oportunidades, responder antes y de manera más eficaz y apalancarse en los resultados de la experiencia para mejorar de forma continua son características destacadas de las organizaciones ágiles. Adaptación y resiliencia son aspectos definitorios de la Agilidad Organizativa.

Una organización ágil se adapta más rápido y mejor en un mundo VUCA porque es capaz de aprender más, más rápido. Aprende más y más rápido sobre el contexto, los agentes implicados y sobre sus propias capacidades. Este aprendizaje acelerado se consigue a través de prácticas como: equipos auto-organizados; trabajo por iteraciones e incrementos; contacto frecuente y temprano con todas las partes interesadas; realización de multitud de pequeños experimentos y otras muchas orientadas a ayudar a los individuos, equipos u organizaciones a despejar con rapidez las incertidumbres, empezando por las más críticas.

En las dos últimas décadas los denominados enfoques **Agile** (en inglés) se han convertido en la mejor (y durante mucho tiempo, única) concreción operativa de la Agilidad en las organizaciones. La comunidad de agilistas en todo el mundo ha creado soluciones de gestión para individuos, equipos y organizaciones que proporcionan un resultado muy superior a otros enfoques más tradicionales en situaciones de gran incertidumbre y/o complejidad. En definitiva, Agile representa una forma de pensar y actuar en y desde las organizaciones más eficaz en esta nueva realidad.

Agilidad de extremo a extremo

El catálogo de soluciones Agile a los problemas de gestión en un mundo VUCA es muy extenso, como el lector podrá apreciar. De hecho, hoy en día podemos hablar ya de una agilidad organizativa “de extremo a extremo”: desde el cliente al proveedor, desde el marketing o las ventas hasta la función de compras, pasando por las operaciones a todos los niveles, marcos metodológicos como *Lean Startup* o SAFe nos enseñan cómo actuar para conseguir los mejores resultados en un entorno cambiante. Su eficacia ha sido contrastada en, literalmente, decenas de miles de implantaciones. Y, sin embargo, la adopción de este enfoque dista mucho de ser universal.

Para entender por qué esto es así, hay que recordar que Agile nace en el entorno del desarrollo de software y ha sido visto tradicionalmente desde la Alta Dirección como algo

más táctico que estratégico. Como consecuencia de esta percepción, la adopción de Agile en comités de dirección y/o consejos de administración, precisamente donde más falta haría, es meramente testimonial. Afortunadamente, hay evidencias que apuntan a un cambio también en este sentido.

Empezando por el hecho más reciente, la aparición en portada de la *Harvard Business Review* de mayo de 2016 del artículo "**Embracing Agile**", firmado por Darrell K. Rigby, Jeff Sutherland e Hirotaka Takeuchi, es muy significativo, representando como lo hace esta revista al 'mainstream' del *Management*. Cientos de miles de directivos en todo el mundo utilizan la HBR como referente a la hora de adoptar nuevos enfoques de gestión.

El segundo hecho que nos proporciona confianza en un incremento de la adopción de enfoques Agile en el mundo corporativo es el tremendo éxito que el marco metodológico *Lean Startup* ha tenido entre grandes empresas consolidadas como TELEFÓNICA o General Electric. *Lean Startup* es un marco para el desarrollo de iniciativas innovadoras en situaciones de gran complejidad e incertidumbre. A pesar del nombre, se trata de un marco *Agile*, no Lean. Y ha sido recibido con entusiasmo por los profesionales de la innovación, del marketing y del desarrollo de negocio en todo el mundo. Podríamos decir que *Lean Startup* se ha convertido en el "caballo de Troya" del agilismo en las áreas "de negocio" de multitud de empresas.

Marcos como *Lean Startup*, en combinación con metodologías como DevOps o aplicaciones en los ámbitos de los aprovisionamientos o en los procesos de selección de recursos humanos permiten hablar con confianza de una evolución hacia organizaciones más ágiles de extremo a extremo. El libro que tiene entre manos el lector recoge una panorámica completísima y actualizada de estos marcos, métodos e instrumentos. Regale un ejemplar a su Consejero Delegado, recomiéndelo a sus jefes y compañeros de trabajo. Esté seguro de que tarde o temprano se lo agradecerán: tal vez haya salvado a su empresa de la trituradora VUCA.

Mario López de Ávila
Promotor de la iniciativa "España Lean Startup"

Introducción

Desde hace ya unos años, los métodos de trabajo ágiles ya no son una moda ni una novedad y cada vez están más extendidos. Se trata de un conjunto de herramientas fundamentales para afrontar los desafíos del trabajo en proyectos (costes y tiempo que se disparan, incertidumbres que no se solucionan) y mejorar la calidad del trabajo de las personas. Nacidos en la industria del software, son ya una forma de trabajo estandarizada en ella, pero, además, es cada día más fácil encontrarlos, especialmente sus conceptos y herramientas, en otros campos. Por ello, es necesario un texto de guía y referencia que ayude a conocer, comprender y aplicar los principios, métodos y herramientas ágiles.

Aunque existe una gran cantidad de libros y artículos, incluso algunos en español, los autores consideran que es necesario un texto que dé una visión general de los métodos más usados y, además, sea práctico para ayudar a aplicarlos. Por eso, la intención de este libro es ser una referencia didáctica más que un texto académico o especializado. Además, con él, los autores quieren compartir su experiencia de años en los que han ayudado a equipos de trabajo a aplicar métodos ágiles y a perfeccionar su modo de trabajo.

De entre todos los métodos ágiles, *Scrum* y *Kanban* son los más difundidos y aplicados en toda clase de proyectos. Además, en los últimos años, *Lean Startup* se ha consagrado como forma de trabajo en actividades de innovación, en la creación de empresas y de sus modelos de negocio. Por ese motivo, y buscando una orientación práctica, se ha centrado buena parte del contenido en mostrar la forma de aplicar estos tres métodos. Además, en este manual, se facilita al lector suficiente material para profundizar en estos métodos y en el estudio de otras variantes y aproximaciones ágiles.

Cómo usar este libro

Este libro es un manual práctico pensado para facilitar el conocimiento de los métodos ágiles y ahondar en el aprendizaje de *Scrum*, *Kanban*, *Lean* y *Lean Startup*. Por ello, contiene toda clase de notas, trucos y advertencias, además de secciones adicionales donde ampliar información. Se ha tomado como ejemplo a lo largo del texto el proyecto de la elaboración de este libro como un medio para mostrar la aplicación de *Scrum* en una actividad real (y no necesariamente orientada al desarrollo de software). El libro se estructura de la siguiente forma:

Introducción: Es este capítulo y, en él, se explica el propósito y forma de usar el contenido del libro.

Capítulo 1. Métodos ágiles: Capítulo en el que se ofrece una visión general de los principales métodos ágiles.

Primera parte: Trabajar con *Scrum*.

- **Capítulo 2. Scrum de un vistazo:** Se presenta *Scrum* de forma sintética para que pueda servir de material introductorio, de referencia y repaso.
- **Capítulo 3. Luces, cámara y Sprint 0:** Se dedica a revisar la etapa preliminar del proceso *Scrum*. Además, se presenta la figura de *Product Owner*.
- **Capítulo 4. ¿Qué tengo que hacer? El Product Backlog:** Dedicado a hablar de la importantísima tarea de poblar el repositorio de actividades del proyecto: el *Product Backlog*.
- **Capítulo 5. Antes de empezar: Sprint Planning:** Donde se presentan las actividades de preparación de cada iteración del proyecto o *Sprints*. También se presenta el rol del *Scrum Master*.
- **Capítulo 6. Manos a la obra: Desarrollo del Sprint:** Dedicado al desarrollo del trabajo en cada iteración o *Sprint*. Se habla, además, del equipo de trabajo y de las herramientas usadas.
- **Capítulo 7. ¿Vamos por buen camino? La Sprint Review:** El proceso de revisión de resultados de cada iteración.
- **Capítulo 8. ¿Cómo lo hemos hecho? La Retrospectiva:** Dedicado al que bastantes autores consideran el proceso más importante de *Scrum*: la revisión de cómo se ha realizado el *Sprint* y cómo mejorar la forma de trabajar de un equipo.
- **Capítulo 9. Scrum en acción: "Tu" Scrum:** *Scrum* no es homogéneo ni monolítico, hay variantes y adaptaciones. Pero, ojo, también se corre el riesgo de caer en una aplicación puramente nominal.

Segunda parte: Aplicación avanzada de los métodos ágiles.

- **Capítulo 10. XP. Una aplicación de los métodos ágiles al desarrollo software:** Prácticas ágiles específicas para el desarrollo de aplicaciones informáticas.
- **Capítulo 11. Scrum en la empresa:** Aplicación de las metodologías ágiles al mundo de la empresa y sus peculiaridades.
- **Capítulo 12. Escalando Scrum:** Cómo coordinar y trabajar con equipos grandes y distribuidos, cómo detectar señales de alarma y otros temas avanzados y útiles en la aplicación de *Scrum*.
- **Capítulo 13. Kanban, el otorgador de permisos:** Presentación de uno de los métodos ágiles más utilizados en el mundo del mantenimiento, el soporte y la logística. Aquí se dan algunos trucos para aplicarlo con éxito.
- **Capítulo 14. Lean Software Development:** Capítulo dedicado a comprender la visión ágil que propone *Lean* y las herramientas de *Lean Software Development*.

Tercera parte: El éxito en los proyectos.

- **Capítulo 15. Lean Startup:** Presenta *Lean Startup*, término con el que se conoce popularmente al método que ayuda a crear nuevas empresas de corte tecnológico, aunque en realidad es una forma ágil (iterativa, flexible, incremental y eficiente) de definir un modelo de negocio.
- **Capítulo 16. El lienzo de modelos de negocio:** En este capítulo aprenderá a identificar los nueve elementos fundamentales de un negocio y cómo crear un modelo.
- **Capítulo 17. Especificaciones ejecutables:** Aquí aprenderá cómo aplicar las especificaciones funcionales en el marco de *Scrum*.

Glosario: Definición de los principales términos empleados en el libro.
Referencias: Enlaces y bibliografía para ampliar información sobre *Scrum* y los métodos ágiles.
Índice alfabético: Para ubicar con rapidez en qué puntos del texto se trata un tema determinado.

Referentes

Además de los libros, artículos y páginas Web que aparecen en el apartado de referencias al final del libro, a lo largo del texto se mencionan determinadas personas. Se trata de autores muy reconocidos en el mundo de los métodos ágiles. Una forma muy recomendada para ampliar información sobre estos métodos es buscar las publicaciones de estos autores. Para facilitar su identificación cuando se haga mención de ellos, se incluyen algunos datos biográficos:

- **Mike Cohn:** Es el fundador de la Agile Alliance y de la *Scrum* Alliance y miembro de la actual junta directiva. Es fundador de la empresa Mountain Goat Software[1], consultora especializada en procesos, formación y gestión de proyectos. Mike Cohn tiene más de 20 años de experiencia ayudando en la adopción de las técnicas ágiles a numerosas empresas como Google, Microsoft, Phillips o Siemens.

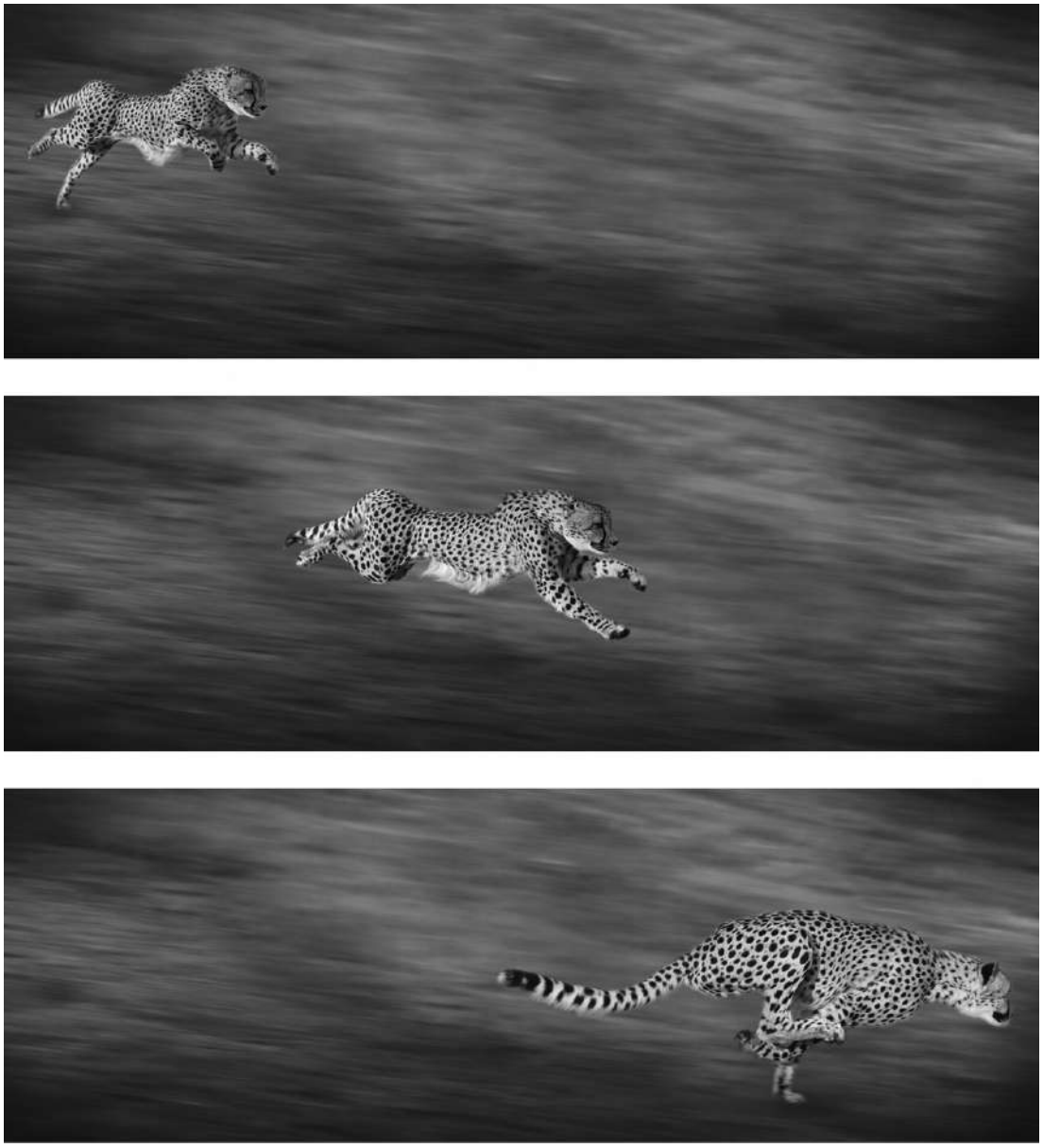


Figura I.2. Agilidad supone también velocidad y adaptabilidad.

- **Kent Beck:** Es uno de los firmantes del *Agile Manifesto* de 2001. Fundador de Three Rivers Institute (TRI) y pionero en la adopción y difusión de la Programación Extrema (XP).
- **Henrik Kniberg:** Miembro de la junta directiva de la Agile Alliance y reputado Coach en métodos ágiles. Durante los 10 últimos años, ha sido CTO de tres compañías IT en Suecia y ha ayudado a muchas otras a adoptar los métodos ágiles tanto para la gestión como para el desarrollo de software.
- **Jeff Shutherland:** Es uno de los firmantes del manifiesto *Ágil* e ideó *Scrum* como un proceso formal en 1995 junto con Ken Schawber. Ha sido presidente de la *Scrum* Foundation y actualmente es CEO de Scrum Inc. Posee una amplia experiencia como Agile Coach ayudando con éxito en la adopción de *Scrum* en empresas como Google,

Yahoo, Microsoft, IBM, Oracle, Adobe, Siemens, Sony/Ericsson, o Accenture, entre otras. También es considerado un experto en la aplicación de *Scrum* en equipos distribuidos y externalizados.

- **Mike Beedle:** Es uno de los firmantes del manifiesto *Ágil* y autor, junto con Ken Schwaber, del primer libro escrito sobre *Scrum*, *Agile Software Development with Scrum*. Es un experto en la aplicación de *Scrum* en la empresa. Como coach reconocido, ha ayudado a aplicar *Scrum* en diferentes entornos empresariales dedicados al Software comercial, banca, compañías de seguros e incluso en grandes organizaciones como el Ministerio de Defensa estadounidense.

Además, se ha creado una comunidad muy activa de personas interesadas en aplicar y difundir los métodos ágiles. Una buena forma de introducirse en ellos, de conocer y compartir de las experiencias de otros es entrar en contacto con organizaciones y foros, como la *Agile Alliance*[2] internacional, o *Agile-Spain*[3] en España. Ambos celebran conferencias muy recomendables.

Más allá de este libro

Los métodos ágiles son un campo muy extenso, que, además, no deja de crecer. El número de libros y artículos publicado cada año lo demuestra, así como la activa comunidad de usuarios y las conferencias y encuentros. Por ello, un libro como este no puede limitarse a estas pocas páginas.

Los autores quieren mantener un contacto con sus lectores y ayudar a la difusión de estas técnicas aprovechando los medios que la tecnología pone a nuestro alcance. Se puede contactar con ellos en Facebook, siguiendo la página .

Los autores hemos puesto todo nuestro esfuerzo e interés en hacer de este libro una obra práctica y útil. Esperamos haber podido alcanzar ese objetivo. Pero la verdadera utilidad de lo que aquí se enseña se obtendrá con la práctica y la dedicación. El primer encuentro con un método como *Scrum* puede ser menos satisfactorio de lo que inicialmente se espera, pero nuestra experiencia nos demuestra que la perseverancia se premia en los métodos ágiles, y ese premio se traduce en una forma de trabajar más satisfactoria, eficiente y productiva, beneficiando tanto al equipo que la aplica como a los clientes para los que trabaja.

---

1 http://www.mountaingoatsoftware.com

2 http://www.agilealliance.org/

3 http://www.agile-spain.org/

1 Métodos ágiles

En este capítulo aprenderá a:

- Comprender el cambio de paradigma que proponen los métodos ágiles.
- Conocer los fundamentos de *Lean Software Development*, Kanban, *Scrum* y *Lean Startup*.
- Descubrir la filosofía de *Pragmatic Programming*, *Dynamic Systems Development Method*, *Feature Driven Development* y *XP*.

Agile en la práctica

La aparición de las metodologías ágiles es una reacción a la falta de respuesta a los problemas históricos del desarrollo de proyectos. La incertidumbre es uno de los grandes desafíos en el desarrollo de proyectos y se ha tratado de combatir con más control sobre el proceso, planificando pormenorizadamente, estimando y diseñando cada paso. Lamentablemente, los proyectos siguen terminando fuera de plazos y coste y sin mejoras apreciables en la calidad.

En el mundo del desarrollo de software, esta situación ya se identificó en 1968 bajo el nombre de la "crisis del software". Para solucionarla, se han sucedido las metodologías, técnicas y herramientas para construir productos con calidad, dentro de plazos y costes prefijados. La solución tradicional ha sido incrementar el control: contar con requisitos detallados desde el primer momento, usar técnicas para valorar la complejidad del trabajo y estimar el esfuerzo necesario, contar con herramientas que controlaran el proceso y midieran la calidad. Sin embargo, esta aproximación falla por la base: apenas hay proyectos software donde los requisitos sean completos y estables antes de empezar el trabajo; las estimaciones siguen llenas de errores; las herramientas no han mejorado la contribución de las personas al trabajo.

En 1986, Takeuchi y Nonaka describieron una forma de trabajo en la que un equipo de trabajo transversal aborda las distintas fases de la misma forma que los jugadores de rugby afrontan una melé (*Scrum* en inglés): empujando con decisión y al unísono.

Esta idea se trasladó a la industria del software y, en 1995, se presentaron artículos que describían ya dos aproximaciones separadas pero semejantes para abordar proyectos *Scrum*. En paralelo, se estaba definiendo una nueva forma de desarrollar software llamada programación extrema o XP (*eXtreme Programming*). La gran diferencia con respecto a la forma de trabajar anterior es que se renuncia a los requisitos completos y estables de partida: se trata de convivir con la incertidumbre y el cambio. Con XP, los requisitos pueden variar y el proceso se adapta a trabajar con esa variabilidad. En lugar de rechazar los cambios, XP los considera como algo natural y saludable. En lugar de estimar como un modo de controlar con precisión el proceso, se abraza la incertidumbre.

Ser capaces de trabajar con un entorno cambiante e incierto es precisamente la mejor forma de adaptarse a las formidables revoluciones tecnológicas vividas en los últimos años, definidas por la velocidad y los cambios constantes y radicales.

El manifiesto Ágil

Todo este caldo de cultivo fermentó en el manifiesto *Ágil*, publicado en 2001. Un conjunto de autores y personas relevantes del mundo del desarrollo software se reunieron para plasmar en unos pocos puntos, las ideas y el sentir general de la industria. Aunque el manifiesto surgió como reacción a la forma de desarrollar proyectos software, con el tiempo ha trascendido este propósito y se ha convertido en la piedra fundacional de una nueva forma de trabajar. Esta forma "ágil" de construir proyectos se fundamenta en cuatro puntos:

- **Valorar a individuos y sus iteraciones frente procesos y herramientas:** Aunque todas las ayudas para desarrollar un trabajo son importantes, nada sustituye a las personas, a las que hay que dar toda la importancia y poner en primer plano. Pero esto no quiere decir que deba trabajarse sin la ayuda de procesos.
- **Valorar más el software (producto) que funciona, que una documentación exhaustiva:** Para poner las cosas en su sitio, porque se estaba dando tanta importancia a documentar el trabajo como al propio objeto del proyecto: el producto. Es un gran error restar atención a lo que queremos construir, olvidando que todo lo demás es secundario. Esto no quiere decir que no haya que documentar. La propuesta es documentar solo lo imprescindible y de forma incremental en paralelo a la construcción del producto.
- **Valorar más la colaboración con el cliente que la negociación de un contrato:** La forma más productiva de sacar adelante un trabajo es establecer un marco de confianza y colaboración con quien nos lo encarga. Sin embargo, se estaba poniendo el foco en cerrar un contrato atado que sirviera ante todo como una herramienta de protección, como si cliente y equipo fueran dos partes enfrentadas, cuando en realidad tienen objetivos e intereses comunes. Como es lógico, esto no quiere decir que no se deban establecer unas bases formales de entendimiento.
- **Valorar más la respuesta al cambio que el seguimiento de un plan:** Se trata de apreciar la incertidumbre como una componente básica del trabajo, por lo que la adaptación y la flexibilidad se convierten en virtudes y no en amenazas. El seguimiento ciego de un plan lleva, salvo contadas excepciones, al fracaso si no se puede corregir la dirección ante los inevitables cambios que van surgiendo. Pero esto no quiere decir que se trabaje sin un plan.

Figura 1.1. Portada de la página Web del manifiesto Ágil.

Aunque pueda parecer a simple vista que se trata de unos puntos aceptables y de sentido común, en realidad marcan una clara frontera con la forma de trabajar tradicional en proyectos y contienen los ingredientes de la receta necesaria para trabajar en el entorno acelerado e inestable de la tecnología y empresa actuales.

Estos cuatro puntos están acompañados de los siguientes principios, que acaban de definir la forma de trabajar de acuerdo con los métodos ágiles:

- La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software (léase “productos”) con valor.
- Aceptar que los requisitos cambian, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventajas competitivas al cliente.
- Entregar software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible (este principio requiere de adaptaciones para poderse aplicar fuera del mundo del software).
- Los responsables de negocio y los desarrolladores (miembros del equipo) trabajan juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

- El software funcionando (producto cerrado) es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para, a continuación, ajustar y perfeccionar su comportamiento en consecuencia.

Todo lo anterior se puede resumir en buscar: orientación al cliente y al producto, flexibilidad, dar relevancia a las personas y a la comunicación, colaboración, velocidad, auto-organización, calidad, simplicidad y mejora continua.

El manifiesto *Ágil* recoge una serie de ideas que describen una nueva forma de trabajar, dejando atrás la orientación inflexible y determinista de los proyectos convencionales. Ahora la incertidumbre no es una amenaza, es una dimensión en la que nos basamos y que aceptamos: hay que sacar lo mejor de ella en vez de combatirla. Esas ideas, valores y principios deben organizarse para no ser solo una colección de buenas intenciones.

¿Cómo hacer realidad el pensamiento ágil?, ¿cómo llevar a la práctica los valores ágiles?, ¿cómo velar para que los principios *agile* se respeten?

Las metodologías tradicionales proponen reglas inclusivas, es decir, proporcionan unas pautas de actuación que indican con detalle qué es lo que hay que hacer en cada momento del proyecto y en cada una de las situaciones posibles. Dependiendo del tipo de proyecto, una metodología de este tipo puede ayudar a organizarse y aportar mucho valor. Es lo que ocurre cuando el cliente tiene claros todos los requisitos y las necesidades que quiere cubrir al inicio de su proyecto y tiene la certeza de que no va a necesitar que se realice ningún cambio ni en los requisitos ni en el alcance de los mismos. En estos casos, se podrá calcular con bastante precisión el coste del proyecto y la duración del mismo. En definitiva, podremos organizarnos sin miedo a correr demasiados riesgos.

Un enfoque tradicional propone fijar los requisitos con un alto nivel de detalle al inicio del proyecto y, a partir de ellos, se hace una estimación del coste y de la fecha de entrega del mismo.

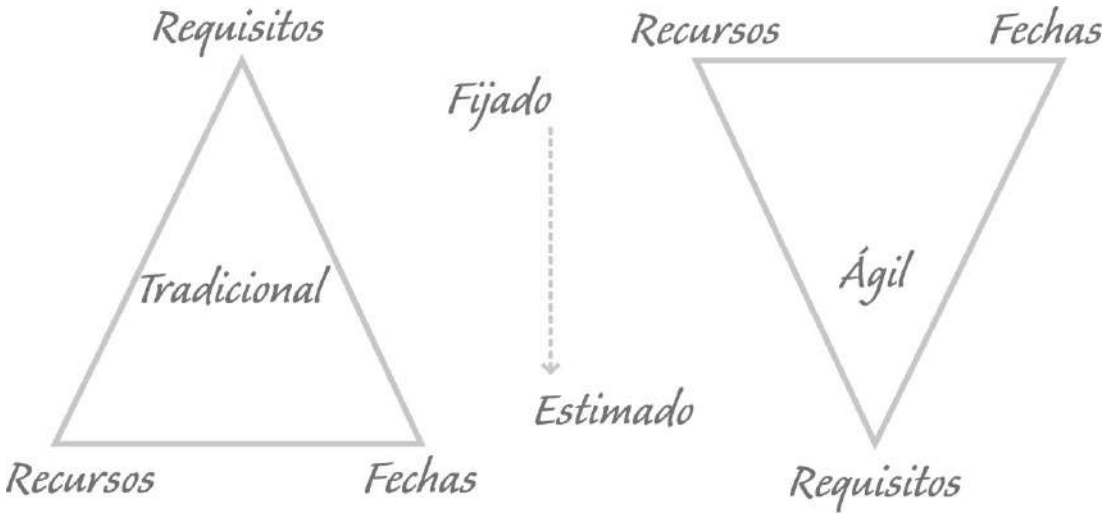


Figura 1.2. Agile propone un cambio de paradigma.

Pero ¿cuántas veces ocurre esto en la realidad? ¿En cuántos proyectos no se incorporan por el camino cambios solicitados por el cliente o marcados por las necesidades del negocio del momento?

Los métodos ágiles proponen un cambio de paradigma: partir de un presupuesto y unas fechas de entrega, y, a partir de ahí, se trabaja para implementar la funcionalidad más valiosa para el cliente en cada momento. Trabajando de esta manera, el alcance será flexible. Pero es que, para ser competitivos, la mayoría de las veces hay que adaptar el producto a medida que se va construyendo.

Lejos de huir de los cambios, los métodos ágiles sugieren formas de construir un producto que acepten esos cambios. Los métodos ágiles ofrecen reglas generativas, es decir, favorecen el que se creen reglas nuevas en el caso en que fuera necesario. Esto quiere decir que se abre la puerta a adoptar nuevas prácticas útiles para el equipo y el producto, en lugar de mantener inflexiblemente las reglas definidas al principio.

Breve introducción a algunos de los métodos ágiles

Los métodos ágiles tienen un filosofía y principios comunes con ciertos aspectos concretos que los diferencian. La idea es que en cada situación se elija el método que mejor se adapte al proyecto que se quiere abordar.

Pero ¿qué hace que un método sea ágil?, es decir, ¿qué es lo que tienen en común estos métodos? El manifiesto *Ágil* precisa esas características definitorias.

Todos ellos consideran la colaboración un elemento clave. Tanto las personas que están construyendo el producto como el cliente deben trabajar en constante comunicación y sentirse miembros de un gran equipo. No son un grupo de personas que simplemente trabajan

próximas físicamente, sino que forman parte de un gran equipo con objetivos comunes. Con este enfoque, la comunicación constante y a todos los niveles es crucial para crear el producto con una calidad excelente y que cumpla exactamente las necesidades del cliente, evitando sorpresas a todos los implicados.

Por otro lado, un método es ágil sí permite construir un producto de forma incremental, es decir, crear algo muy sencillo inicialmente y que vaya siendo enriquecido y completado de forma progresiva. No se construirán trozos de producto por separado que luego tendremos que hacer encajar al final como un rompecabezas, sino que se construye contemplando la totalidad desde el principio.

Otro factor común de estos métodos ágiles es su sencillez. Sus reglas son sencillas y de sentido común, pero, eso sí, es necesaria la experiencia y profesionalidad para obtener el máximo beneficio de ellas.

Nota:

Para que un método pueda considerarse ágil, debe ser adaptativo, es decir, debe considerar siempre la posibilidad de introducir modificaciones y cambios en cualquier etapa.

Existen métodos ágiles de proceso o gestión como son **Scrum** o Kanban. En el caso de que el proyecto sea un desarrollo de software, una buena gestión no es suficiente y se necesita también seguir unas buenas prácticas de programación. Eso permitirá una gestión optimizada, a la vez que se crea un software de calidad. Es aquí donde entran en juego los métodos ágiles de programación, como, por ejemplo, la Programación eXtrema (XP). En definitiva, para crear un buen producto software, necesitamos una combinación de mejores prácticas de gestión con mejores prácticas de programación, ambas compartiendo los valores y principios ágiles.

A continuación, se describen brevemente algunos de los métodos ágiles más extendidos. Para profundizar más, en el apartado de referencias hay bibliografía y publicaciones más especializadas.

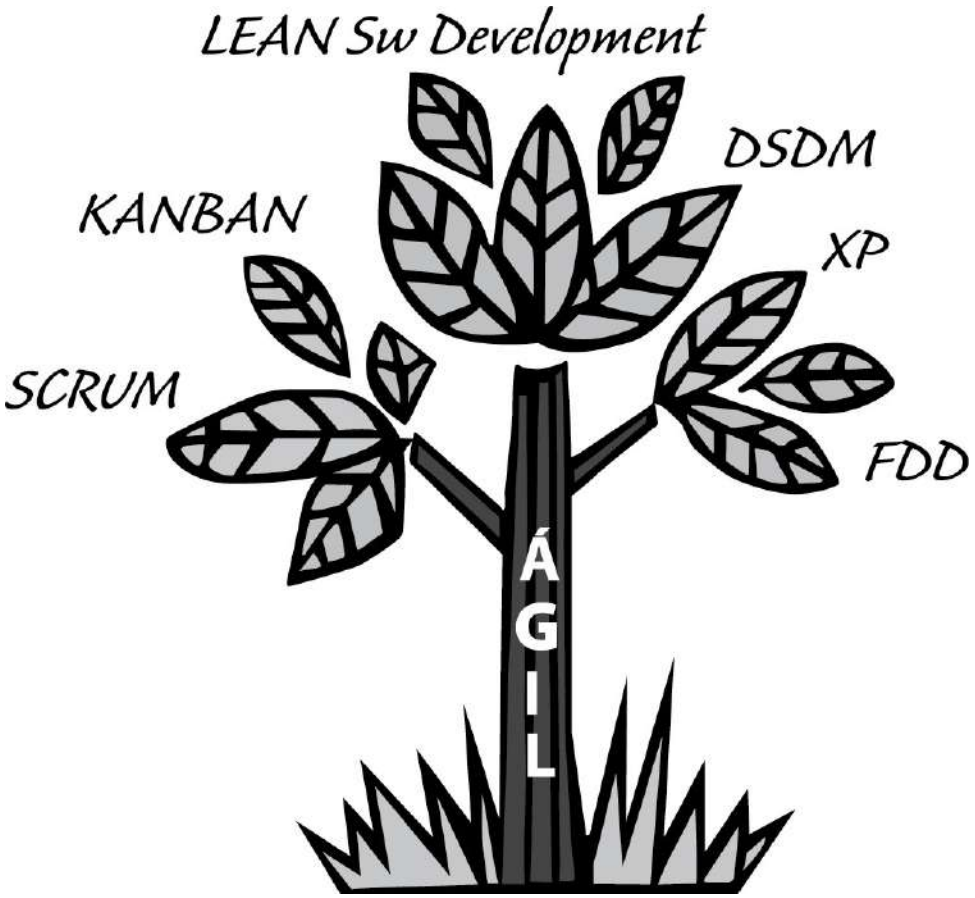


Figura 1.3. Métodos ágiles.

Lean Software Development

Lean Software Development no es tanto un método ágil como una traslación al mundo del software de los principios de los métodos *Lean* procedentes del mundo de la fabricación industrial. *Lean Software Development* tiene tres objetivos principales: reducir drásticamente el tiempo de entrega de un producto, reducir su precio y reducir también el número de defectos o *bugs*, es decir, mejorar la calidad.

Lean se basa en una serie de principios, que se han enumerado con distintas variantes. Esta es una lista de esos siete principios para conseguir los objetivos de *Lean Software Development*. En el capítulo 14 podrá encontrar en mayor detalle la lista original definida en 2003:

- **Eliminar desperdicio:** Todo aquello que no le aporta un beneficio al cliente y que no sea directamente valioso para él, debe ser eliminado. Si el cliente no lo necesita o no es lo que quiere en este momento, directamente no debe hacerse. Por este mismo motivo, y buscando siempre mantener el foco en lo esencial, hay que simplificar todo lo accesorio, optimizar los procesos, evitar la falta de información y las interrupciones al equipo de trabajo.

- **Optimizar el todo:** Hay que pensar desde un punto de vista global y orientado al largo plazo. La optimización de una pequeña parte del sistema puede afectar negativamente al conjunto del mismo.
- **Calidad integrada:** El producto debe construirse con una calidad óptima desde el primer momento. Deben cubrirse todo tipo de pruebas de forma que los defectos se corrijan lo antes posible con una construcción dirigida por las pruebas constantes. En el caso de un desarrollo de software, implica la realización de pruebas de integración, pruebas del producto completo, etc.
- **Aprender constantemente:** La creación debe entenderse como un experimento, de forma que se acepte el cambio con normalidad. La clave está en ir aprendiendo y entendiendo lo que se necesita a medida que se construye, ya que no podemos adivinar el futuro. Se deben hacer planes, por supuesto, pero siendo tolerantes al cambio y aprendiendo de la experiencia.
- **Reaccionar rápido:** Supone una gran ventaja sobre los competidores el implementar rápidamente y con calidad las soluciones que el cliente necesita y que le interesan en ese momento para posteriormente ir adaptando de forma rápida el producto a medida que se vayan detectando nuevas necesidades.
- **Mejora continua:** El foco de la mejora debe centrarse en las personas y en los procesos que hacen posible construir un producto y no en mejorar exclusiva y directamente el producto en sí. De esta forma, se mejorará el producto actual y el sistema estará listo para poder crear otros productos con éxito en el futuro.
- **Cuidar al equipo de trabajo:** Un equipo de trabajo debe estar motivado y esto se consigue de varias formas: proporcionando cierto grado de autonomía para poder tomar decisiones con sentido; ofreciendo a cada persona la posibilidad de aprender y mejorar de manera permanente; y, por último, haciendo que sientan que su trabajo es valioso en todo momento.

Scrum

Scrum propone un marco de trabajo que puede dar soporte a la innovación, basándose en equipos autogestionados. Con *Scrum* se pueden obtener resultados con calidad, en iteraciones cortas (entre una y cuatro semanas) llamadas *Sprints*.

Scrum es el método ágil más aplicado y con más elementos aplicables. Esto le convierte en el más apropiado para introducirse en este modo de trabajo y, por ello, se le dedica buena parte de este libro.

En este capítulo se dará simplemente una primera aproximación para poder compararla con el resto de los métodos ágiles. En la primera parte del libro podrá encontrar una explicación detallada de cómo funciona *Scrum* y cómo aplicarlo de manera práctica.

Scrum (o melé en castellano) es la jugada del deporte del rugby que se utiliza para

reincorporar al partido una pelota que se había quedado fuera de juego. En esta jugada, el equipo actúa como una unidad para desplazar a los jugadores del equipo contrario.

Los primeros en utilizar este término dentro de un contexto de desarrollo fueron Hirotaka Takeduchi e Ikujiro Nonaka[4] en el año 1986 para describir una nueva forma en la que trabajar dentro de un proceso de desarrollo acelerado, respondiendo a la necesidad de ser cada vez más competitivo. La comparación con el rugby se realiza por la similitud entre la forma de jugar en este deporte y la necesidad de que se modifique la manera de trabajar entre los equipos de desarrollo, ya que el deporte del rugby es altamente coordinado, colaborativo y reactivo.

En el año 2001, Ken Schwaber y Mike Beedle publican el primer libro sobre *Scrum*: *Agile Software Development with Scrum*[5] y, poco a poco, empieza a haber una gran aceptación de *Scrum* por numerosos equipos y se extiende esta nueva forma de trabajar.

La *Scrum* Alliance fue fundada en el año 2002 por Ken Schwaber y Mike Cohn. El objetivo de esta organización es compartir y aumentar de forma constante el conocimiento de *Scrum*, proporcionando un foro para el aprendizaje de forma interactiva. Para más información, puede visitar la página http://www.SCRUMalliance.org..

Scrum se basa en los siguientes principios:

- **Inspección y adaptación:** En *Scrum* se trabaja en iteraciones llamadas *Sprints*, que tienen una duración de entre 1 y 4 semanas. Cada iteración termina con un producto entregable (por ejemplo, software ejecutable o los planos de un edificio). Al finalizar cada iteración, este producto se muestra al cliente para que opine sobre él. A continuación, el equipo se reúne para analizar la manera en que está trabajando. Uniendo los dos puntos de vista, “el qué” se ha hecho y “el cómo” se está construyendo, se aprenderá con la experiencia y se podrá mejorar iteración tras iteración.
- **Auto-organización y colaboración:** El equipo se gestiona y organiza a sí mismo. Este nivel de libertad implica asumir una responsabilidad y un gran nivel de compromiso por parte de todos. Esta auto-organización funcionará siempre que exista una alta colaboración y espíritu de equipo. Los líderes y clientes colaborarán igualmente con el equipo de desarrollo en todo momento, facilitando su trabajo, resolviendo dudas y eliminando posibles impedimentos.
- **Priorización:** Como en el resto de los métodos ágiles, es crucial no perder tiempo y dinero en algo que no interesa inmediatamente para el producto. Para ello, es necesario tener unos requisitos perfectamente priorizados reflejando el valor del negocio.
- **Mantener un latido:** Es tremendamente valioso mantener un ritmo que dirija el desarrollo. Este latido marcará la pauta del trabajo y ayudará a los equipos a optimizar su trabajo. El tener un ritmo fijo de trabajo, tanto en el día a día como en las iteraciones o *Sprints*, permite que el equipo sea predecible, ya que este aprenderá a estimar la cantidad de trabajo a la que puede comprometerse. El mantener un latido ayuda a todos

a centrarse en crear el producto y, para ello, es básico tener fechas clave de una iteración muy estables. De esta forma se evita perder tiempo con cuestiones del tipo: hoy es martes, ¿a qué hora es la *Daily* (reunión diaria de sincronización)? o ¿el final de *Sprint* es el lunes próximo o este *Sprint* se dijo que duraría una semana más que el anterior? Por otro lado, mantener este latido permite convocar con mucho adelanto reuniones como la *Sprint Review* para que los asistentes puedan organizar sus agendas con suficiente antelación y no faltar a la cita.

Una de las principales características de *Scrum* es que, en cada iteración, todas las etapas de la creación de un producto se solapan, es decir, en cada *Sprint* se realiza la planificación, análisis, creación y comprobación de lo que se va a entregar al final del mismo.

El marco de trabajo general de *Scrum* está compuesto por una serie de roles, reuniones y de paneles de información o artefactos que se indican a continuación:

- **Los roles en el equipo Scrum:**
  - El **Product Owner** o dueño del producto. Es el responsable desde el punto de vista del negocio.
  - El **Scrum Master** es el responsable de que el equipo sea productivo, ayudándole en todo momento a conseguir el objetivo acordado y de asegurar que los principios de *Scrum* se están respetando.
  - El **equipo**. Es el responsable de la construcción del producto.

Importante:

Los roles en Scrum representan una responsabilidad en el proceso y no la posición dentro de la organización.

- **Los artefactos de Scrum:** Los *Backlog* o repositorios son los artefactos en los que el *Product Owner*, equipo y *Scrum Master* escriben los requisitos y tareas.
  - El **Product Backlog** es el lugar que contiene los requisitos del cliente priorizados y estimados. Es propiedad del *Product Owner*, aunque todos los afectados deben asesorar durante su creación y en el mantenimiento del mismo para que esté permanentemente actualizado. El *Product Backlog* está escrito en lenguaje de negocio y debe revisarse la priorización, al menos, antes del inicio de cada *Sprint*.
  - El **Sprint Backlog** es la selección de requisitos del *Product Backlog* negociados para el *Sprint* y que se ha descompuesto en tareas por el equipo para expresar los requisitos del cliente en un lenguaje técnico. El *Sprint Backlog* es propiedad del equipo.
  - El **Burndown Chart** es una gráfica en la que se representa el trabajo pendiente del equipo. Existen dos tipos de gráficas principales: la relacionada con el *Sprint* y la

relacionada con la totalidad del proyecto.

- **Las reuniones en Scrum:** Se basan en el principio de *time-boxing* para acotarlas en el tiempo. Por ejemplo, en el caso del *Daily Meeting* o reunión diaria se recomienda que esté entre 10 y 15 minutos, mientras que para el resto de reuniones se sugiere una hora de reunión por semana de iteración (*Sprint Planning*; *Sprint Review*) o aproximadamente una hora para la *Retrospective*. Poniendo estos límites de tiempo, se fomenta optimizar su contenido y no perder el foco.
  - **Planificación del Sprint (Sprint Planning):** Esta reunión es, como su nombre indica, el momento en el que se planifica el *Sprint*. La reunión debe finalizar con un objetivo claro y compartido sobre el trabajo que hay que realizar para la iteración siguiente y con un *Sprint Backlog* adecuado. El equipo selecciona los ítems del *Product Backlog* con los que considera que puede comprometerse a realizar durante el *Sprint* y los dividirá de forma colaborativa en tareas.
  - **Reunión diaria (Daily Meeting):** La *Daily Meeting* es el momento de la sincronización del equipo en la que cada miembro comenta con el resto en qué estado se encuentra el trabajo que está realizando y con qué piensa continuar. Es el momento también para compartir con el equipo, de forma muy breve, si se tiene algún impedimento para continuar con el trabajo y así facilitar que se desbloquee.
  - **Revisión del Sprint (Sprint Review):** Al finalizar el *Sprint*, el equipo analiza el estado de su trabajo con el *Product Owner* y con cualquier otra persona que pueda aportar información valiosa. Esta revisión del trabajo debe hacerse de manera informal y no debe emplearse demasiado tiempo en prepararse. Este es el momento de analizar para mejorar "el qué" estamos construyendo.
  - **Retrospectiva del equipo (Sprint Retrospective):** Después de la *Review*, el equipo se reunirá para buscar mejorar en su trabajo y analizar los aspectos que le impiden ser más productivo. Es este el momento de analizar para mejorar "el cómo" estamos trabajando.

Nota:

Scrum es poco prescriptivo, pero lógicamente se pueden añadir los roles, artefactos o reuniones que sean necesarios. Eso sí, tal y como recomienda Henrik Kniberg[6]*, es mejor estar seguro de que se necesita algo nuevo antes de incorporarlo, basándonos siempre en la filosofía general Agile de hacer las cosas de la manera más sencilla posible. En caso de duda, comience por lo mínimo y vaya añadiendo lo que realmente se necesite en cada momento.*

De forma muy simplificada se podría resumir el flujo del trabajo con *Scrum* de la siguiente manera:

1. El *Product Owner* escribe en el *Product Backlog* todas las funcionalidades y requisitos que quiera que su producto contemple. Eso sí, debe priorizarla indicando el orden en que quiere que se vaya construyendo su producto. Los ítems más prioritarios deben estar más detallados que los que no son tan urgentes.
2. El equipo estimará cada uno de estos requisitos en función de su complejidad. Teniendo en cuenta la prioridad marcada por el *Product Owner* y la estimación realizada por el equipo, se acordará la cantidad de trabajo que se vaya a abordar en el siguiente *Sprint*. Ojo, los requisitos seleccionados no podrán cambiarse durante el *Sprint*.
3. Empieza el *Sprint* y el equipo se sincronizará diariamente con la *Daily Meeting*.
4. Al finalizar el *Sprint*, el equipo muestra al *Product Owner* el trabajo realizado que debe ser un producto potencialmente entregable. Con la opinión y sugerencias del *Product Owner* y la información obtenida en la retrospectiva posterior que realizará el equipo, se preparará la siguiente iteración.

Este flujo de trabajo se repetirá tantas veces como sea necesario hasta que se complete el *Product Backlog*, o bien se acabe el presupuesto o se llegue a una determinada fecha.

Scrum no es una metodología de trabajo, es más bien un marco de referencia. No da recomendaciones concretas para cada situación. Ayuda a que se planteen las preguntas correctas y son las personas las últimas responsables de encontrar las respuestas acertadas.

En definitiva, *Scrum* ayuda enormemente a que salga a la luz todo aquello que nos impide ser más productivos y, cuando un problema se detecta, se convierte inmediatamente en un obstáculo al que se le puede poner solución.

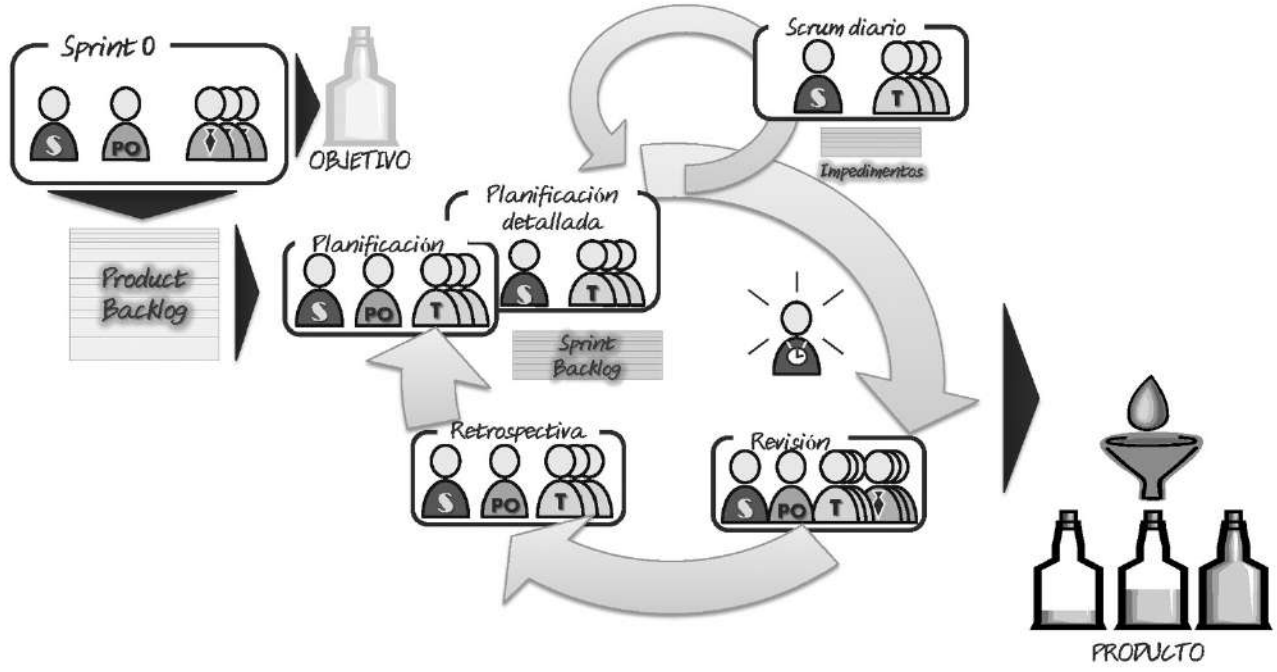


Figura 1.4. Ciclo de Scrum.

Scrum es tremendamente eficaz aplicado al desarrollo de aplicaciones software, pero no es exclusivamente en este campo donde ha demostrado su utilidad. Se ha incorporado también con éxito en todo tipo de proyectos tecnológicos y no tecnológicos.

Las reglas de *Scrum*, como se ha visto, son pocas y muy sencillas; por esto, *Scrum* es muy fácil y a la vez complicado aplicar con éxito. Ken Schwaber lo compara con el ajedrez[7] en el sentido de que en ambos casos las reglas del juego son sencillas, pero para ser un maestro es necesario practicar, aprender y mejorar continuamente. *Scrum* o el ajedrez no fracasan o triunfan. Simplemente marcan reglas que hay que seguir para jugar. En el caso del ajedrez, con la práctica se puede llegar a ser un gran en maestro. En el caso de *Scrum*, ese maestro será el que consiga que la organización funcione con éxito, que los clientes lo valoren y los usuarios lo aprecien y que sea respetado por la competencia.

Figura 1.5. Scrum, como el ajedrez, no triunfa o fracasa Simplemente marca reglas.

Scrum puede ser adaptado a las necesidades del proyecto que se vaya a abordar y proporciona un marco de trabajo y comunicación que ayuda a alcanzar el éxito. Está siendo aplicado con éxito cada vez en más empresas y para la creación de los más diversos productos. Por este motivo, se va a dedicar una buena parte de este libro a describir y analizar *Scrum* en profundidad.

Kanban

Kanban es una palabra de origen japonés y que significa “tarjetas visuales”. Aplicando este método se consigue mostrar permanentemente y de forma muy visual el estado del proyecto a todos los implicados. Métodos similares al que propone Kanban llevan aplicándose con éxito en las cadenas de producción desde hace más de un siglo. La aplicación de Kanban al desarrollo de software es comparativamente reciente.

Kanban es un método tremendamente útil para gestionar los productos cuyos requisitos cambian constantemente, bien porque aparezcan nuevas necesidades o bien porque su prioridad varíe. Este método también es útil en los casos en los que sea muy complicado planificar el trabajo, así como cuando no se pueda comprometer un equipo a trabajar con iteraciones de duración fija y predeterminada por el motivo que sea (interrupciones, cambios, dependencias, etc.). Se usa mucho para la resolución de incidencias y actividades de mantenimiento: es decir, cuando no se puede prever de antemano la cantidad de trabajo ni su naturaleza. De forma simplificada, los pasos que debe seguir para trabajar con Kanban son los siguientes:

- **Visualizar el flujo de todo el trabajo:** En un panel organizado en columnas debe estar representado todo el flujo del trabajo que hay que realizar en el proyecto, desde el principio hasta el último momento. Cada actividad se representa por una tarjeta (de ahí el nombre de Kanban). Este panel debe estar accesible y bien visible para todos los miembros del equipo.
- Para que el panel sea útil tiene que representar en qué **estado del flujo** está cada ítem en cada momento. La primera columna representa el *Backlog* del producto, es decir, la lista priorizada de las necesidades o actividades pendientes. Se puede usar tantas columnas como estados sean necesarios para que todo el flujo de trabajo esté contemplado.
- **Divida el trabajo en ítems pequeños** y escriba cada uno en una tarjeta. Priorícelos y colóquelos ordenados en la primera columna del tablero. Una buena práctica es tratar de dividir los ítems de forma que la carga de trabajo sea similar de unos con otros. Esto proporciona una gran ventaja, ya que permite estimar visualmente el trabajo asociado a cada estado.
- **Limite el trabajo en curso:** Esta es una de las claves para que trabajar con Kanban funcione. Es imprescindible poner un límite al número de ítems permitidos en cada columna y de esta forma evitar colapsos, cuellos de botella y eliminar cuanto antes los impedimentos que surjan y que impidan trabajar con un ritmo sostenible. Este número que indica el límite permitido en cada columna debe estar visible en la parte superior de la misma.
- **Mida el tiempo empleado en completar un ciclo completo:** Calcule el tiempo que se emplea desde que se empieza a trabajar con un ítem o tarea hasta que se da por cerrado

o terminado y trate de buscar la manera de disminuir este tiempo. Esta práctica ayuda también a ser predecible y poder hacer una estimación previa de cuánto tiempo empleará en completar cada ítem.

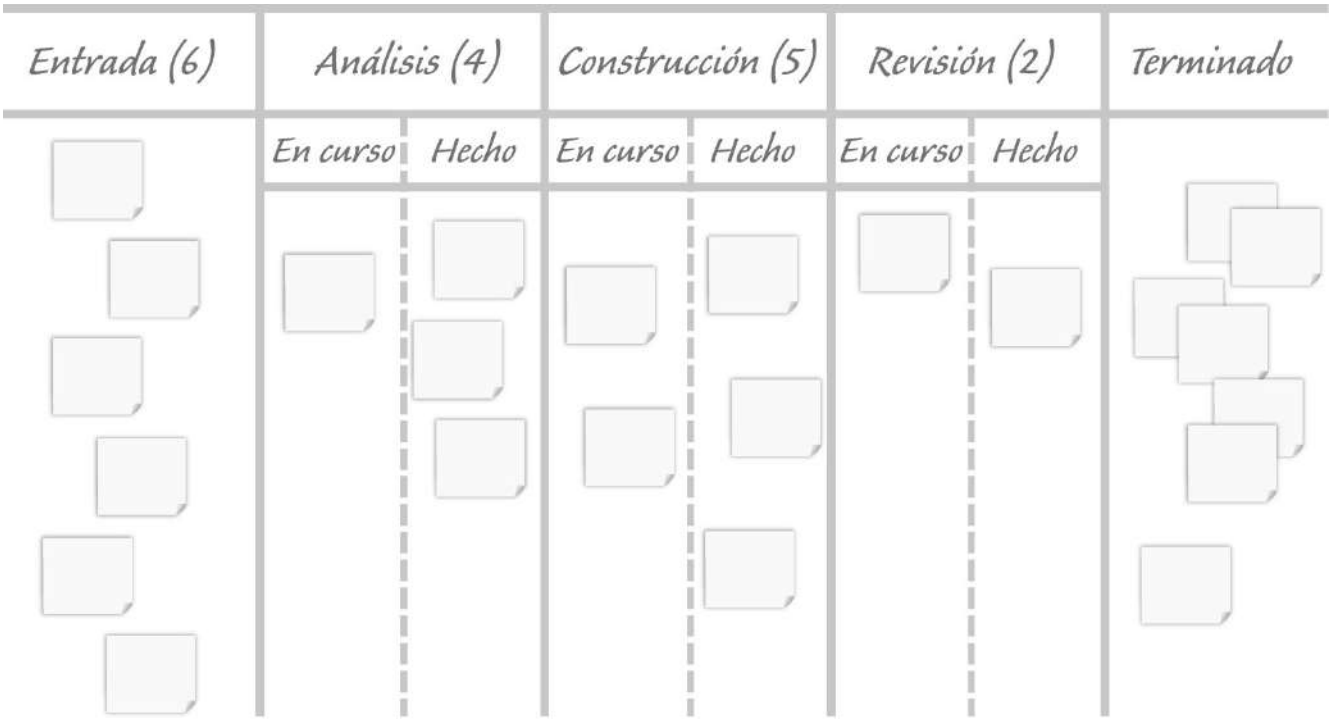


Figura 1.6. Panel de Kanban.

Con Kanban, los implicados en la creación de un producto tienen acceso a toda la información del mismo y al estado de cada una de sus partes en cada momento. El grado de compromiso aumenta notablemente, ya que todos pueden participar en la mejora directa e inmediata del proceso. Trabajando de esta forma tan visual, se facilita, tal y como se ha explicado, identificar los problemas, detectar cuellos de botella y eliminar impedimentos según se producen, reduciendo costes y aumentando la productividad y la calidad. Por otro lado, todo el equipo participa en la mejora de la totalidad y no se centra solo en su parte.

Las características de Kanban hacen que pueda utilizarse para organizar y gestionar el trabajo en cualquier campo y no exclusivamente para proyectos de desarrollo de software. Por ejemplo, Kanban es muy usado por equipos de soporte y mantenimiento. Pero puede ser muy útil también para cualquier otra área de una organización, como recursos humanos, el departamento financiero, etc. Kanban se usa también como una herramienta de productividad personal, es decir, para seguir las tareas de una única persona.

El uso de Kanban puede ser una buena elección si se pretende hacer una transición en una empresa a las metodologías ágiles de forma gradual.

En el capítulo 13 se expone con más detalle la forma de trabajar con Kanban.

Pragmatic Programming

En realidad, *Pragmatic Programming* no es un método en sí mismo, sino de una serie de mejores prácticas de programación englobadas dentro de la filosofía ágil. Estas prácticas fueron descritas en un libro en el año 2000 por Andrew Hunt y David Thomas[8], ambos firmantes del manifiesto *Ágil*.

La filosofía de estas prácticas se puede resumir muy brevemente en los siguientes puntos:

- Cuando se comprometa a realizar un trabajo, debe asumir la responsabilidad de hacerlo lo mejor que pueda. Ante cualquier problema, piense en soluciones y no en excusas.
- Construya con un buen diseño y cree código de calidad. Corrija los defectos en cuanto los detecte o, si esto no es posible, planee corregirlos lo antes posible.
- Si considera que es necesario realizar algún cambio, fomente que se realicen y forme parte activa de este proceso de cambio.
- Es crucial construir un producto que sea satisfactorio para el cliente, pero es importante también saber detectar en qué momento es necesario dejar de construirlo y no seguir durante un tiempo indefinido añadiendo funcionalidad que no se ha solicitado.
- El aprendizaje continuo debe ser una constante para cualquier persona implicada en la construcción del producto.
- Aunque esté programando, la comunicación con los demás es clave y, por ello, debe tratar de mejorar sus habilidades de comunicación constantemente.

Feature Driven Development (FDD)

El desarrollo orientado a funcionalidad (FDD) es un método ágil concebido para el desarrollo de sistemas informáticos. Este método no pretende cubrir todo el proceso de desarrollo de un producto, solo se centra en las fases de diseño y construcción. Sus puntos clave son el trabajo en iteraciones, el control continuo, la calidad de lo creado y entregas frecuentes para poder realizar un seguimiento continuo en colaboración con el cliente y poder así incorporar sus necesidades en el producto con frecuencia.

FDD[9] propone seguir una serie de pasos secuenciales para realizar el diseño y la implementación del sistema. Estos pasos son:

- **Crear un modelo global:** Un primer paso será tener un conocimiento del alcance, los requisitos y el contexto del sistema en el que se va a construir el producto. Al finalizar esta etapa, los miembros contarán con una descripción general del sistema. Aunque FDD no da indicaciones precisas de cómo elaborar las necesidades que el producto debe cubrir, en esta fase sería interesante que existieran unos requisitos o especificaciones al menos a alto nivel.

- **Crear una lista de funcionalidades:** Es necesario plasmar ese modelo global, junto con los demás requisitos del conjunto del sistema, en una única lista de necesidades o funcionalidades que cubrir. Los desarrolladores participarán en la creación de esta lista y estas funcionalidades se revisarán por los usuarios y clientes para que se completen y validen.
- **Planear por funcionalidades:** A la hora de hacer un plan a alto nivel, debe siempre tenerse en cuenta la prioridad de las funcionalidades y las dependencias entre ellas. Para definir los hitos de entregas, debe hacerse también desde un punto de vista de grupos de funcionalidades.
- **Diseñar y construir por funcionalidad:** Se deben diseñar y construir las funcionalidades de forma iterativa. En cada iteración se seleccionará un conjunto de funcionalidades y estos ciclos deben oscilar entre algunos días y dos semanas. Puede haber varios equipos trabajando en paralelo en diferentes grupos de funcionalidades. Esta etapa de diseño y desarrollo incluye las actividades de pruebas unitarias, revisión de código e integración.

Dynamic Systems Development Method (DSDM)

La primera versión de DSDM nace en 1994 en el Reino Unido y es considerado uno de los marcos de trabajo más extendidos para el desarrollo rápido de aplicaciones en este país.

La forma de trabajar que propone este método para el ciclo de vida de un proyecto está estructurada en 5 fases, de las cuales las dos primeras se realizan una sola vez y las tres últimas se realizan de forma iterativa e incremental. Estas etapas son: estudio de la viabilidad del proyecto, estudio del negocio, iteraciones del modelo funcional, iteraciones para la creación del diseño y desarrollo del producto y, finalmente, iteraciones para la implementación.

Tal y como explica Dean Leffingwell[10], la filosofía de DSDM es sencilla:

- El desarrollo de un producto debe entenderse como un trabajo en equipo, ya que para que tenga éxito debe combinarse el conocimiento de las necesidades del negocio que tienen los clientes con el perfil técnico de los desarrolladores.
- La calidad debe contemplarse desde dos puntos de vista: la solidez técnica y la facilidad de uso.
- El desarrollo puede y debe hacerse de forma incremental. ¡No es necesario entregarlo todo a la vez! Es mejor entregar una parte útil a tiempo, a tener demasiado tarde todo el producto.
- Debe trabajarse inicialmente en las funcionalidades que aporten mayor valor al negocio y por tanto, los recursos se deben invertir en el desarrollo de las mismas.

Uno de los principios básicos en los que se basa DSDM es la creencia de que un requisito no se puede prefijar completamente al inicio del desarrollo, y, cuando así se hace, solo una parte de este es realmente valiosa para el cliente. Identificar esta parte del requisito más estratégica o necesaria precisa trabajar profundamente el alcance de los requisitos y su prioridad. Otro aspecto que resultó revolucionario cuando lo propuso DSDM fue dar la vuelta al enfoque de los métodos tradicionales en los que se fijaban unos requisitos y, en función de ellos, se estimaban tanto los recursos como la fecha de entrega. Lo que DSDM propone es fijar los recursos destinados a un producto y la fecha de entrega y hacer una estimación de la funcionalidad que se entregará. En definitiva, se sabrá cuándo se va a entregar algo valioso al cliente, pero de antemano no se conocerá el detalle de lo que se va a entregar.

De manera resumida, algunas de las prácticas fundamentales de DSDM son las siguientes:

- Debe mantenerse un latido. Se debe trabajar con iteraciones de duración fija y establecer unos plazos de entrega con fechas fijas. Es la forma de mantener la flexibilidad en los requisitos y que el producto final sea realmente valioso. Cada iteración se basa en una serie de requisitos priorizados antes de llevarla a cabo.
- Los requisitos deben estar priorizados y clasificados de forma que esté claro qué necesidades son fundamentales para el éxito del proyecto y, si estos fallaran, el proyecto podría ser cancelado. Con esta priorización, se podrá detectar así mismo qué requisitos son importantes pero el éxito del proyecto no recae sobre ellos, cuáles tienen poco impacto y finalmente aquellos requisitos que pueden ser aplazados sin demasiado problema.
- Se fomenta el prototipado de forma que los usuarios puedan comprobar que el detalle de los requisitos es el suficiente para la construcción del producto final y fomenta el intercambio de información entre desarrolladores y usuarios. De esta forma, el producto final cubrirá las expectativas del cliente.
- La calidad debe asegurarse en todo el proceso. Como la mayoría de los métodos ágiles, DSDM da una gran importancia a las pruebas durante todo el ciclo de vida del desarrollo, de igual manera que lo hacen para una correcta gestión de la configuración.

Nota:

Para acceder a más información sobre DSDM puede visitar la página Web www.DSDM.org.

Programación eXtrema o eXtreme Programming

Tal y como lo define Kent Beck[11] *eXtreme Programming* (XP) es un método ágil para el desarrollo de software muy útil a la hora abordar proyectos con requisitos vagos o

cambiantes. XP es especialmente útil si se aplica a equipos de desarrollo pequeños o medianos. Es un método adaptativo, es decir, se ajusta muy bien a los cambios. Propone desarrollar el código de forma que su diseño, arquitectura y codificación permitan incorporar modificaciones y añadir una funcionalidad nueva sin demasiado impacto en la calidad del mismo. Por otro lado, XP es un método muy orientado hacia las personas, tanto a las que están creando el producto como a los clientes y usuarios finales.

Desarrollando como propone XP, se obtienen rápidamente resultados. Al trabajar con pequeñas iteraciones, se puede obtener con frecuencia comentarios del cliente, lo que tiene como resultado que el producto final cubra ampliamente sus expectativas y necesidades. Como en otros métodos ágiles, la forma de crear el producto será de forma incremental con todas las ventajas que ya hemos comentado que esto supone. Para XP las pruebas son la base de la construcción y propone que sean los desarrolladores los que escriban las pruebas a medida que van construyendo el código y se realice una integración continua, de forma que el software creado tenga una gran estabilidad. Las pruebas automáticas se realizan de forma constante para poder detectar los fallos rápidamente. Cuanto antes se detecte un problema, antes podrá resolverse sin que las consecuencias y el impacto sean mayores.

Antes de cada iteración se planifica el trabajo que va a realizarse y, a continuación, se realizan de forma simultánea el análisis, el desarrollo, el diseño y las pruebas del código. Para que esto sea posible, es necesario seguir unas prácticas[12] que se comentan en detalle en el capítulo 10 de este libro.

XP se basa en un conjunto de ideales a los que llama valores, que son los que guían el desarrollo en XP.

Estos valores son los siguientes:

- Comunicación.
- Simplicidad.
- *Feedback*.
- Valentía.
- Respeto.

Los valores de XP son demasiado abstractos y es necesario concretar algo más para ponerlos en práctica. Para ello, XP propone algunos principios útiles para un mejor desarrollo. Estos principios son:

- Humanidad.
- La economía.
- La búsqueda del beneficio mutuo.
- La autosemejanza.
- Mejora continua.

- Diversidad.
- Reflexión.
- Simultaneidad de fases o flujo.
- Oportunidad.
- Redundancia buscando soluciones.
- Aprender de los fallos.
- Búsqueda constante de la calidad.
- Avanzar con pequeños pasos.
- Aceptar la responsabilidad de todos los implicados en el desarrollo de un producto.

Finalmente, la aplicación de estos valores y principios a un proyecto específico de desarrollo de software se hará aplicando las prácticas que XP propone. Kent Beck redefinió XP posteriormente en la que basa el desarrollo en trece prácticas primarias y once prácticas añadidas a modo de corolario. La aplicación de las prácticas primarias aporta un beneficio directo en la creación de software. Las prácticas corolario deben usarse siempre y cuando las primarias ya se apliquen porque asume que existe una madurez y experiencia en el desarrollo.

Las prácticas primarias son las siguientes:

- Trabajar con historias de usuario.
- Realizar ciclos semanales de desarrollo.
- Organizar revisiones trimestrales.
- Trabajar con holgura.
- El equipo debe sentarse junto.
- El equipo debe ser completo, es decir, estar compuesto por todas las personas necesarias para llevar a cabo el producto con éxito.
- Tener información sobre el proyecto en el puesto de trabajo.
- Mantener la energía en el trabajo a un ritmo sostenible.
- Realizar con frecuencia la programación en parejas.
- Diseño incremental.
- Realizar las pruebas antes de programar.
- Construir en diez minutos.
- Integración continua.

Las prácticas corolario son:

- Participación real de los clientes.
- Despliegue incremental.
- Negocie el alcance del contrato.
- Pague por funcionalidad.

- Continuidad de los equipos.
- Reducir los equipos.
- Análisis de las causas.
- Código y pruebas.
- Código compartido.
- Código base único.
- Despliegue diario.

Clave:

XP apuesta porque las pruebas se hagan pronto, con frecuencia y automatizadas.

Lean Startup

Este método de trabajo es más reciente que los anteriores y no nace para atender necesidades en entornos tecnológicos. Aunque puede ser utilizado para propósitos muy diversos, ha demostrado ser especialmente útil para definir productos y modelos de negocio, por lo que se aplica tanto por nuevas empresas en proceso de formación (*startup*) como en departamentos de empresas ya consolidadas que definen nuevos productos y servicios.

Lean Startup es complementario de otros métodos, no de manera excluyente. Es importante seleccionar en cada momento la mejor forma de trabajo para cada etapa o propósito. Así, puede definirse un producto con *Lean Startup*, implementarse usando *Scrum* y gestionar su soporte con Kanban. Es más importante seleccionar la herramienta adecuada en cada caso que ceñirse ciegamente a un dogma.

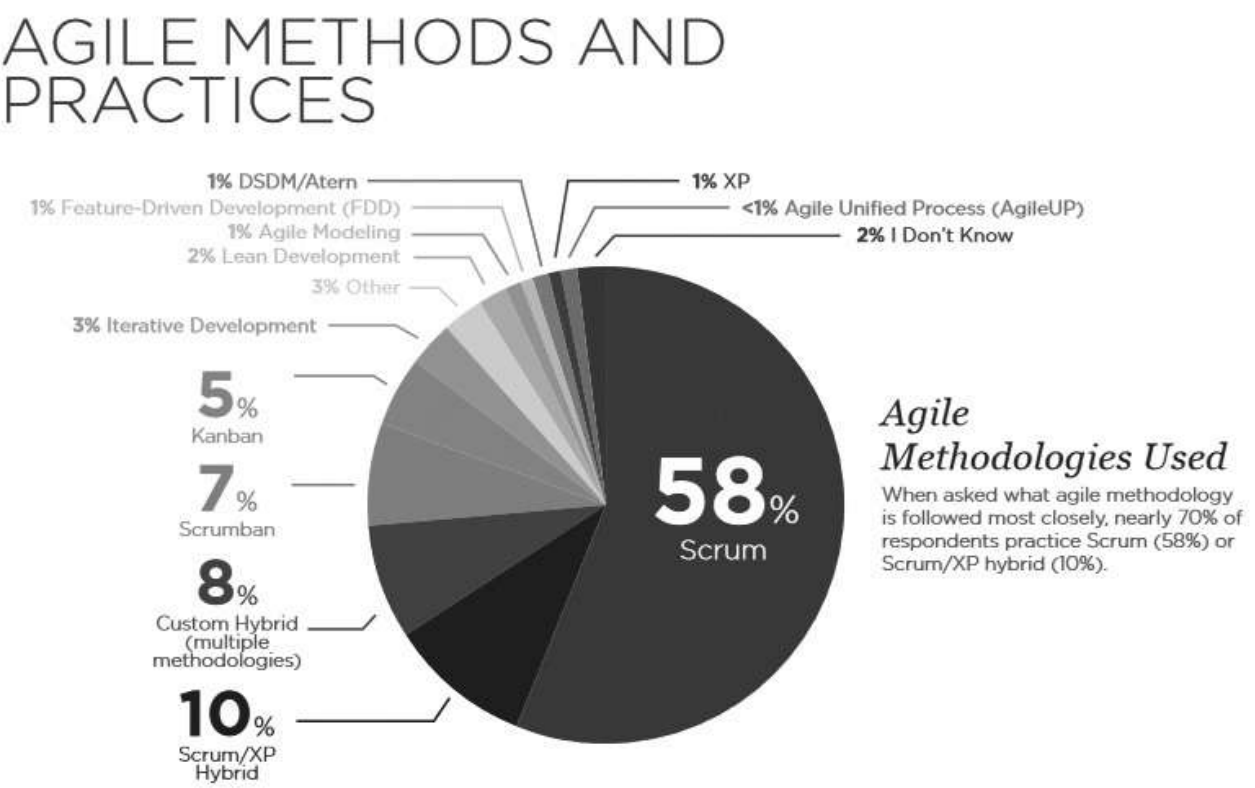


Figura 1.7. Gráfica de la adopción de los diferentes métodos ágiles (VersionOne, 2016).

El origen de *Lean Startup* está en los trabajos de Steve Blank y, especialmente, en el libro de Eric Ries[13], alumno suyo, que ha contribuido decisivamente a popularizarlo.

Lean Startup es un método iterativo, inspirado en la filosofía Lean, que avanza en el diseño del producto a través de iteraciones en las que se va refinando el concepto, cambiando el enfoque si el planteamiento inicial no acaba de funcionar.

En el capítulo 15 se explica *Lean Startup* en detalle, pero se anticipan aquí alguno de sus conceptos básicos:

- **Producto Mínimo Viable, o MVP (Minimum Viable Product):** Se trata de la definición del producto en el que se está trabajando (o el producto en sí) al final de una iteración. Esa definición debe ser capaz de validar las hipótesis que se han planteado como objetivo a través de unas métricas. El MVP debería incluir las capacidades y funciones básicas del producto, aquellas que dan el mayor valor posible al cliente, dejando de lado elementos superfluos.
- **Ciclo "Crear-Medir-Aprender":** *Lean Startup* se basa en iteraciones y un avance progresivo hacia el objetivo final, al igual que todos los métodos ágiles. El ciclo de trabajo se basa en tres etapas: crear el producto a partir de las ideas; ponerlo en el mercado y medir su comportamiento; extraer información y aprender más para poder aplicarlo en la siguiente iteración.
- **Métricas:** La insistencia en medir resultados y extraer información también es común a

muchos métodos ágiles, pero en *Lean Startup* es especialmente importante. Mientras en métodos como *Scrum* o Kanban se mide el proceso, en *Lean Startup* se pide el producto, o, más bien, su comportamiento en el mercado. La creación de métricas adaptadas a cada circunstancia es uno de los aspectos más relevantes de este método de trabajo.

- **Pivotar:** Esta expresión se ha popularizado gracias a *Lean Startup*. En cada iteración se plantean unas hipótesis, que se trasladan al producto y cuyo resultado se evalúa por medio de las métricas para comprobar la validez de las asunciones iniciales. Si las hipótesis se ven refrendadas, se persevera, pero, si deben cambiarse, se considera un cambio de rumbo, pivotando en una nueva dirección. Normalmente se pone como ejemplo a compañías de Internet que nacieron con un propósito y fueron cambiando de orientación hacia algo completamente distinto a medida que iban descubriendo las verdaderas necesidades de los clientes.

Al igual que otros métodos, *Lean Startup* también tiene una serie de principios que lo fundamentan. Son los siguientes:

- Hay emprendedores por todas partes. Lo que viene a decir que este método no está dirigido a un único tipo de actividad (pequeñas empresas innovadoras que arrancan), sino que puede aplicarse en muchos otros campos.
- El emprendimiento es gestión. No debemos olvidar que al final una organización necesita control, no se trata solo de crear un producto nuevo.
- Aprendizaje validado. En cada paso aprendemos sobre el producto que estamos diseñando y ese conocimiento debe estar fundamentado en hipótesis validadas por experimentos y métricas, no en intuiciones o prejuicios.
- Contabilidad de la innovación. No hay que perder de vista los aspectos aburridos del proceso, hay que medirlo y gestionarlo, priorizar trabajos, analizar información, establecer y valorar objetivos. Además de los aspectos más atractivos, hay que atender a los más aburridos de la innovación.
- Crear-Medir-Aprender, o la orientación iterativa del trabajo para poner a prueba las ideas iniciales y refinarlas con lo que se aprende de su exposición al mercado real.

En los siguientes capítulos, se profundizará en varios de estos métodos, con atención especial a *Scrum*, así como en otros temas que le ayudarán a tener una visión suficientemente amplia de los métodos ágiles y le permitirán aplicarlos en su trabajo, proyectos e incluso actividades personales.

Visiones alternativas a los métodos ágiles

Desde que se publicó la primera edición de este libro en 2011, se ha producido una rápida popularización de los distintos métodos ágiles. Aunque *Scrum* y Kanban siguen siendo los más aplicados, la difusión de *Lean Startup* ha contribuido a llevar la visión ágil más allá del entorno tecnológico y del software, donde se ha convertido en la cultura de trabajo dominante.

La certificación, es decir, el obtener títulos oficiales a cargo de organizaciones habilitadas para ello, se ha multiplicado. A los iniciales certificados de *Scrum Master* y *Product Owner* otorgados por la Scrum Alliance[14], se han unido una larguísima lista de títulos, cursos y certificados que han agregado un "Agile" a su nombre (como, por ejemplo, el *Certified Agile Tester* del ISQI[15], un conocido organismo certificador en temas de calidad, o distintos títulos de gestión de proyectos con un matiz "Agile", seguramente más nominal que real).

A esto hay que añadir que muchas consultoras han adoptado, al menos nominalmente, *Agile* y se ha acabado convirtiendo en algo obligado el hecho de mencionar la aplicación de métodos ágiles en toda propuesta o contrato de proyecto.

Para quienes creen en los beneficios de estos métodos de trabajo, esta tendencia hacia una mayor popularidad puede parecer positiva. Pero no todo el mundo lo ve así. Hay quien habla de "mercantilización", "pérdida de valores", de sustituir la esencia por la adopción nominal mientras se siguen aplicando métodos de trabajo caducos cuya ineficacia se ha demostrado repetidas veces. Por ello, han surgido también en los últimos tiempos movimientos que quieren devolver la esencia perdida.

La siguiente enumeración se entenderá mejor si ya se tiene un cierto grado de conocimiento y experiencia en la aplicación de métodos ágiles, por lo que, si este libro es su primer contacto con ellos, se le recomienda volver más tarde sobre esta lista:

- Vigencia del manifiesto *Ágil*: Una encuesta realizada por ACM en el décimo aniversario de la publicación del manifiesto *Ágil* mostró que, para la mayoría de las personas que los aplica, sus principios siguen estando vigentes y que no es necesario revisarlos. Eso no impide que incluso alguno de sus firmantes originales considere que hay que retirar el término "ágil" debido a la popularización y mercantilización del mismo. Otras visiones contra el manifiesto hablan de que todo lo ágil se ha convertido en una doctrina, perdiendo su lado más creativo y la visión individual en favor de una suerte de "dogma".
- *Heart of Agile* y *Modern Agile*: Se trata de movimientos que propugnan una vuelta a las raíces de lo ágil. Ponen foco en la mejora continua, el valor de las personas, la entrega de valor y otros principios básicos desde una perspectiva muy *Lean*, es decir, simple y desprovista de elementos considerados accesorios o innecesarios.
- "*No project*": Propone romper con la idea de que se puede combinar la gestión tradicional de proyectos con una gestión *Agile*. Su punto de partida es romper con la idea de "proyecto" con sus fechas, seguimiento, diagramas y herramientas de control. El movimiento "*No estimate*" va en la misma dirección: evite estimar como una forma

de romper con control que supone trabajar con la orientación usada hasta ahora en los proyectos, más predictiva que iterativa.

- Personas y no organizaciones: Agile no debe servir como excusa para poner las organizaciones y sus necesidades por encima de las personas y sus relaciones, que deberían ser los verdaderos protagonistas del modo de trabajo ágil.
- Ágil no es "rápido": En la experiencia personal de los autores aparece con demasiada frecuencia ese error. Cuando se habla de una solución ágil, de ser ágiles en la obtención de resultados, de que la toma de decisiones debe ser ágil... se está pensando en velocidad. Es cierto que la definición en el diccionario habla tanto de rapidez como habilidad, pero hay que ser consciente de que "ágil", en el sentido dado a esta filosofía, se refiere también a la flexibilidad, a la adaptación y a la simplicidad. Dejar que se asocie "ágil" solo con "rápido" es sacrificar elementos básicos (por encima de todo, la calidad) a la consecución inmediata de resultados.

Por último, hay que comentar que otra de las dificultades de la adopción de *Agile* es su aplicación en trabajos o proyectos grandes que involucren a equipos muy numerosos. Más adelante, en el capítulo 12 se tratará precisamente sobre cómo afrontar este reto.

El futuro que espera al mundo de los métodos ágiles es equilibrar su popularización con mantener su esencia para así evitar una aplicación nominal sin seguir sus principios, lo que restaría todo su valor.

Si el primer contacto con el mundo *Agile* del lector procede de la lectura de estas páginas, sepa que va a entrar a formar parte de una comunidad activa, creyente en sus valores, celosa de sus principios y dispuesta a llevar los beneficios de la agilidad por encima de convertir el término "ágil" en una moda de gestión más.

¡Buena suerte!

---

4 T*he New New Product Development Game*. Hirotaka Takeduchi and Ikujiro Nonaka. *Harvard Business Review*. 1986.

5 *Agile Software Development with SCRUM*. Ken Shwaber & Mike Beedle. Publisher: Prentice Hall. 2001.

6 *KANBAN and SCRUM. Making the most of both.* Henrik Kniberg and Mattias Skarin. C4Media Inc. 2010.

7 http://www.kenschwaber.wordpress.com/2011/04/07/SCRUM-fails/.

8 *The Pragmatic Programmer.* Andrew Hunt y David Thomas. Addison Wesley. 2000.

9 *A Practical Guide to Feature-Driven Development.* Palmer, S.R. and Felsing, J.M. Upper Saddle River, NJ, Prentice Hall. 2002.

10 *Scaling Software Agility. Best Practices for Large Enterprises*. Dean Leffingwell. Alistair Cockburn and Jim Highsmith, Series Editors. 2007.

11 *eXtreme Programming Explained*. Kent Beck. Reading, Mass., Addison-Wesley. 1999.

12 *eXtreme Programming Explained: Embrace Change (2nd Edition)*. Kent Beck with Cynthia Andres. Ed. Addison

Wesley. 2004.

13 *El método Lean Startup*. Eric Ries. Ed. Deusto. 2011.

14 https://www.scrumalliance.org/.

15 http://www.agile-teaming.org/.

Primera parte
Trabajar con Scrum

2 Scrum de un vistazo

En este capítulo aprenderá:

- Los valores de *Scrum*.
- Cómo es el ciclo de trabajo de un proyecto *Scrum*.
- Cuáles son los principales roles.
- Las herramientas y artefactos *Scrum*.

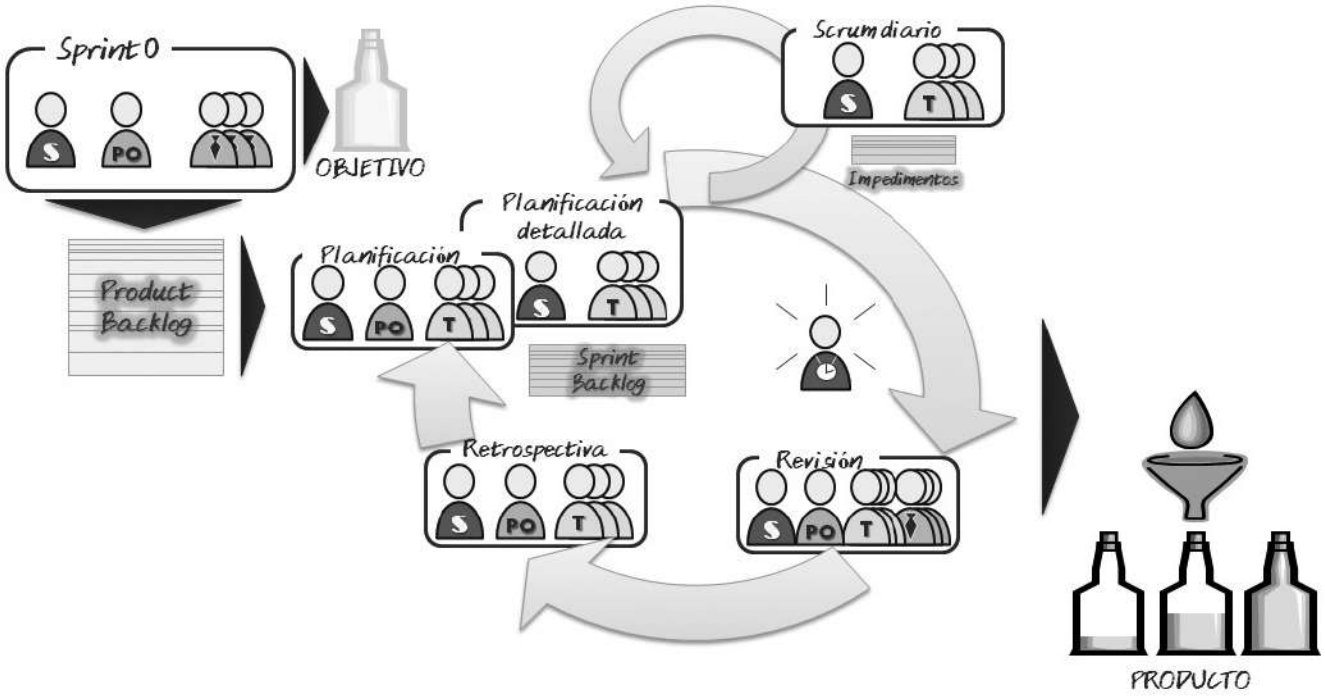


Figura 2.1. El ciclo Scrum.

Introducción

Antes de estudiar cómo se desarrolla el proceso *Scrum* detalladamente, se empezará por dar una visión general como introducción a los conceptos que se manejarán más adelante.

Esta primera parte del libro se estructura de la siguiente forma: en este capítulo, tendrá una visión de cómo funciona *Scrum*, sus distintas etapas y roles; y, en sucesivos capítulos, se irá viendo en detalle cada uno de estos conceptos. Este capítulo servirá de guía y referencia en el futuro.

Como ayuda, se ha diseñado un "mapa de *Scrum*" como una página especial en la que se describe gráficamente el proceso y se enumeran los valores, roles, artefactos y componentes de *Scrum*. Podrá encontrarla en el propio libro o en Internet[16]. Fotocopie o descargue esa página, distribuya copias entre su equipo, déjela en un lugar visible y úsela como mapa o guía para familiarizarse con la terminología y los procesos de *Scrum*.

El resto del capítulo es una introducción al contenido posterior. Junto con el mapa, será su guía en *Scrum*. Encontrará información más detallada en los siguientes capítulos del libro.

Visión esquemática del ciclo Scrum

Véase ahora un resumen del ciclo de trabajo con *Scrum* de acuerdo con el mapa que encontrará en el libro. Podrá revisar estos conceptos con más extensión en los siguientes capítulos.

Sprint 0. Todo empieza aquí. Es una etapa previa muy importante para el desarrollo del resto del trabajo y en la que tiene un papel relevante el Product Owner o PO, que es la persona que actúa como punto de contacto entre cliente y equipo. Partiendo de las necesidades del proyecto, el PO se encarga de armar las bases para el trabajo posterior: crear el equipo, identificar los recursos necesarios, fijar requisitos y plazos, traducir las necesidades de cliente en unos requisitos y elaborar un diseño preliminar formal. El resultado es un Backlog o repositorio del proyecto en el que se han introducido todos los requisitos expresados en forma de temas, épicas e historias de usuario (de menor a mayor grado de detalle). El contenido del *backlog* se ordena de acuerdo con la prioridad.

Todo el trabajo posterior se divide en iteraciones o *Sprints*, que, a su vez, se agrupan en una o varias *Releases* o entregas (al menos habrá una final) de acuerdo con la longitud del proyecto, las necesidades del cliente y la naturaleza del trabajo.

Cada una de esas iteraciones o *Sprints* arranca con el Planning, en el que el PO prioriza todas las historias de usuario (temas y épicas) y añade criterios de aceptación para determinar cuándo se han cumplido. Para ello, podría contar con la ayuda del Scrum Master o SM (facilitador del trabajo, intermediario entre el PO y el equipo y vigilante del cumplimento de los principios de *Scrum*). El equipo de trabajo valora el esfuerzo necesario para realizarlas y selecciona, siguiendo la priorización fijada, las que podrán realizarse en el transcurso del *Sprint* de acuerdo con la capacidad de trabajo del equipo. En una segunda etapa de la planificación, el equipo traduce las historias al lenguaje del proyecto y las subdivide en unidades menores o tareas. Con todo ello se construye el Sprint Backlog o repositorio de los trabajos que se realizarán durante la iteración.

A lo largo del *Sprint* (entre una y cuatro semanas) se va realizando el trabajo y el equipo va exponiendo sus avances en la Daily meeting, *Scrum* diario o reunión diaria. Se trata normalmente de un foro interno (compuesto por el equipo y el SM) donde cada participante comenta los avances realizados, el trabajo futuro y los impedimentos para la actividad. Estos impedimentos se guardan en un Impediment Backlog y el SM vigila que se resuelvan.

El final del *Sprint* lo señala la **Review meeting** o reunión de revisión de resultados, donde el equipo, con el SM, expone los trabajos realizados y los resultados alcanzados al PO y al resto de personas que puedan estar interesadas, para que los acepten o no, y recoger comentarios y sugerencias que se puedan aplicar en la próxima iteración.

Finalmente, en la reunión de Retrospectiva, el equipo y el *Scrum Master* revisan el proceso e identifican, por un lado, aspectos positivos que merece la pena cuidar y mantener y, por otro, puntos de mejora que hay que resolver dentro del proceso de mejora continua que

supone *Scrum*.

Todo el proceso queda documentado y se hace un seguimiento de la capacidad del equipo o velocidad como medida útil para el incremento de la productividad.

Todo este proceso se explicará con más detalle y ejemplos en los siguientes capítulos. No se preocupe si ahora no entiende algunos de los nombres y conceptos.

Los valores de Scrum

Vamos a empezar revisando los valores de *Agile* y *Scrum*. Ambos deberían estar en un sitio bien visible y repetirse periódicamente como un mantra. Pero no se trata de unos mandamientos inamovibles que se aprenden de memoria: merece la pena que todo el equipo sea consciente de ellos, los revise y examine periódicamente su cumplimiento.

Los valores que se deben tener presentes son:

- **Mejora continua:** Las metodologías ágiles se han creado para mejorar una forma ineficiente de trabajar. Pero no se quedan ahí: propugnan que esa mejora se amplíe y continúe en el futuro. Los miembros de un equipo *Scrum* deben identificar continuamente puntos de mejora y hacer lo posible por aplicarlos para realizar su trabajo de forma más productiva y con mayor calidad.
- **Calidad:** Es el objetivo de todos nuestros esfuerzos por mejorar la forma en la que trabajamos y los productos que construimos. Por ese motivo, la calidad debe ser un componente básico e innegociable. Sacrificar calidad frente a cualquier otro aspecto (como plazos o costes) compromete el proceso y sus resultados, y además se aleja de la filosofía *Scrum*.
- **Time-boxing:** Dentro del esfuerzo por alcanzar una mayor productividad hay que eliminar los ladrones de tiempo, y en un proyecto son muchos. Fijar una duración temporal estricta para las actividades que tienden a alargarse y dispersarse supone un beneficio claro para el proceso. *Time-boxing* significa que una reunión de una hora solo va a durar una hora, que no se van a permitir divagaciones innecesarias, y que solo se van a considerar los temas relacionados con el asunto que se trata y no otros. *Time-boxing* significa aprovechar el tiempo, evitar perderlo.
- **Responsabilidad:** Una organización en la que prima la auto-organización solo funciona cuando los miembros del equipo adquieren un grado de responsabilidad superior. En una organización muy jerarquizada, la responsabilidad se acumula a medida que se asciende en la organización. En *Scrum,* la responsabilidad es compartida y afecta a todos por igual.
- **Multidisciplinar:** El equipo de trabajo debe ser capaz de realizar todas las tareas necesarias del proyecto. En lugar de contar con equipos especializados solo en alguna

de las actividades necesarias, en *Scrum* se espera que cada uno pueda ser autónomo y realizar todos los trabajos precisos sin contar con contribuciones externas.

Figura 2.2. El trabajo en equipo es una de las bases de Scrum.

- **Flexibilidad:** Con *Scrum* se deja de lado la idea de que todo proyecto parte de una descripción estática de lo que quiere el cliente. En su lugar, *Scrum* reconoce la realidad y abraza el cambio, se define en torno a la idea de que los requisitos pueden cambiar. Por ese motivo, la flexibilidad es una de sus señas de identidad y se aplica a todo el proceso.
- **Ritmo (latido):** Para que la flexibilidad no degenere en una absoluta incertidumbre, *Scrum* favorece que el equipo trabaje a un ritmo determinado. Alcanzar ese ritmo será la base para convertirse en un equipo maduro, capaz de funcionar de forma sincronizada y de ofrecer estimaciones de alcance y fechas a sus clientes.
- **Compromiso:** *Scrum* exige un alto compromiso de todos los participantes en el proyecto. La confianza y autonomía que otorga a todos los participantes requiere que su actitud hacia el proyecto sea activa y comprometida.
- **Simplicidad:** *Scrum* prefiere las soluciones simples a las complejas. Es un rasgo de calidad y un valor añadido que se da al producto realizado, y facilitará la labor de los que tengan que trabajar en el futuro con él.
- **Respeto:** *Scrum* se centra prioritariamente en las personas y las relaciones entre ellas. Solo un respeto mutuo entre los participantes garantiza las relaciones necesarias para el éxito de un proyecto.
- **Coraje:** Los participantes en un proyecto *Scrum* deben afrontar decisiones comprometidas, tomar iniciativas y actuar en función de un objetivo común. Todo esto se traduce en coraje para avanzar decididamente sin esperar órdenes, especialmente cuando el camino no está completamente claro.

- **Foco:** *Scrum* no permite distracciones. El proyecto es la actividad más importante del equipo y todos los demás implicados, y debe mantenerse la atención concentrada en él. La dedicación plena al proyecto es una de las consecuencias.
- **Predictibilidad:** El equipo debe adoptar un ritmo determinado, trabajar de una forma ordenada y disciplinada, y todo con el objetivo de acabar siendo predecibles. Asumiendo el cambio y la incertidumbre como componentes naturales del trabajo y no como inconvenientes, el objetivo es acabar siendo capaces de anticipar aproximadamente qué cantidad de trabajo puede realizarse en un periodo determinado.
- **Personas:** Los métodos ágiles se centran más en las personas que en los procesos o los métodos[17]. Podría definirse una metodología rica y compleja, llena de pasos, diagramas y documentos, pero sin la contribución decidida de las personas nunca llegará a buen puerto. Por eso, *Scrum* se centra sobre todo en las personas participantes o interesadas, en favorecer el flujo de comunicación entre ellas para lograr unas relaciones ricas y fluidas.

Esta es la lista de valores de los autores, pero no es la única. Pueden encontrarse otras listas más resumidas o más extensas y con valores que no se contemplan aquí, pero no porque no sean importantes. Por ejemplo, la Revisión del trabajo, que se traduce en ser transparentes y hacer que el trabajo pueda ser objeto de escrutinio como una forma de favorecer la calidad; Colaboración, implícita en buena parte de los valores anteriores e imprescindible para el trabajo en equipo; Contar con el cliente, que en *Scrum* no es una figura ajena y lejana, sino que se integra perfectamente en el trabajo del equipo; Trabajo en iteraciones o ciclos cortos, como medio para hacer que afloren cuanto antes los posibles problemas; Priorización, porque hay que ser conscientes de qué es realmente importante y saber asignarle la relevancia que merece; Trabajo en equipo, uno de los valores supremos e implícitos de *Scrum*; Generosidad, muy relacionado con el anterior; Comunicación, básica para una forma de trabajo tan dependiente de las personas; y Capacidad (y disposición) de aprendizaje, que refleja la necesidad de mejora continua de *Scrum*.

Los roles

Ya se han mencionado varios papeles especializados en el trabajo con *Scrum*. A continuación, se verá qué es lo que hacen, cuáles son sus responsabilidades y cómo se relacionan entre sí.

Nota:

Un rol que engloba a varios es el de "Equipo Scrum", que incluye al Product Owner, al

Scrum Master y al equipo de trabajo.

Figura 2.3. Los roles de Scrum.

El cliente

En realidad, en *Scrum*, más que de cliente se debe hablar de los *stakeholders*, es decir, todas las personas y organizaciones que tienen algún interés en el trabajo o proyecto que se va a realizar. Por ejemplo, una Web de comercio electrónico puede ser encargada por el departamento de IT de una empresa a otra distinta, y sería en ese sentido el cliente, pero en el proyecto tienen mucho que decir las áreas de explotación, marketing, financiero, logística, etc. Por ello, cuando se habla en este libro del "cliente" se hace referencia al conjunto de los interesados en llevar a buen término el proyecto.

Pues bien, aunque no parezca que se le dedica mucha atención en *Scrum*, en realidad juega uno de los papeles más importantes, al ser quien tiene una necesidad que plantear al equipo, y cuenta con los recursos (generalmente económicos) para construir la solución. Es decir, es dueño de los requisitos y de los recursos.

Los *stakeholders*, en su conjunto, no tienen por qué entrar a formar parte de proceso *Scrum*, pero es conveniente que lo conozca y esté familiarizado con su terminología y forma de operar. A fin de cuentas, se está hablando de dejar la aproximación tradicional de desarrollar proyectos, en la que se partía de unos requisitos, supuestamente, completos y detallados, que al cabo de un tiempo se convertían en un producto funcional sin mucho contacto entre cliente y equipo. En su lugar, se ofrece al cliente una nueva forma de trabajar, en la que no es necesario tener perfectamente cerrados y delimitados los requisitos y con la

que podrá ir revisando resultados parciales a intervalos regulares.

El papel de cliente en *Scrum* implica ante todo dos grandes tareas: proporcionar requisitos y validar resultados. Es decir, definir el producto que se quiere construir y examinar cuidadosamente los resultados intermedios (y finales) que ofrezca el equipo para dar sus comentarios, correcciones y sugerencias, su *feedback*.

El Product Owner

El PO, *Product Owner* o Dueño del Producto, forma parte del cliente y actúa como intermediario entre este y el equipo. Por ello, debe ser capaz de hablar el lenguaje de negocio o de los requisitos de cliente y estar familiarizado con los métodos y conceptos empleados por el equipo. Se trata de un papel fundamental, capaz de transmitir al equipo los requisitos y reacciones del cliente, actuando como él cuando surjan dudas y cuestiones sobre el producto que se está construyendo. Puede ser difícil contar con todos los *stakeholders*, pero el *Product Owner* debe estar siempre disponible para el equipo.

El PO es el representante del cliente (o de quien necesita o encarga el nuevo producto, proyecto, actividad...) ante el equipo, por lo que está enfocado prioritariamente a los aspectos de negocio, no a los técnicos, y, por ello, es el encargado de transmitir la visión de producto al equipo. Es responsable del éxito o fracaso del producto y de la rentabilidad del proceso.

Con respecto al contenido y desarrollo del trabajo, el PO tiene varias responsabilidades importantes. Para empezar, es quien fija las fechas clave de las distintas entregas y de priorizar los distintos requisitos. Y la forma de ejercer ese control es manteniendo al día el *Product Backlog*. Esto significa que todos los elementos contenidos en él (temas, épicas e historias) estarán correctamente descritos, con unas condiciones de aceptación comprensibles y debidamente priorizadas. Para poblar este *Backlog*, el PO debe mantener un contacto continuado con el conjunto de los *stakeholders*, comprender perfectamente sus necesidades, hablar su “idioma” y traducir sus requisitos en elementos del *Backlog*, es decir, en el lenguaje del equipo de trabajo.

Por último, el PO es quien acepta o rechaza las entregas del equipo por medio de las revisiones del trabajo realizado en cada *Sprint* y entrega.

El *Product Owner* mantiene una relación estrecha con el *Scrum Master* y asiste al menos a dos reuniones muy importantes dentro del ciclo de trabajo de cada *Sprint*: la *Review* y el *Planning*.

En la revisión o *Review*, el *Product Owner* actuará en representación del cliente cuando este no pueda asistir y examinará el trabajo del equipo para darlo por válido o no. El trabajo realizado se revisa de acuerdo con los criterios de aceptación expresados en la descripción de cada historia, bien por medio de una explicación, bien examinando el producto (por ejemplo, un documento, o unos planos), bien demostrándolo (lo que es especialmente importante cuando se trabaja en el desarrollo de software).

En la planificación o *Planning*, el PO, que previamente habrá incluido y priorizado las distintas historias en el *Backlog*, debe dar todas las explicaciones y aclaraciones que precise el equipo, así como fijar unos criterios claros de aceptación del trabajo.

El *Product Owner* es, ante todo, un intermediario entre el mundo del negocio y el equipo de trabajo, lo que hace que tenga que conocer los condicionantes de los dos. Además, tiene una gran capacidad de decisión definiendo y aceptando el trabajo, así como delimitando el entorno (recursos, fechas) en el que se desarrollará.

El Scrum Master

El otro gran papel diferenciado en el mundo *Scrum* es el del SM o *Scrum Master*. Ante todo, hay que dejar claro que el papel de SM no es el de un jefe de proyecto; con *Scrum*, ese concepto desaparece. Si hay una palabra que define al *Scrum Master* esa es "facilitador". Su principal cometido es mejorar la productividad del equipo y eso lo consigue aislándolo de interferencias externas, eliminando impedimentos y procurando que fluya la comunicación y la colaboración. Además, es responsable de introducir y fomentar las prácticas Agile.

Como facilitador trabaja muy cerca del Product Owner, pero, por encima de todo, trabaja muy cerca del equipo, a quien protege de interferencias externas, forma en técnicas Scrum, orienta y vela por alcanzar la máxima calidad y productividad en el proceso,

El SM supervisa el *Backlog*, asegurándose que todas las historias estén correctamente descritas, priorizadas y estimadas. También es el verdadero supervisor de todo el proceso, por lo que, entre otras cosas, ayuda al equipo a evaluar el resultado del trabajo, analizando la velocidad del equipo, velando por la calidad y siguiendo de cerca el proceso, su principal objetivo.

Es también el intermediario entre el mundo exterior (PO, otros equipos) y el equipo de trabajo. Esta tarea forma parte de su misión de fomentar la productividad protegiendo al equipo de interferencias externas.

Pero, por encima de todo, el SM es quien debe encargarse de fomentar las buenas prácticas, la formación y la aplicación de nuevas herramientas. Sin embargo, su objetivo último es hacerse prescindible y permitir que un equipo suficientemente maduro y entrenado sea capaz de auto-organizarse y funcionar sin la figura del SM.

Hay mucha literatura sobre el papel del *Scrum Master* y sobre qué es lo que hace de una persona un buen *Scrum Master*. Una buena síntesis es la lista de los seis atributos principales que selecciona Mike Cohn[18]:

- **Responsable:** El *Scrum Master* es responsable de que el equipo aplique correctamente *Scrum*, eso sin la coerción que implica una autoridad formal. Un buen *Scrum Maste*r es como un director de orquesta: los verdaderos protagonistas son los músicos, pero el director se encarga de que el talento y trabajo de todos ellos funcione conjuntamente de

la mejor manera posible.

Figura 2.4. El Scrum Master es un facilitador, como un director de orquesta.

- **Humilde:** El *Scrum Master* no debe restar protagonismo a los miembros del equipo. No debe destacar ni distinguirse. Convence por medio del ejemplo y la inspiración.
- **Colaborador:** Si *Scrum* se basa en la colaboración, el *Scrum Master* debe ser el abanderado de este principio, fomentando la colaboración y buscando la forma de frenar las actitudes contrarias y egoístas.
- **Comprometido:** Formalmente puede parecer que es el *Product Owner* o el equipo quienes tienen los compromisos más fuertes con el éxito del trabajo. Sin embargo, es imposible que el proyecto llegue a buen puerto si el propio *Scrum Master* no adopta una actitud comprometida con el proyecto, sus fines y la forma de llevarlo a cabo.
- **Influyente:** Al no contar con autoridad formal, el *Scrum Master* no tiene más remedio que convencer por medio del ejemplo y de recurrir a su capacidad para persuadir a otros. Esto obliga a que un buen *Scrum Master* deba dotarse de unas armas más políticas que metodológicas, técnicas o científicas.
- **Entendido:** Precisamente, una forma de influir en otros es desplegando un conocimiento erudito, tanto de *Scrum* y aspectos metodológicos como del campo de aplicación en el que se esté desarrollando el trabajo.

El equipo

Llega el turno de los verdaderos protagonistas de *Scrum*: los componentes del equipo de trabajo. No tienen un rol específico asignado, pero sin ellos es imposible llevar a buen puerto el trabajo, proyecto o actividad donde se estén aplicando los principios de *Scrum*.

La forma tradicional de desarrollar cualquier trabajo se basa en la jerarquía, la autoridad formal y la estructuración. Frente a ella, el equipo en *Scrum* se auto-organiza, tiene la responsabilidad final por el éxito del trabajo y es capaz de asumir cualquier actividad dentro de las necesarias para desarrollar el proyecto.

Estas características imponen ciertos límites al equipo. Por ejemplo, de tamaño: hay un límite máximo de 10 a 12 personas a partir del cual debe sopesarse la división y el paso a una versión extendida de *Scrum* (o *Scrum of Scrums*). Los miembros del equipo deben tener un elevado grado de compromiso, lo que es especialmente cierto a la hora de planificar, momento en el que hay que establecer un acuerdo entre las demandas del *Product Owner* y lo que va a ofrecer el equipo al final de cada *Sprint*.

Figura 2.5. Scrum significa “melé”, una jugada que requiere de la colaboración y el empuje de todo el equipo.

Un equipo en *Scrum* debe ser capaz de auto-organizarse. Esto le diferencia de un equipo tradicional donde un jefe de proyecto asigna a cada persona tareas concretas y fijas. El equipo *Scrum* deberá contar con personas capaces de asumir cualquier actividad dentro del trabajo, lo que no siempre es posible, sobre todo si se requieren perfiles muy especializados para ciertas partes del proyecto.

Con la ayuda del *Scrum Master*, el equipo deberá ser capaz de seguir la evolución de su productividad y mejorarla. También con la ayuda del SM, pero, como parte de su compromiso, los miembros del equipo deben entrar en un proceso continuo de mejora, tanto en lo que se refiere a conocimientos y habilidades como a la aplicación de las mejores prácticas en su trabajo.

Ese compromiso se traduce también en la necesidad de una dedicación continuada y exclusiva de los miembros del equipo al trabajo, salvo que sea preciso contar puntualmente con perfiles muy especializados.

El coach

Se trata de una figura conveniente, aunque no obligatoria. El cometido genérico del proceso de *coaching* es el de formar, dirigir y aconsejar al equipo en las prácticas *Scrum*. Por ello, el *coach* actúa ante todo como formador y mentor del equipo. Su labor es ayudar a todos los implicados en el proyecto a aplicar de la mejor manera las técnicas *Scrum*, a resolver sus conflictos y enderezar el camino cuando las cosas se tuercen.

Para ello, asiste a las reuniones, revisa resultados y herramientas, habla con los implicados y todos los que estén directa o indirectamente involucrados en el proceso, hasta formarse una imagen fiel de cómo se está aplicando *Scrum* y de los principales problemas que pueda encontrarse el equipo. Con todo ello, elaborará unas guías de actuación que irá aplicando continua y progresivamente en el tiempo, haciendo un seguimiento cercano de su aplicación y resultado.

El *coach* no es un mero consejero: se involucra, participa y hace todo lo posible para motivar al equipo y conseguir sus objetivos.

No siempre hay un *coach* por equipo. Normalmente, en empresas u organizaciones suficientemente grandes pueden contar con *coach* que atienden a varios equipos.

El proceso

El proceso *Scrum* se divide en dos grandes etapas: la preparación o *Sprint* 0, y las sucesivas iteraciones o *Sprints*, agrupadas en entregas o *Releases*.

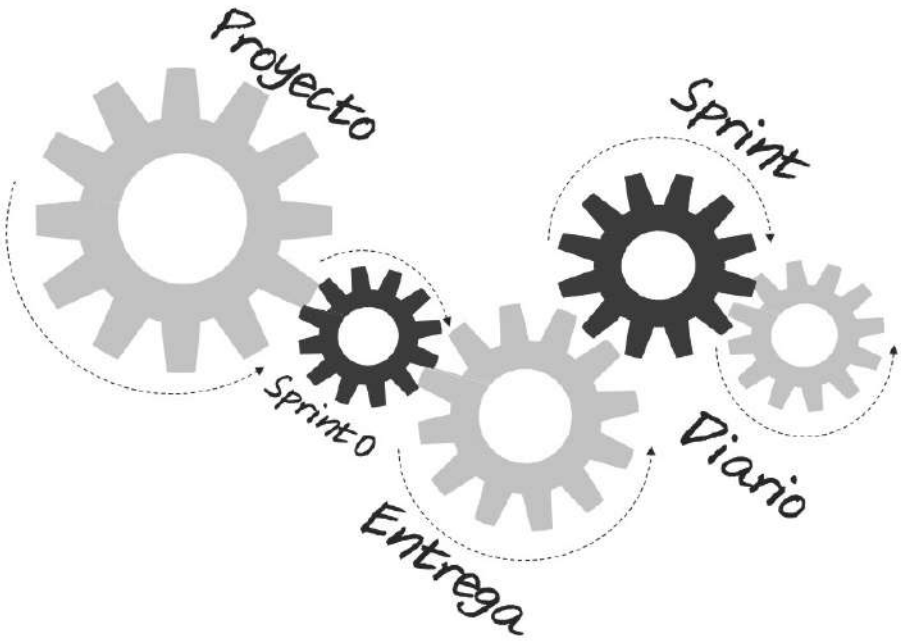


Figura 2.6. Los ciclos del proceso Scrum.

Sprint 0

Es el momento en el que se definirá la misión del trabajo que se va a realizar, así como las herramientas que se usarán y el equipo que trabajará con ellas para alcanzar el objetivo final del trabajo.

El *Sprint* 0 es una etapa muy importante y de duración variable, pero no indefinida. No debería escatimarse tiempo ni esfuerzo a desarrollarlo: hay que verlo como una inversión que ahorrará muchos problemas y quebraderos de cabeza futuros.

El *Sprint* 0 tiene muchos objetivos, aunque podemos destacar dos por encima de los demás: definir las condiciones y el contenido del trabajo o alcance.

Las condiciones que van determinar el alcance del proyecto incluyen los recursos (financiación, personas, herramientas) necesarias para desarrollarlo, así como el marco temporal con la distribución de entrega de resultados. Al final del *Sprint* 0 debe quedar claro si el proyecto es viable y si cuenta con los medios y el apoyo necesarios para llegar a buen fin.

El otro gran objetivo del *Sprint* 0 es definir el contenido del trabajo y eso se consigue por medio de una primera versión del *Product Backlog*, que contiene la lista de grandes trabajos y tareas que van a desarrollarse en el transcurso del proyecto.

El *Product Backlog* va a recoger la visión de los requisitos principales del proyecto: principales funcionalidades o resultados, productos generados, definición de la interacción con el usuario, si la hay, entre otros.

El *Product Owner* es el gran protagonista de esta etapa: debe conseguir los apoyos y recursos necesarios para llevar a cabo el trabajo, seleccionar equipo y *Scrum Master*, acordar alcance y fechas con el cliente y, en general, establecer las condiciones para llevar a buen fin del proyecto que se quiere desarrollar.

Sprints

Una vez arranca el proyecto, su desarrollo se divide en iteraciones, etapas o *Sprints*. Cada una de estas etapas sigue una secuencia muy precisa de reuniones que tiene como principal cometido garantizar el cumplimiento de los compromisos del equipo de trabajo y el *Product Owner*.

Una de las decisiones clave del *Sprint* 0 es elegir la duración que inicialmente tendrá cada *Sprint*. Hay muchas opiniones al respecto, aunque lo más habitual es que oscile entre 1 y 4 semanas, generalmente 2 o 3. Además, hay que fijar un calendario de *releases* o entregas, es decir, los momentos en los que, pasado un número determinado de *Sprints* se va a ofrecer al cliente o destinatario del trabajo, un resultado parcial antes de completarlo.

Si el *Product Backlog* recoge el conjunto de los trabajos que se van a realizar para alcanzar los requisitos del cliente, hay un subconjunto, el *Sprint Backlog*, que contiene aquellos que se van a llevar a cabo durante la duración de un *Sprint* determinado. El contenido de este *Backlog* es un compromiso entre las necesidades del cliente expresadas por medio del *Product Owner* y la capacidad de producción del equipo de trabajo. El alcance del trabajo de cada *Sprint* se define a partir de los objetivos que fije el *Product Owner*, de la priorización que haya hecho de las tareas en el *Product Backlog* y del compromiso que haga el equipo acerca de aquellas que finalmente llevará a cabo.

El *Sprint* tiene tres etapas diferenciadas y marcadas por una serie de reuniones: arranca con *Sprint Planning* (aunque previamente el *Product Owner*, con la ayuda del *Scrum Master*, haya revisado y priorizado el *backlog*), se desarrolla en el tiempo que se haya fijado con reuniones diarias o *Daily Meetings* y termina con una reunión de *Review* y otra de Retrospectiva. Estos son los otros hitos del proceso que vamos a revisar a continuación.

Sprint Planning

Al comienzo de cada *Sprint* o iteración, hay que dedicar un tiempo a planificar el trabajo que se va a hacer. Antes de la reunión, el PO (que podría contar con el apoyo del *Scrum Master*) revisa el *Product Backlog* para asegurarse de que están incluidas todas las historias de usuario (requisitos en lenguaje de negocio en que se divide el conjunto de la actividad) que le gustaría ver incluidas en la próxima iteración, todas ellas están correctamente descritas y priorizadas.

Ese *Backlog* priorizado es el punto de partida para el trabajo de planificación. La reunión tiene que terminar con unos objetivos claros: conseguir la lista de historias a la que se compromete el equipo a realizar, o *Sprint Backlog*; un propósito global y claro para el *Sprint* que sugiere el *Product Owner*; el compromiso firme del equipo para realizar las historias que ha seleccionado; la estimación que hace el equipo de la complejidad o esfuerzo preciso para desarrollar cada una de las historias, lo que además da la velocidad o capacidad total de trabajo del *Sprint*; y que todos los miembros del equipo entiendan el contenido y alcance de cada una de las historias que propone el PO.

El *Sprint Planning* se divide a su vez en dos etapas. En la primera, con el concurso del *Product Owner*, se revisa cada historia del *Product Backlog* siguiendo el orden de la priorización. El PO las explica, detalla los criterios de aceptación que servirán para validar el trabajo realizado y atiende a las preguntas, dudas y aclaraciones que plantee el equipo. Para cada una de esas historias de usuario, el equipo hará una estimación, lo que servirá para delimitar el alcance del *Sprint* una vez se conoce cuál es la velocidad o capacidad habitual del equipo.

Esta estimación la hace el equipo usando algunos de los muchos sistemas ideados para ello. La estimación se valorará como esfuerzo necesario para realizar cada historia de acuerdo con el equipo. Una técnica muy habitual es la llamada *Planning Poker*, en la que se usan unas cartas marcadas con números para que se asigne una valoración a cada historia partiendo de lo que cada miembro del equipo vote usando su criterio y obteniendo el valor final como una valoración media, un compromiso entre todos, o cualquier otro criterio establecido de previo acuerdo.

Un elemento muy importante de cada historia de usuario es el criterio de aceptación que define qué es lo que permite determinar si la historia se ha completado o no.

Cuando se ha alcanzado un número suficiente de historias como para ocupar el trabajo del equipo durante el *Sprint*, se procede a su división en partes más manejables o tareas. Este trabajo lo hace el equipo de forma autónoma y permitirá desmenuzar cada historia en un conjunto de tareas más manejables y expresadas en el lenguaje del dominio de trabajo (arquitectura, desarrollo software, marketing o el que aplique en cada caso). El equipo incluirá para cada tarea una definición de completitud, definición de hecho o *Definition of Done*, que es más concreta y describe los criterios que permitirán determinar desde un punto de vista técnico si se han completado o no.

El conjunto de historias de usuario y las tareas en las que se dividen conforman el *Sprint Backlog*, la definición del trabajo que se va a desarrollar durante la iteración o *Sprint*.

Daily Meeting o Scrum diario

Una vez arranca el trabajo, la mecánica es simple y repetitiva: cada miembro del equipo selecciona la siguiente tarea que va a abordar de acuerdo con el resto de los miembros (para

evitar que, por ejemplo, dos personas seleccionen la misma) y siguiendo la priorización del *Backlog* establecida por el PO. Usando las herramientas que se hayan seleccionado para el trabajo, marca la tarea para indicar que está en curso, va reflejando su evolución e informa cuando se completa.

Para evitar que se pierda la necesaria sincronización entre el trabajo del equipo, mantener el ritmo y "tensión" y para fomentar la comunicación interna, *Scrum* propone el *Daily Meeting* o *Scrum* diario. Se trata de una reunión diaria (salvo que se acuerde otra frecuencia, y eso en circunstancias justificadas) en la que participan todos los miembros del equipo y el *Scrum Master*, y a la que puede asistir el PO. En ella, cada miembro del equipo detallará qué actividades ha realizado, cuáles son las que piensa abordar a continuación y qué impedimentos hay para continuar su trabajo.

El SM participa como facilitador: no dirige la reunión dando a paso a cada miembro del grupo, sino que su cometido es el de "empujar" y facilitar su desarrollo. También es muy importante que tome nota de los impedimentos que haya para seguir su solución.

La *Daily Meeting* es una reunión muy breve, en la que deben dejarse de lado las discusiones de detalle para que sean tratadas en otro momento por las personas directamente implicadas.

Truco:

Un riesgo habitual es que las reuniones diarias se alarguen más de lo conveniente. El Scrum Master deberá buscar formas de dinamizarlas: esto puede desde simplemente insistir en recalcar el contenido, duración y alcance, hasta llegar a forzar que se hagan de pie, para que la incomodidad impida que se extiendan sin razón.

Las *Daily Meetings* permiten tomar el pulso del trabajo y detectar en momentos muy tempranos problemas que de otro modo podrían crecer y convertirse en obstáculos serios. Con ellas, se garantiza un conocimiento actualizado del estado de los trabajos por parte de todos los miembros del equipo, lo que es una forma de incrementar su grado de compromiso, la sincronización y la auto-organización.

Review o revisión

El final de cada *Sprint* o iteración está marcado por una revisión del resultado y de la forma de alcanzarlo. De lo primero se encarga la Revisión o *Review*.

Se trata de una reunión que se hace con la participación del *Scrum Master*, del conjunto del equipo, del *Product Owner* (es decir, el equipo *Scrum*) y de los *stakeholders* (clientes, usuarios y quien tenga interés y pueda aportar valor al producto o proyecto).

En ella, se repasa el trabajo realizado a través de los resultados obtenidos (programas,

documentos, diseños o el formato que se haya acordado). En este punto es donde destaca la relevancia de los criterios de aceptación: la descripción de los resultados esperados y por qué se ha alcanzado o no el objetivo de cada una de las historias incluidas en el *backlog*.

Siguiendo el orden de priorización de las historias incluidas en el *Sprint Backlog*, los miembros del equipo implicados van exponiendo el resultado alcanzado. Esta exposición de resultados debe ser de la forma más gráfica y directa posible, de manera que, si se puede demostrar el resultado, se haga así antes que enunciarlo o describirlo.

A la vista de los resultados, el *Product Owner* y el conjunto de los *stakeholders* (si están presentes) determinan si se han alcanzado o no los objetivos propuestos inicialmente y expresados en los criterios de aceptación. Si no es así, se identifican claramente los elementos pendientes de completar para que sean abordados en un próximo *Sprint*, salvo que se cambie la prioridad.

De esta reunión va a salir la lista de historias de usuario completadas y pendientes. Las primeras sumarán a la hora de calcular la velocidad real, que solo incluye los trabajos completamente realizados. Aunque solo quede pendiente una mínima tarea de las muchas en las que se haya podido dividir una historia, la historia de usuario se considerará incompleta. Esta es una forma de forzar una inclinación hacia completar el trabajo y no arrastrarlo *sprint* tras *sprint* sin poder cerrarlo.

Al final de la revisión de los resultados del *Sprint*, el PO decidirá si el objetivo general propuesto para el *Sprint* se ha alcanzado o no.

Retrospectiva

Para muchos, esta es la reunión más importante de *Scrum*, la que define el espíritu de esta forma de desarrollar proyectos. Si uno de los principios de *Scrum* es la mejora continuada, la retrospectiva es el medio para analizar la forma en la que se hacen las cosas y cómo mejorar el conjunto del proceso. Requiere la participación activa de todo el equipo, ya que es una reunión pensada para él, para examinar su funcionamiento y mejorar su trabajo. Cuenta con el *Scrum Master* como facilitador y encargado de seguir el cumplimiento de los principios de *Scrum* (aunque en ocasiones es positivo que sea otro miembro del equipo). En principio se trata de hacer un análisis del proceso, no de los requisitos y resultados de negocio, pero, si el PO afecta o se ve afectado, es conveniente que asista.

Figura 2.7. La retrospectiva es un momento para revisar el camino andado y mirar hacia el futuro.

La retrospectiva se centra en analizar la forma de trabajar de manera crítica, destacando los puntos fuertes y débiles del equipo e identificando formas de mejorar. Esto supone una revisión de la evolución concreta del *Sprint* y sus resultados: qué objetivos tenía y si se han alcanzado, las dificultades encontradas, dónde se ha fallado, también qué se ha hecho bien y qué aspectos positivos hay que destacar y conservar.

Un punto básico de la retrospectiva es la valoración de la velocidad del equipo. Como facilitador, el *Scrum Master* se encarga de recopilar la estimación previa y contrastar el valor obtenido finalmente. Hay muchas formas de medir esa velocidad, todas ellas subjetivas, ya que no hay medida objetiva posible (excepto para trabajos repetitivos, muy definidos y con poca incertidumbre). Una forma muy habitual es usar la estimación de complejidad que hacen los miembros del equipo en forma de puntos de historia o *Story Points*. La suma de los puntos de todas las historias contempladas inicialmente en el *Sprint* da la velocidad estimada. A la hora de conocer la velocidad realmente alcanzada, solo se cuentan los puntos de aquellas historias que se han considerado completamente cerradas en la *Review*.

La suma de los puntos de todas las historias de usuario concluidas (de acuerdo con los criterios de aceptación y el juicio del *Product Owner*) da la velocidad alcanzada en el *Sprint*. Con el tiempo, los equipos tienden a tener un valor de velocidad estable que facilita la estimación del alcance de cada *Sprint*. Si se repiten con la misma duración y mismas personas, ese valor tiende a ser predecible. La evolución de esa velocidad es un indicador de la mejora o degradación del proceso de desarrollo del trabajo del equipo.

Otro aspecto que hay que cuidar en la Retrospectiva es la identificación de los riesgos e impedimentos que amenazan el trabajo, para ayudar a su resolución. Es igualmente importante destacar todos los éxitos del trabajo para que sirvan de referencia y ejemplo

futuros.

En la Retrospectiva, cada miembro del equipo tiene la oportunidad de expresar su visión crítica sobre qué se está haciendo bien (y merece la pena conservar y potenciar), qué se está haciendo mal y de qué manera mejorar la forma de trabajar.

Existen muchas técnicas para recoger la visión de los miembros del equipo. Puede ser una recopilación de sugerencias sin orden o una ronda para garantizar que nadie se queda sin dar su punto de vista. Puede sintetizarse con facilidad en tres categorías (qué se hace bien, qué mal y qué hay que corregir) o distinguirse otras muchas. En cualquier caso, al final de la retrospectiva debe quedar claro qué puntos de acción concretos hay que abordar para seguir un proceso continuado de mejora.

Esos puntos de acción deben ser asumidos y compartidos por todo el equipo como parte de su compromiso para mejorar su forma de trabajo y productividad, y su aplicación es una de las prioridades que deberá seguir el *Scrum Master*.

Nota:

Las retrospectivas deben ser un momento para la colaboración positiva que permite encontrar soluciones entre todos los miembros del equipo. No deben ser nunca un proceso de búsqueda de culpables o una reunión cuyo objetivo sea el de destacar problemas. No hay que caer en la autocomplacencia, pero tampoco forzar la aparición de problemas que quizá no sean reales.

Periodo de mejora

El *Improvement Period* o periodo de mejora es una etapa opcional tras el *Planning*, cuyo propósito es reflexionar y aplicar cambios que mejoren el proceso. La idea es que el equipo dedique ese tiempo a revisar y mejorar la forma de trabajo, no a continuar con el desarrollo del proyecto.

Se trata de un concepto que no todos los autores contemplan y que no es habitual encontrar entre los equipos que aplican *Scrum*.

Refinamiento

La preparación del *Backlog* o Refinamiento (anteriormente conocida como *Grooming*) es una actividad previa a la planificación. En ella, se revisa el repositorio global del proyecto para actualizarlo, completarlo y asegurar que la priorización es la adecuada.

Conceptos y entidades Scrum

Además de los roles y las etapas, hay otros conceptos y entidades que definen la forma de trabajar con *Scrum*.

Entidades

En *Scrum*, el trabajo se subdivide en una serie de unidades manejables, capaces de ser abordadas en cada *Sprint*. Esas unidades son las historias de usuario o *User Stories*. Sin embargo, estas unidades pueden resultar muy grandes para ser tratadas individualmente por los miembros del equipo, o muy pequeñas como para definir los grandes bloques en los que se divide un proyecto. Por ello, existen otras entidades que forman una jerarquía. Estos niveles expresan también el grado de incertidumbre (menor cuanto más cercanas al equipo). Vea cuáles son, empezando por las más pequeñas y detalladas y avanzando hacia las grandes divisiones del trabajo, en una secuencia creciente de incertidumbre:

- Una **Tarea** es un trabajo concreto, idealmente realizado por una persona dedicando entre medio y tres días, es decir, siempre dentro de unos límites muy concretos. Las tareas se expresan en el lenguaje del dominio técnico del trabajo (construcción naval, programación de teléfonos móviles, urbanismo, marketing...), no en el lenguaje de negocio del cliente; durante la segunda etapa del *Sprint Planning*, los miembros del equipo las crean subdividiendo las historias de usuario en unidades más manejables.
- Las **Historias de Usuario** son la definición en lenguaje del negocio que hace el *Product Owner* de los requisitos del trabajo. Esos requisitos se analizan y desmenuzan en unidades más pequeñas, abordables en el curso de un *Sprint* por el equipo. En el proceso de estimación del *Sprint Planning* es cuando el equipo determina la complejidad de la historia, de acuerdo con la explicación que haga el PO y si es posible abordarla o no en el transcurso del *Sprint*. Se puede fijar un valor límite dependiente de la velocidad media del equipo, a partir del cual se supone que la complejidad de la historia excede al *Sprint* y debería subdividirse o variar su alcance.
  Las historias de usuario son la unidad básica de cuenta del *Sprint*. La suma de los puntos de historia del *Sprint* determina la velocidad del equipo y solo se consideran completadas cuando se han realizado todas sus tareas y el *Product Owner* ha aceptado sus resultados por completo.
- Las **Épicas** son agrupaciones de historias de usuario que definen grandes bloques operativos dentro de un proyecto. Pueden ser el sistema de facturación en una tienda electrónica, la estructura de un edificio, la operativa en oficina de una nueva cuenta corriente, el alcantarillado en un proyecto de urbanización, el motor de un vehículo, la

instalación eléctrica en el diseño de una fábrica, etc.

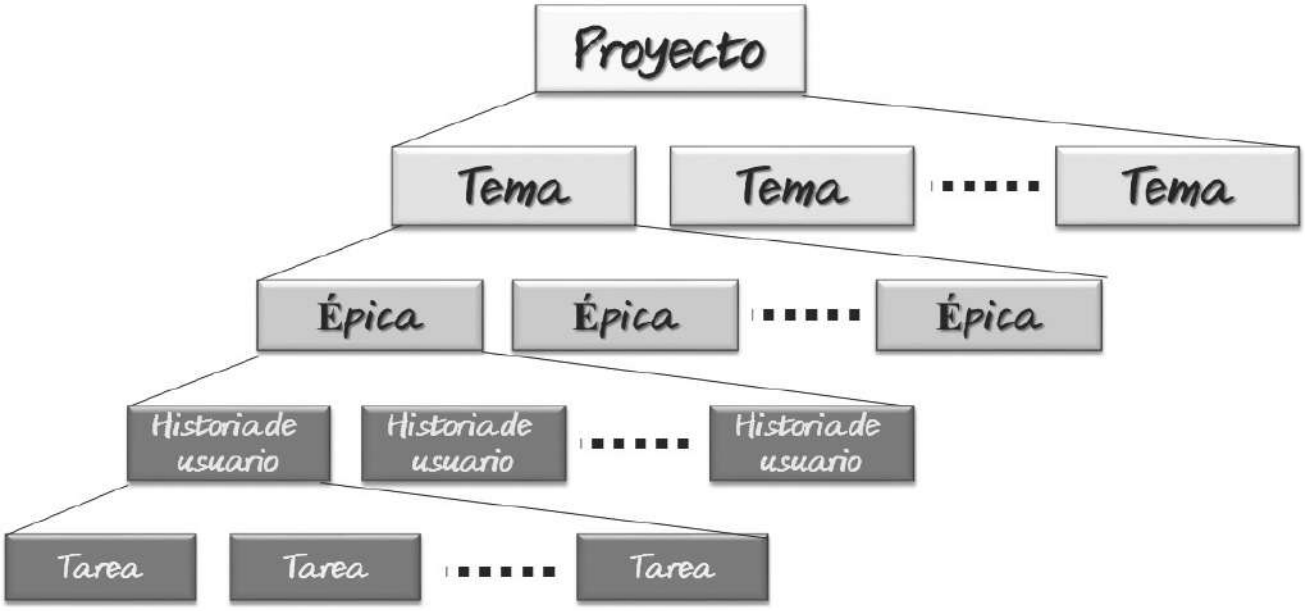


Figura 2.8. La jerarquía de entidades Scrum.

A veces, las épicas no son suficientes y hay que crear agrupaciones aún mayores, como los Temas, que definirían las grandes ideas o requisitos del cliente. A veces, por simplificación, y porque el proyecto no es tan grande, se simplifica usando solo épicas y no temas. En el diseño de un barco, los temas podrían ser las grandes características que lo definen como capacidad de carga, autonomía, calado y dimensiones máximas, tripulación o coste de combustible. Una épica podría ser el diseño de la planta motriz con unos requisitos concretos de consumo, potencia, peso, volumen, tecnología... Una historia dentro de ella sería el diseño concreto de la alimentación del motor, mientras que una tarea podría ser una serie de pruebas de presión para seleccionar la tubería (material, sección, aislamiento) más apropiada para llevar el combustible al motor.

En ocasiones, el equipo necesita incluir trabajos necesarios para el proyecto pero que no forman parte de los requisitos de usuario. Eso no supone ningún inconveniente, siempre y cuando se argumente su necesidad y se negocie con el PO la priorización adecuada. Este tipo de historias de usuario pueden incluir necesidades de formación, de evaluación de herramientas y técnicas o de creación de componentes comunes (una librería software, una bancada de motor, armar una impresora 3D para prototipado) necesarios para continuar el trabajo, aunque no se reflejen en los requisitos de usuario.

Artefactos

El trabajo que se va a realizar, definido de acuerdo con la jerarquía anterior, se compone

de múltiples elementos que deben estar al alcance de todos los participantes en el proyecto y que reciben el nombre de artefactos.

Los almacenes, repositorios, pilas o *Backlogs* son artefactos (generalmente sustentados por herramientas) en los que se guardan todos los elementos que definen el trabajo.

En *Scrum* se definen al menos tres *backlogs*; si bien, dentro de su principio de flexibilidad y en función de las necesidades del trabajo, podrían definirse otros. Los tres que prescribe *Scrum* son:

- Product backlog **o pila de producto:** Se trata de la lista que contiene todas las entidades que definen el trabajo del proyecto. Gestionado por el *Product Owner*, refleja la visión del cliente, por lo que las entidades que contiene se refieren a los requisitos: épicas, temas e historias de usuario. El PO lo ordena de acuerdo con la prioridad de las necesidades de negocio. La unidad de medida son los *Story Points* o puntos de historia, que reflejan la medida del esfuerzo asignada por el equipo en el momento de la planificación.

  La descripción de cada historia y de los criterios que definen su cumplimiento es una parte crítica de la creación del *Product Backlog*. Por eso, su creación es un trabajo muy importante dentro del desarrollo de *Scrum* y, aunque es posible añadir y modificar su contenido, es fundamental partir de un conjunto lo más amplio y detallado posible, que describa al menos todos los elementos fundamentales del proyecto. Es una buena práctica usarlo tanto para recoger el trabajo futuro, pendiente de realizar, como para guardar un rastro de las actividades completadas.
- **Sprint Backlog o pila de Sprint:** Se trata de la lista de los trabajos que se van a realizar en una iteración o *Sprint* determinado. Por ello, contiene las historias de usuario y, sobre todo, las tareas que el equipo, que es quien gestiona este *backlog*, ha identificado en el momento de la planificación de detalle.

  La unidad de referencia es el tiempo, ya que las tareas concretas se pueden estimar con más exactitud. También la descripción es más concreta, desglosando las acciones que deben completarse en cada tarea, y no en forma de requisitos, como ocurre con las historias de usuario.

  El *Sprint backlog* se puebla a partir del **Product Backlog**, seleccionando historias en función de la priorización hecha por el *Product Owner*. El equipo estima cada historia, rechazando para que sean divididas aquellas que excedan una complejidad determinada (generalmente indicada por un número alto de puntos, como 13 o más) y añadiéndolas al *backlog* hasta que se alcanza una suma de *Story Points* en las historias similar a la velocidad habitual del equipo.

  Cada historia seleccionada se divide a su vez en tareas, descritas en el lenguaje del dominio técnico del trabajo, pequeñas, detalladas y con una estimación de tiempo para su realización. Por regla general, se asume que cada tarea debe poder ser realizada por una persona.

- El Impediments backlog **o pila de impedimentos:** Es un repositorio que recoge todo aquello que impide alcanzar los objetivos del proyecto, lo que incluye también degradar o amenazar la calidad del producto final.
  Este repositorio es gestionado por el *Scrum Master* y se mantiene actualizado a lo largo del *Sprint* con todos los impedimentos que se detecten y que se manifiesten en la *Daily Meeting*. Estos impedimentos se priorizan para su resolución de acuerdo con el impacto que tengan en las actividades del proyecto.

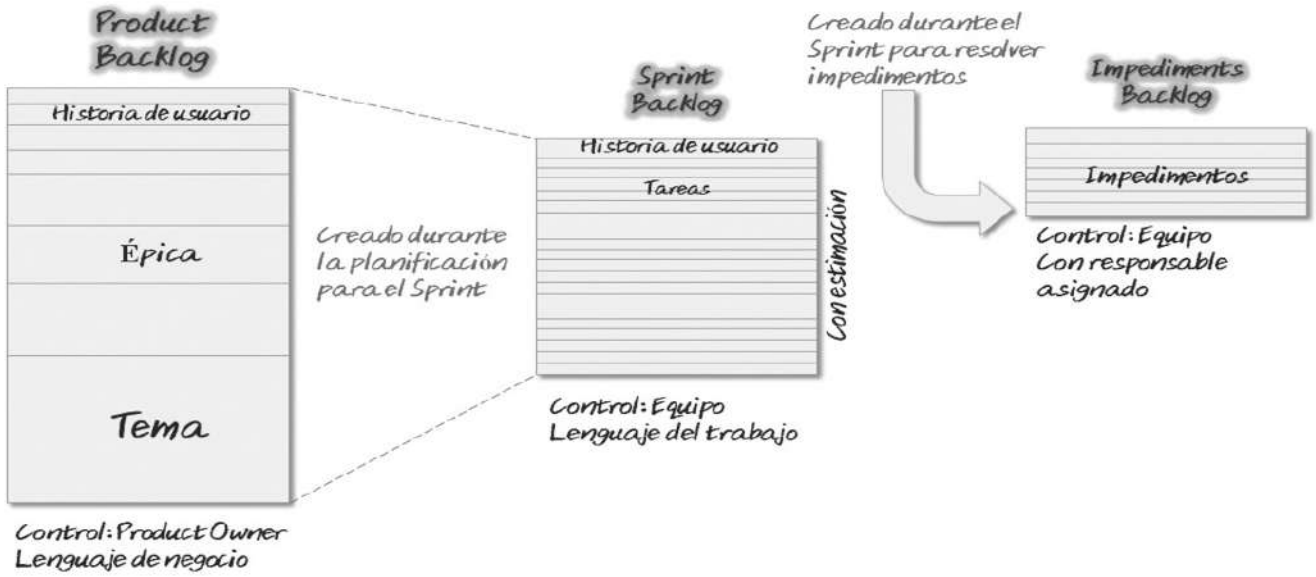


Figura 2.9. Los Backlogs de Scrum.

Los otros artefactos importantes de *Scrum* son las gráficas. Por encima de todas hay una que representa la evolución del trabajo: es la llamada *burn-down chart*. En ella, se representa en un eje el tiempo X (u horizontal) y en el otro Y (o vertical) la cantidad de trabajo que se debe realizar. Esa cantidad de trabajo puede medirse en puntos de historia o como la suma de entidades (tareas e historias de usuario) que se van a realizar a lo largo del *Sprint* o del conjunto del proyecto (lo que es menos habitual). Una línea recta marca la evolución ideal del trabajo: se sitúa al principio en el número total de puntos o entidades que se van a resolver y llega hasta el final del periodo señalando cero, o que se ha resuelto completamente.

Diariamente, el SM se encarga de actualizar la cantidad de trabajo pendiente. Puede ocurrir que en un día no se resuelva gran cosa (e incluso aumente, aunque no sea muy ortodoxo, cuando ha sido necesario añadir más tareas al *Sprint Backlog*), mientras que en otros se avance mucho. La distancia entre la evolución real y la ideal o estimada va a ir señalando si el trabajo se está realizando a un ritmo adecuado o si hay algo que corregir en el proceso. Se trata de una herramienta básica de consulta diaria.

Velocidad

La velocidad es un concepto de *Scrum* que define la capacidad del equipo para realizar sus actividades. Se trata de una herramienta para la estimación y medida del proceso que debe usarse con precaución. Esto se debe a su naturaleza arbitraria y subjetiva: los intentos de medir el trabajo han fracasado históricamente, incluso en campos tan controlables como el desarrollo de software (parece fácil cuando se pueden contar, por ejemplo, las líneas de código). Hay que tener en cuenta que, cuando hablamos de la dificultad de medir un trabajo, nos referimos a aquellos no repetitivos con un grado de incertidumbre alto, que implican creatividad e innovación.

En *Scrum*, la medida de la velocidad, o capacidad de trabajo del equipo de trabajo, se basa en la estimación que hace el propio equipo de la complejidad (e incluso del esfuerzo). Esa estimación es un valor intuido a partir del conocimiento que se tiene de la actividad (que puede no ser correcto) y de la experiencia previa del equipo (que puede no ser la adecuada) complementado con todo tipo de distorsiones e interferencias: por ejemplo, un cliente que presiona puede forzar una estimación demasiado optimista.

Hay muchas técnicas para obtener la medida de la velocidad. La más habitual es la estimación de las historias de usuario durante el *Planning* y la recopilación de las historias de usuario efectivamente cerradas en la *Review*. Esa medida puede basarse en cualquier medio a condición de que el criterio seguido sea siempre el mismo por todo el equipo y a lo largo de todo el proyecto.

Al tratarse de una estimación más intuida que medible, no tiene sentido alguno establecer unas reglas muy estrictas sobre significado de los valores o la forma de obtenerlos. En este caso, la experiencia continuada ayudará a afinar progresivamente la estimación, aunque cualquier cambio en el entorno (el equipo, las herramientas, las técnicas, la forma de expresar los requisitos de cliente) tendrá impacto sobre el acierto de la estimación.

La velocidad estimada para un *Sprint* es la cantidad de *Story Points* o puntos de historia obtenidos de la valoración de complejidad que hace el equipo de las historias de usuario incluidas en una iteración.

Al terminar la *Review*, la suma de los puntos de las historias completamente cerradas de acuerdo con el *Product Owner* es el valor de la velocidad real alcanzada por el equipo en ese *Sprint*.

Si la valoración se hace con criterios similares a lo largo de los distintos *Sprints* y el entorno no varía dramáticamente, se debería ir viendo un valor estable de velocidad alcanzada por el equipo, valor que servirá para estimar en el *Planning* la cantidad de trabajo que se puede realizar en cada *Sprint*.

Aunque es posible jugar con estos números para obtener todo tipo de indicadores, su fiabilidad es muy baja. Como mucho se puede llegar a corregir el valor de velocidad con la cantidad de esfuerzo teórico (número de personas y jornadas laborales) en cada *Sprint*, de

forma que dos valores de velocidad similares no se consideren de la misma forma si en un *Sprint*, por ejemplo, falta la mitad del equipo.

El uso de la medida de velocidad debe ser interno y circunscrito al seguimiento del trabajo del equipo y como ayuda para su estimación. Tratar de usarlo como medida objetiva (por ejemplo, para comparar a un equipo con otro) solo dará lugar a que surjan malas prácticas, como podría ser sobrevalorar la complejidad de cada historia para así mostrar una velocidad más alta.

La evolución del trabajo durante el *Sprint* se visualiza con algunas herramientas, que se comentan en el siguiente apartado.

Herramientas

Scrum es una ayuda en nuestro trabajo, por lo que no debe suponer más esfuerzo, más complicaciones ni problemas. Si la gestión de artefactos y gráficas puede suponer una dificultad para la adopción de *Scrum*, conviene apoyarse en las herramientas disponibles. Muchas de estas herramientas son sofisticadas aplicaciones con versiones para dispositivos móviles, pero las hay muy simples igualmente útiles.

Una de las más conocidas, posiblemente la más sencilla y útil, es el uso de paneles con etiquetas adhesivas o *post-it*. Por término general, se usan como una representación del *Backlog*, especialmente del *Sprint Backlog*. En un panel, que puede ser móvil, o la propia pared, se dibujan unas áreas que representan los distintos estados por los que pueden pasar las historias de usuario y tareas (pendiente de iniciar, en curso, terminada, impedida) y unos *post-it* que representan cada entrada en el *Backlog*, que se colocan en el área que le corresponda.

Cuando se va a iniciar una tarea, se mueve del área “pendiente de empezar” al área “en curso”. Si el panel se coloca en un lugar bien visible, todas las personas involucradas en el proyecto podrán conocer de un vistazo su estado.

Usando símbolos y distintos colores para letras o etiquetas, se puede enriquecer la información, añadiendo datos sobre los componentes a que se refieren, quién trabajan en ello, a qué *release* corresponde, etc.

Otros diagramas, como el *burn-down chart*, también pueden construirse usando paneles, papel y rotuladores, aunque en este caso puede resultar más complicado reflejar los cambios. Por ello, existen herramientas informáticas que permiten reflejar el contenido y estado de un proyecto desarrollado con *Scrum*. Las más básicas son hojas de cálculo en las que cada celda refleja una tarea, historia, épica o tema, y que se disponen en columnas que representan estados, componentes, entregas o cualquier otra clasificación de la información relevante para el proyecto.

Hay herramientas más sofisticadas, que incluyen modelos que reflejan los elementos básicos de la metodología, permiten participar a todos los miembros del equipo *Scrum*

(equipo, más PO y SM) y que representan todos los distintos artefactos.

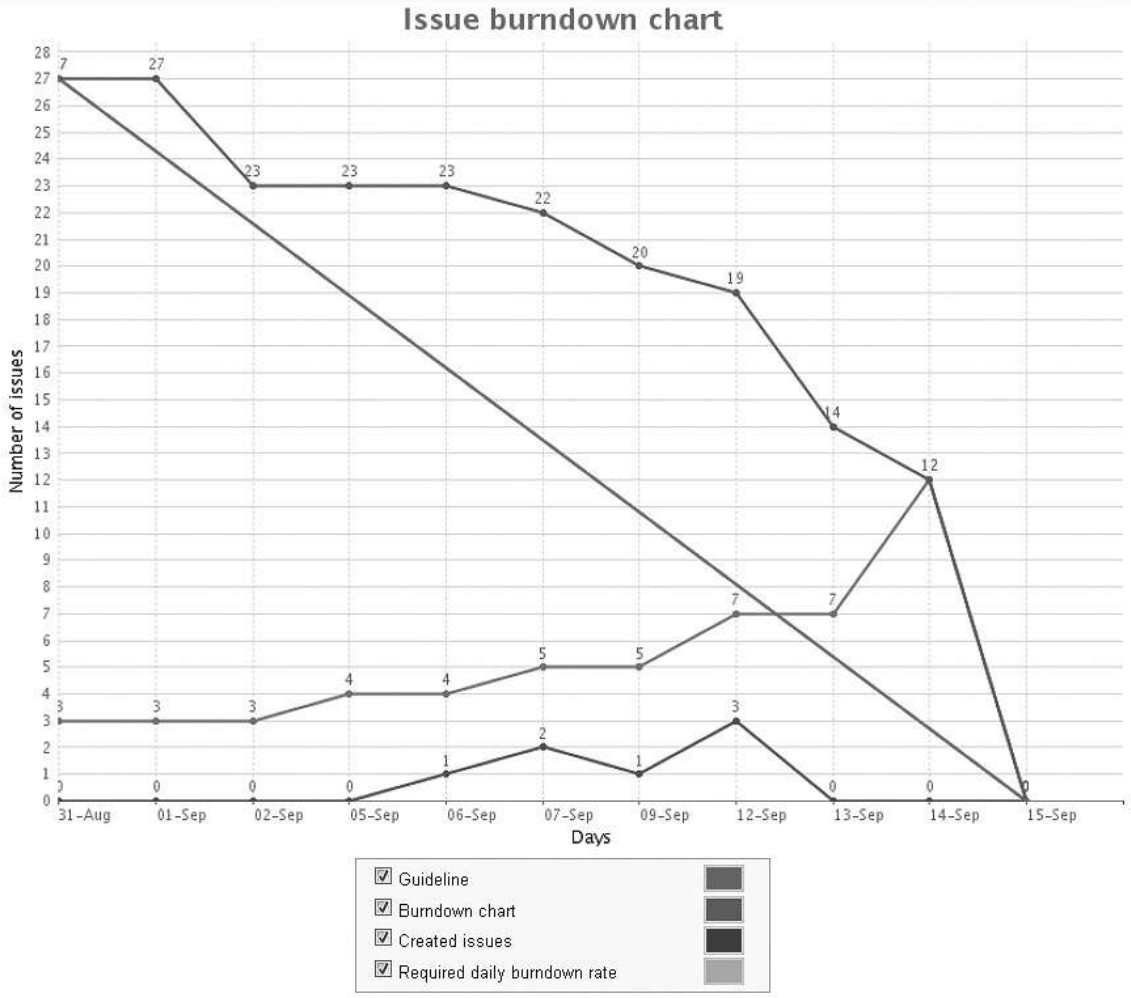


Figura 2.10. Herramienta informática para la generación del burn-down chart de Scrum.

Sin embargo, hay una herramienta extremadamente simple y efectiva, cuya eficacia ha sido constatada por años de uso de *Scrum*: mejorar la fluidez en la comunicación entre el equipo y todos los involucrados en el proyecto haciendo que compartan un mismo espacio físico.

El entorno de trabajo

En *Scrum* se da una gran importancia a la comunicación y a la difusión de información dentro del equipo de trabajo. Aunque hoy en día existen medios técnicos que facilitan el flujo de la información incluso con personas situadas a gran distancia, nada sustituye a la cercanía física. Por ello, una recomendación dentro de las metodologías ágiles es acercar al equipo de trabajo idealmente en un mismo recinto. Se trata de una forma de incentivar la comunicación, la colaboración, la difusión de información y el entendimiento común de los aspectos básicos

de trabajo.

La cercanía física no es solución mágica, pero sí es una ayuda para reducir riesgos. Por ejemplo, es más difícil que dos miembros del equipo elijan una misma tarea si la comunicación es suficientemente fluida. Las técnicas de desarrollo de software XP (*eXtreme Programming*) que se presentan más adelante en este libro hablan incluso del trabajo de dos personas en un mismo punto del proyecto a la vez, una junto a la otra.

Otra ventaja derivada de la proximidad física del equipo es poder usar herramientas que pongan a la vista de todos los datos básicos de la evolución del proyecto (por medio de paneles, por ejemplo). Esa cercanía facilita también el conocimiento de la evolución cuando se cierra una tarea determinada o se abre una nueva entrada en el *Backlog* de impedimentos.

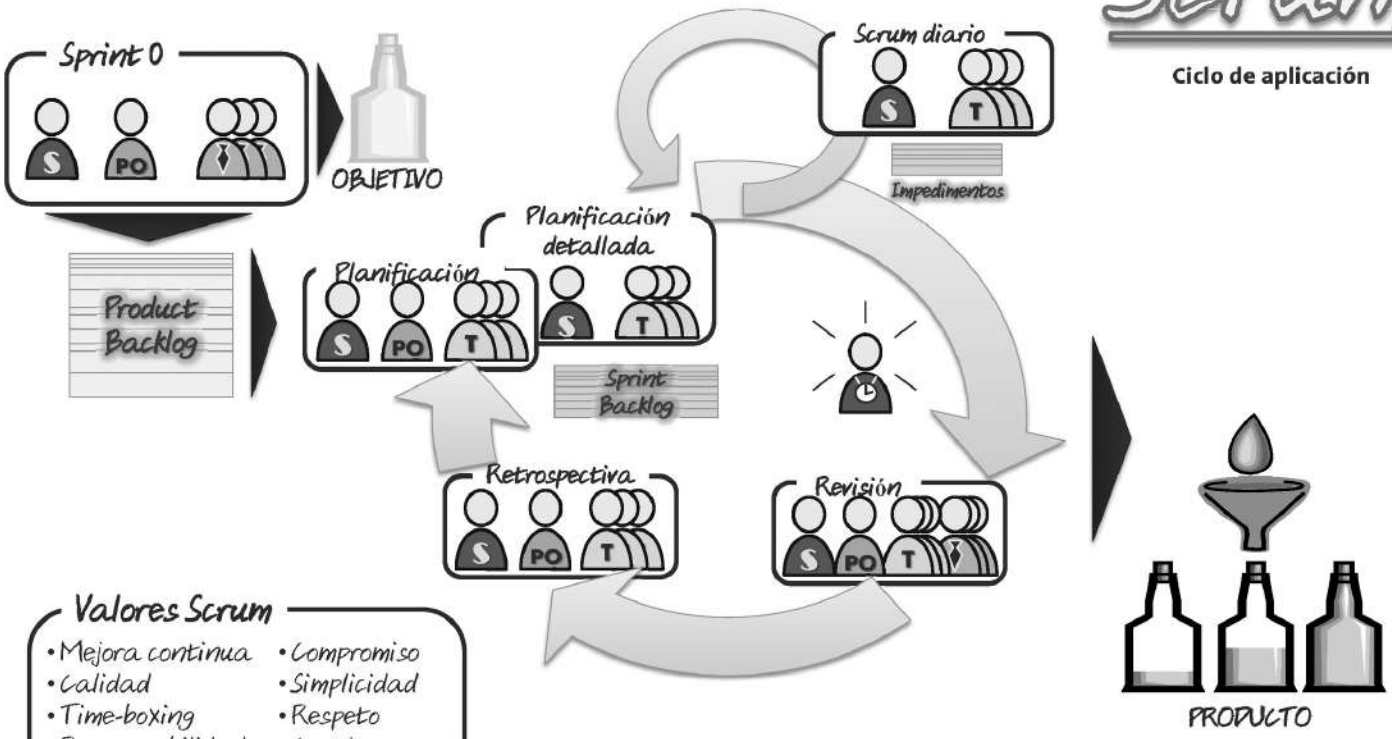
A continuación

Los siguientes capítulos se dedican a mostrar detalladamente todos los conceptos esbozados en este. Puede usar el contenido de este capítulo como una guía rápida o referencia del resto del contenido de esta primera parte, pero es muy importante que estudie detalladamente los demás capítulos. Allí es donde conocerá toda la riqueza de *Scrum* y descubrirá lo útil que puede resultar para desarrollar proyectos de manera productiva y con calidad.

Además, para hacer más fácil la comprensión de los conceptos presentados, se hará referencia a un proyecto real y concreto: la elaboración de este libro que tiene en sus manos.

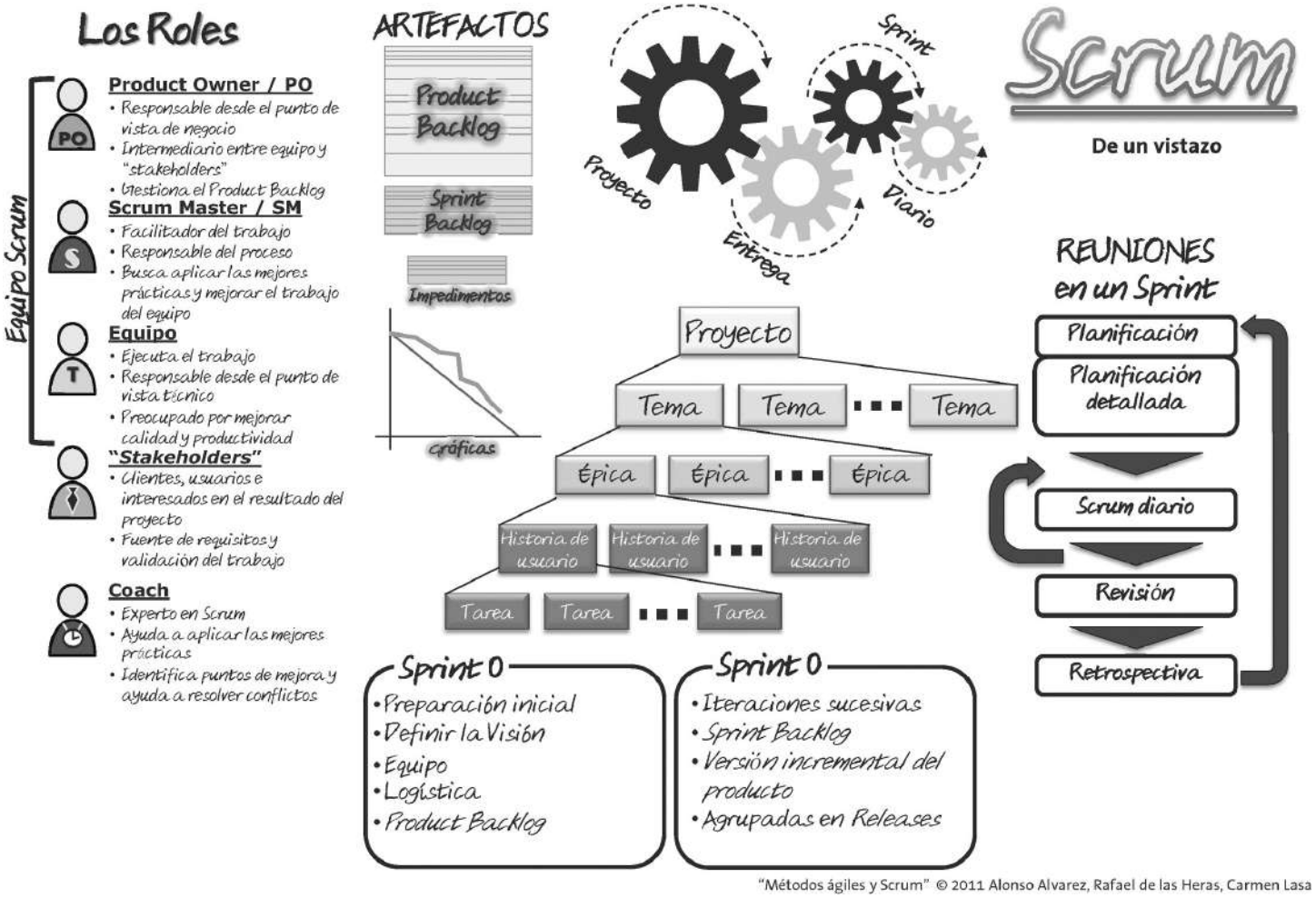
Scrum

Ciclo de aplicación



Valores Scrum

- Mejora continua
- Calidad
- Time-boxing
- Responsabilidad
- Multidisciplinar
- Flexibilidad
- Ritmo
- Compromiso
- Simplicidad
- Respeto
- Coraje
- Foco
- Predictibilidad
- Personas



---

16 Busque el libro y, en él, descargue los complementos que lo acompañan.

17 Centrarse más no quiere decir dejar de lado los otros factores, solo dar más relevancia a los primeros.

18 *Leader of the Band* se puede encontrar en http://www.scrumalliance.org/articles/36-leader-of-the-band.

3 Luces, cámara y Sprint 0

En este capítulo aprenderá:

- La definición del *Sprint* 0.
- El rol del *Product Owner*.
- Cómo crear la Visión.
- Cómo crear un plan de entregas.

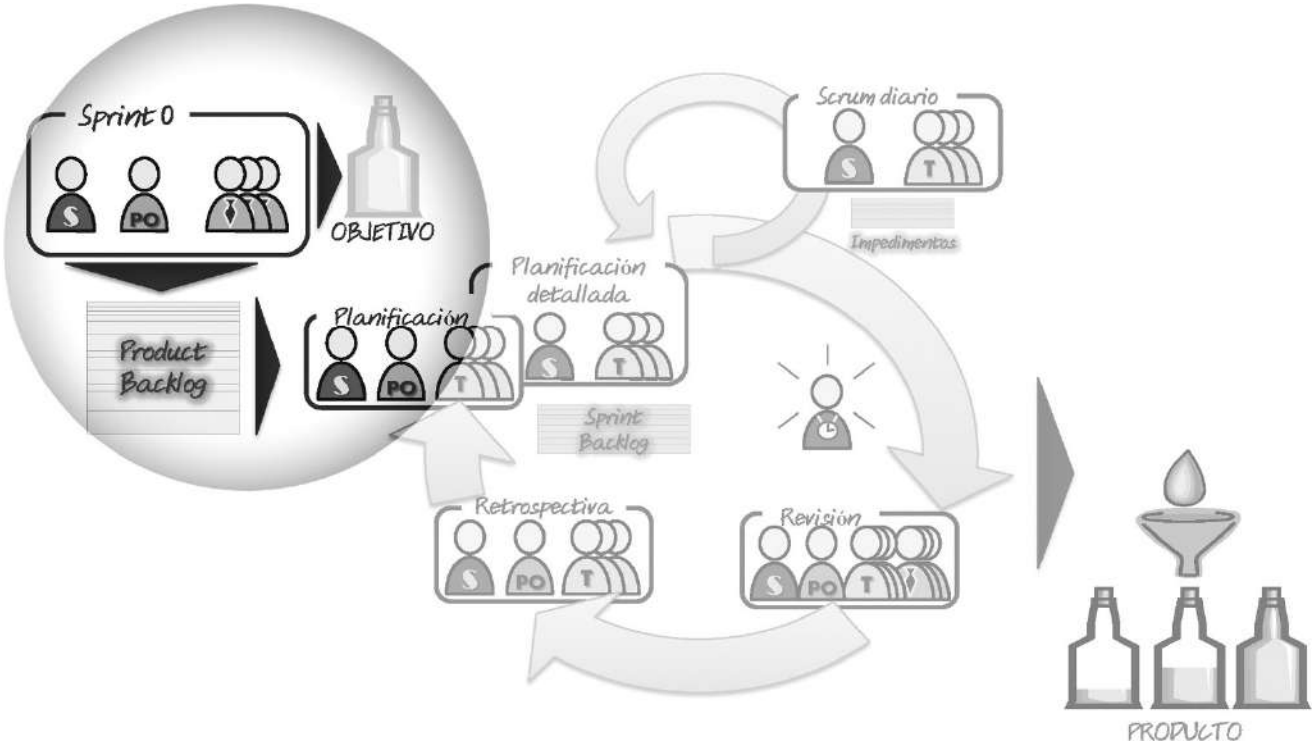


Figura 3.1. El Sprint 0 en el ciclo Scrum.

Tenemos un proyecto: escribir un libro. Ese proyecto tiene unos plazos aproximados, un esbozo de contenido, unos autores iniciales, un título provisional... Pero más allá de estos precarios elementos iniciales es un lienzo en blanco.

Hemos convencido a nuestro editor de que merece la pena aprovechar nuestra experiencia en el campo de los métodos ágiles y, en particular, *Scrum*, para publicar un texto divulgativo, en castellano. Pero no seríamos consecuentes con nuestro propio conocimiento y experiencia si no nos apoyáramos en estas técnicas para llevar a cabo nuestro proyecto.

Por ello, antes de empezar a escribir una sola palabra, vamos a dedicar un tiempo a diseñar este libro. Trazaremos las grandes líneas de su contenido, identificaremos los trabajos que vamos a realizar, organizaremos esta actividad. Todo esto lo haremos en función de su grado de concreción y de la prioridad que tiene para el resto del proyecto. En pocas palabras, vamos a empezar este libro por el *Sprint* 0.

Cuando es el momento de empezar a crear algo nuevo, en cualquier ámbito, existen una serie de requisitos y una preparación mínima. *Scrum* no es una excepción y necesita de una gestación que facilite el resto de las actividades que se realizarán posteriormente.

En muchos ámbitos de *Scrum*, a este proceso de gestación o preparación inicial, que recoge todas las actividades necesarias para iniciar las iteraciones de trabajo, se le llama *Sprint* 0, *Sprint Zero* o *Inception Sprint*.

Nota:

Muchos puristas de Scrum prefieren no asociar esta fase con la palabra Sprint. La razón es que al final de un Sprint siempre se espera un incremento de valor para el cliente, que en esta fase no se suele producir. Así, Ken Schawber, co-creador de Scrum, define el término Sprint 0 como un término mal usado para describir la planificación que tiene lugar previa al primer Sprint.

Más allá de las discusiones acerca de la idoneidad del término, lo que está completamente asimilado y acordado es que tiene que existir una fase inicial. En esta fase, se prepara toda la logística, mecánica y metodología a seguir durante todo el desarrollo del proceso de creación o proyecto. ¿Quién será el desencadenante de esta gestación? El *Product Owner*.

El Product Owner. El visionario

En los productos y proyectos en los que *Agile* no está presente, hay un cliente que tiene una necesidad y define unos requisitos para cubrir esta necesidad. Estos requisitos son tomados por un equipo que quiere convertirlos en realidad. Los interpreta según su conocimiento y crea lo que entiende que el cliente está esperando. El resultado suele distar de lo que el cliente había querido explicar inicialmente en sus requisitos.

En *Scrum*, una de las medidas para mejorar este proceso de entrega a un cliente y minimizar la diferencia con lo finalmente entregado al cliente es la creación de un rol específico. La tarea de este rol es ayudar a converger la Visión del cliente, con la Visión del equipo de trabajo. Este rol es el *Product Owner*, PO o dueño del producto.

¿Qué es un PO?

En *Scrum*, el PO es la voz del cliente, es el visionario que aúna las necesidades de todos los clientes y personas a las que les puede afectar o resultar relevante el producto o proyecto en desarrollo. A este conjunto de personas se les conoce colectivamente como *stakeholders*. El PO es el responsable de inspeccionar y adaptar estas necesidades, al esquema de trabajo definido en *Scrum*. Esto quiere decir que constantemente estará creando nuevos requisitos y los priorizará para que el equipo de trabajo pueda manejarlos y llevarlos a cabo. Estos requisitos se pueden crear continuamente porque el PO mantiene una Visión actualizada del producto o proyecto. Esta visión, una vez convertida en el objetivo perseguido, representa todas las características que aportan valor al usuario o cliente final.

Podría parecer que la responsabilidad del *Product Owner* simplemente se queda en

trasladar los requisitos de los clientes al equipo de trabajo. De hecho, esto ocurre en muchos casos, restando potencial a esta forma de trabajar. El papel del PO acarrea muchas más tareas o responsabilidades que suelen pasar desapercibidas y, a menudo, son ignoradas. El PO es el estratega del producto, se encarga de definir una estrategia a largo y corto plazo para garantizar el éxito del producto, ya que es el responsable final del éxito del proyecto y de su ROI (Retorno de inversión). Por esta razón, el *Product Owner* debe tener a su alcance todos los medios y poder de decisión para poder materializar ese éxito.

Además, cuando se realiza un proyecto, junto a los intereses del cliente final, se generan muchas expectativas por parte de otras personas (gestores, proveedores, etc.) que deben ser gestionadas correctamente. Es responsabilidad del *Product Owner* representar al producto frente a todas estas personas o entidades, incorporando toda la información derivada de ellas como requisitos del producto. Esta responsabilidad requiere unas capacidades especiales de comunicación y negociación para poder conseguir siempre lo mejor para el producto o proyecto en desarrollo.

También es interesante remarcar que el *Product Onwer*, como estratega del proyecto, debe asumir ciertas tareas de gestión sobre él. Estas tareas implican definir y actualizar el plan de entregas del proyecto, como se verá en una posterior sección de este capítulo, o controlar que el presupuesto para la ejecución del producto o proyecto sea el correcto.

El *Product Owner* tiene que guiar al equipo en la dirección correcta para que llegue a donde el cliente quiere llegar. Desde este punto de vista, el *Product Owner* es un líder, pero siempre sin perder de vista que es también un miembro de un equipo que tiene un interés común. Como es fácil de imaginar, tener intermediarios siempre es un problema, así que lo ideal sería siempre que el cliente fuera el *Product Owner*. En muchas organizaciones, no es posible que el cliente sea el propio PO, por lo que se crea la figura del *Proxy Product Owner,* como representante de los clientes aunando sus peticiones.

Nota:

Cuanta más gente exista entre el equipo y el cliente, más distorsión sufrirán los requisitos y menos se parecerá el resultado final a lo que el cliente está esperando.

Nadie más que el *Product Owner* puede interferir en el guiado del producto. El equipo solo debe “obedecer” las directrices definidas por el PO. Estas directrices no son órdenes, son las prioridades y requisitos que están definidos en el *Product Backlog* o pila de producto, al que se dedicará el siguiente capítulo. El *Product Owner* no guía el producto o al equipo diciendo cómo hacer las cosas. Especifica qué se tiene que hacer y en qué orden para que el equipo se autogestione y encuentre la mejor manera de llevarlo a cabo. El *Product Onwer* reta al equipo y este se compromete y responde al desafío.

Precisamente, compromiso es una de las palabras claves en *Scrum* y ayuda a que se diferencien los roles. Para visualizar los distintos roles, se suele usar una historia sobre un

cerdo y una gallina que dice así: Estaban un cerdo y una gallina hablando tranquilamente sobre el futuro y la gallina le dice al cerdo: “¿Por qué no abrimos juntos un restaurante?”. El cerdo le responde: “Muy buena idea, ¿cómo llamaremos al restaurante?”. Se hace un silencio y finalmente responde la gallina: “¿Y si lo llamamos ‘Huevos con jamón’?”. “No, gracias, esto tiene truco.”, respondió el cerdo. “Yo estaría totalmente comprometido, mientras que tu sólo estarías involucrada”, explicó el cerdo.

El *Product Owner* es un papel comprometido al 100% con el equipo y con el producto. “Es un cerdo”. Cualquier otro implicado en el producto o *stakeholder* (usuarios, marketing, ventas...) son como las gallinas, podrán aportar mucho al producto o proyecto pero no llegarán a estar comprometidos.

Nota:

Algunas veces se habla de pasar el test del pato al Product Owner. Este test dice que, si andas como un pato, hablas como un pato y tienes plumas como un pato... eres un pato. Este test viene a decir que, si no participas en las reuniones como Product Owner, el equipo no te hace partícipe de las decisiones. Por esta razón, no tendrás potestad para modificar el Backlog, ya que puede que no estés lo suficientemente comprometido. Eres una gallina que no está desempeñando correctamente su papel.

Todas estas responsabilidades que, como se ha visto, un PO debe asumir, se traducen en las siguientes tareas operativas dentro de *Scrum*:

- Definición de la Visión del producto.
- Organización de dinámicas de obtención de requisitos.
- Creación y mantenimiento del *Backlog*.
- Resolución de dudas del equipo en cualquier momento.
- Preparación de las reuniones de estimación y planificación.
- Asistencia a las reuniones de *Scrum*.
- Aceptación o rechazo del trabajo realizado durante un *Sprint* por parte del equipo.

¿Qué no debe ser un PO?

El papel del *Product Owner* es un papel peligroso: cualquier problema en el desempeño de este rol puede afectar a la totalidad del proyecto. La razón es que cualquier problema en este rol se propagará, a través del *Backlog*, al equipo y al producto.

Roman Pichler hace en su libro *Agile Product Management with Scrum*[19] una enumeración muy interesante de tipos de *Product Owner* conflictivos que pueden llegar a darse y los problemas relacionados. Son los siguientes:

- **El PO sin poder:** Un *Product Owner* en esta situación deberá escalar a un nivel jerárquico superior cada decisión que tome. Esto implicará retrasos y bloqueos que se acumularán a lo largo de todo el proyecto.
- **El PO saturado de trabajo:** Un *Product Owner* saturado de trabajo no podrá dedicar tiempo a tareas tan vitales como el mantenimiento del *Backlog* y a las reuniones de trabajo para ello. Esto hará que el equipo no cuente con un repositorio actualizado con la funcionalidad que se quiere implementar, creando bloqueos de trabajo y reuniones de estimación infinitas.
- **El PO parcial:** El PO debe ser un rol desempeñado por una única persona. En muchas organizaciones este rol se divide. Un ejemplo de esta división es cuando la parte de negocio del rol lo desempeña una persona y la parte tecnológica otra persona. La consecuencia es que, a la hora de definir las prioridades, nunca se llegará a un consenso. Esto producirá tensiones innecesarias y discusiones en las reuniones de estimación y planificación.
- **El PO a distancia:** Un *Product Owner* separado del equipo y que no pueda asistir físicamente a las reuniones dificultará mucho la cohesión del equipo *Scrum*. Se complicará también el proceso de comunicación y resolución de dudas que debería fluir de forma natural en el equipo.
- **El Proxy PO:** Cuando el rol de *Product Owner* no lo puede desempeñar la persona que idealmente lo debería hacer, por los problemas citados con anterioridad, se suele asignar un representante o *Proxy*. Si a este *Proxy* no se le otorga la potestad suficiente para tomar decisiones, se tendrá un ejemplo de PO sin poder, con los mismos problemas citados con anterioridad.
- **El equipo de PO:** Es una situación parecida al PO parcial, pero aquí se tiene un conjunto de *Product Owners* que tienen todas las competencias. Si no hay un elemento único que los organice o haga de portavoz, se tendrán problemas de definición de requisitos y en su priorización debido a los distintos puntos de vista.

El *Product Owner* no es un héroe solitario, no tiene por qué resolver todos sus problemas por su cuenta. El *Product Owner* puede (y debería) contar con un equipo de producto al que recurrir siempre que sea necesario. En este equipo puede haber desde analistas de negocio o expertos en marketing hasta diseñadores de servicios.

Un visionario visionando la Visión

Como se ha podido ver, el *Product Owner* tiene muchas responsabilidades y obligaciones, pero por algún punto se tiene que empezar a trabajar. La chispa que enciende todo es la Visión. Sin la Visión no se puede hacer nada. No se pueden conseguir fondos o recursos, ya

que no se sabe qué se quiere hacer. Es lo primero que tiene que hacer el *Product Owner*: definir la Visión del producto del que es dueño.

La idea es poder crear una Visión fuerte que sirva de apoyo para todo el proceso de creación del proyecto o producto. Pero ¿qué es la Visión?

De entre todas las definiciones existentes de la Visión, en este texto nos decantamos por la descrita por Robert Kaplan y David P. Norton en su libro *Strategy Maps*[20]. En este libro, se habla del concepto de la Visión desde el punto de vista de una empresa, pero fácilmente puede ser extrapolable a proyectos o productos.

La Visión es descrita como un resumen de las metas a medio y largo plazo a las que se quiere llegar. Es una imagen mental de dónde se quiere estar o qué se quiere tener en un determinado plazo de tiempo. Es una información de alto nivel y de propósito general. No tiene que ser detallada, pero sí tiene que ser sencilla y clara. Sin ambigüedades. La sencillez es la clave para definir el objetivo que se quiere alcanzar. Siguiendo el principio de simplicidad, que enunciaba Guillermo de Ockham en el siglo XIV, si se tienen varias alternativas posibles y equivalentes, la solución más sencilla siempre debe ser la elegida, ya que será la correcta. La simplicidad no tiene que ver con el maquillaje del producto o del proyecto, apunta directamente a la esencia de lo que se quiere construir. Por esta razón, debe ser parte inherente de su Visión.

Figura 3.2. La sencillez de la navaja de Ockham.

La información de la Visión se puede manejar de manera interna, con todos los implicados en el proceso de creación del producto o del proyecto, así como de forma externa para transmitir cómo se quiere que se perciba lo que se va a crear. La Visión debe contener la esencia de lo que se está creando y sus claves. Es la base o cimiento sobre los que se construirá todo lo demás, por lo que debe ser clara y fácilmente comunicable a todo el mundo. Es muy importante que la Visión sea un instrumento de comunicación, ya que será lo que ayude al *Product Owner* a definir la dirección en la que todos los miembros del equipo

deberán empujar. El equipo remará rumbo a la Visión, manteniendo la motivación, si la entiende y la comparte. Debe convertirse en un activo compartido del equipo. La Visión no es solo la base para crear algo nuevo, sino que también será un instrumento vital para conseguir presupuesto para su realización. También proporcionará un material de apoyo en las fases de lanzamiento, venta y promoción del producto.

En resumen, la Visión es el origen y el final de nuestro proceso, ya que es lo primero que se tiene que hacer para poder empezar a trabajar, pero también representa al punto al que se quiere llegar y dónde terminará el proceso.

Es relevante matizar que la composición de la Visión dependerá mucho de lo que se quiere crear. En el caso particular en el que se esté creando un producto, definir la Visión implica resolver una serie de preguntas, que van a llevar a la creación de los dos elementos básicos de una Visión de un producto: la propuesta de valor y el modelo de negocio (véase también el capítulo 16 donde se habla de los *canvas* o lienzos de negocio, o el 15 donde se expone el método *Lean startup*, creado precisamente para diseñar modelos de negocio, entre otras cosas).

Definir la propuesta de valor implica responder una serie de preguntas relacionadas con la manera en la que los usuarios finales del producto se beneficiarán del mismo. Ejemplos de estas preguntas son:

- ¿Qué problema se intenta resolver?
- ¿Cómo es la solución general al problema que se plantea?
- ¿Quiénes van a ser los usuarios o clientes?
- ¿Cuál es la situación en la que están esos usuarios?
- En la situación en la que los usuarios están, ¿qué necesidad suya se quiere cubrir?
- ¿Quiénes son los competidores y cómo se compara el nuevo producto, con los productos de la competencia?

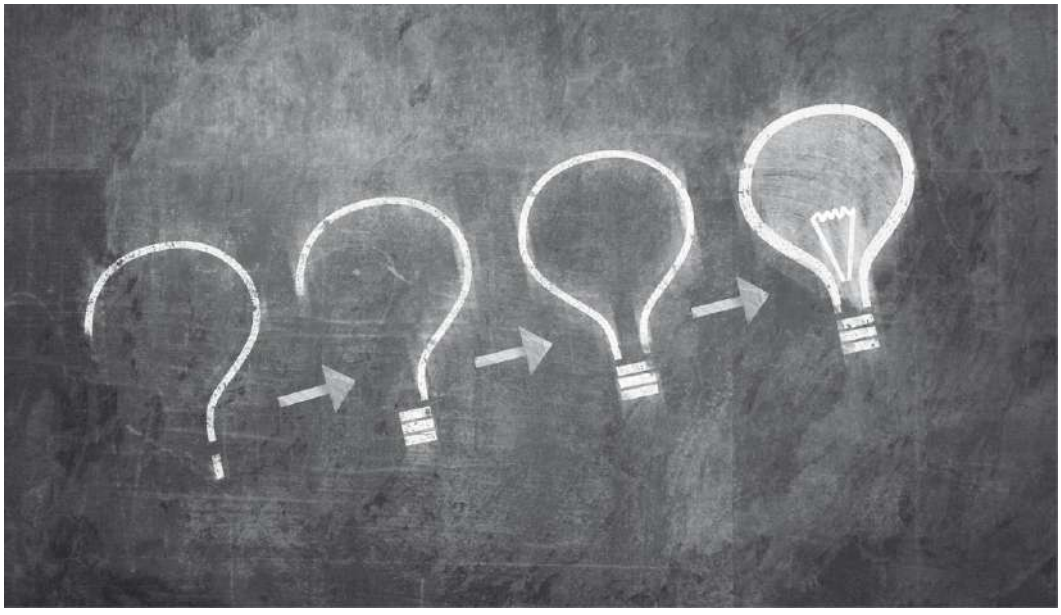


Figura 3.3. Definir al modelo de negocio implica responder a una serie de preguntas clave.

Nota:

Hay que darse cuenta de que la propuesta de valor resalta el problema al que se dirige el producto, más que a los detalles solución del problema. Los detalles de la solución se empezarán a conocer cuando se inicie la creación del Product Backlog, como se verá en el siguiente capítulo.

Por otro lado, definir el modelo de negocio implica responder una serie de preguntas acerca de cómo los creadores del producto se beneficiarán de él. Ejemplos de estas preguntas son:

- ¿Cómo se va a conseguir el presupuesto para realizar el negocio?
- ¿Cuál va a ser el presupuesto y el tiempo para crear y lanzar el producto?
- ¿Cómo se va a generar beneficio?
- ¿Cómo van a encontrar y conseguir el producto los usuarios?
- ¿Cuáles son los costes del producto?

Tener ambos elementos como componentes de la Visión es muy importante. Por lo tanto, no definir correctamente la propuesta de valor o el modelo de negocio puede elevar la probabilidad de fracaso de forma considerable.

Nota:

Un ejemplo muy claro de problemas en la definición de la Visión se pudo observar en la burbuja de las empresas .com, en las que la Visión estaba creada con una propuesta de valor definida, pero sin existir un modelo de negocio detrás de este.

Al introducir la Visión, esta se definía como un resumen de la imagen mental que se tiene. Tiene que ser clara y concisa. Para garantizar estas propiedades, es útil someter a la Visión al test del *elevator pitch, moore elevator* o discurso del ascensor. Este test dice que la Visión debería poder expresarse en dos frases que pudieran ser comunicadas durante el tiempo que se tarda en subir varios pisos en un ascensor. Este test también indica una plantilla que debería poder aplicarse a la Visión como se explica en el libro de Geoffrey Moore, *Crossing the Chasm*[21]. Esta plantilla tendría el siguiente formato:

- Para: ... .
- Que actualmente están desconformes con: ... .
- Nuestro producto es: ... .
- Qué ofrece: ... .

- A diferencia de: ... .
- Nuestro producto consiste en: ... .

Si habiendo definido la propuesta de valor y el modelo de negocio, se puede sintetizar la Visión en esta plantilla, se puede decir que se cuenta con una buena Visión, lista para generar las características principales del producto.

Ya se ha definido la primera versión de la Visión y, además, se ha cumplido la guía del discurso del ascensor. Con esto, se han extraído las temáticas principales que va a cubrir el producto. ¿Se tiene ya una poderosa Visión y se está listo para crear el *Product Backlog*? Todavía no, ya que se ha creado la Visión, pero se puede decir que es una Visión en blanco y negro. Ahora toca colorearla, mejorarla y ensalzarla. Para esto, se van a analizar una serie de técnicas que van a permitir tener la Visión lista para dar el siguiente paso.

El modelo de Kano

En la década de los años ochenta, el profesor Noriaki Kano elaboró una teoría sobre la creación de productos y la satisfacción de clientes. En este modelo se diferenciaban 3 tipos de características que un producto puede tener, sobre las cuales se puede analizar la satisfacción del cliente. Estas funcionalidades se clasifican en:

- Las características básicas u obligatorias: Son aquellas características que cuando el producto no las contiene producen un alto nivel de insatisfacción en el cliente, pero que, cuando sí están contempladas, producen al cliente un sentimiento neutro, ya que era lo que esperaba. Un ejemplo de este tipo de características podría ser el papel higiénico en el cuarto de baño de un hotel. No suele ser muy satisfactorio para los clientes no encontrar papel cuando lo necesitan.
- Las características de rendimiento o lineales: Son aquellas que, cuanto más se tienen, más satisfacción producen al cliente y, cuantas menos se tienen, menos grado de satisfacción producen. Si se sigue con el ejemplo del hotel, en este caso, se podría hablar de los canales de televisión que se pueden ver en la televisión del hotel. Cuantos más, mejor.
- Por último, están las características inesperadas o emocionantes: Son aquellas que producen un sentimiento de innovación o novedad. Son los atributos de un producto que, si no están, al no esperarse, dejan al cliente con un sentimiento neutro, pero cuando están producen un alto grado de satisfacción. Un ejemplo para este tipo de característica son los bombones que algunos hoteles dejan en la habitación por las tardes.

Nota:

Las características no son de un tipo para siempre. Lo normal es que las características inesperadas o emocionantes de hoy se conviertan en las básicas de mañana. Hace tiempo, el mando a distancia de la televisión era algo novedoso, pero hoy día se considera como una característica básica de las televisiones.

La siguiente imagen indica cómo se comporta el grado de satisfacción del cliente según el cumplimiento de las necesidades, dependiendo del tipo de funcionalidad.

Como se puede observar, existe un área de indiferencia donde la presencia de una funcionalidad en esa cuantía no afecta en ningún modo al usuario; a esta área se le denomina zona de indiferencia.

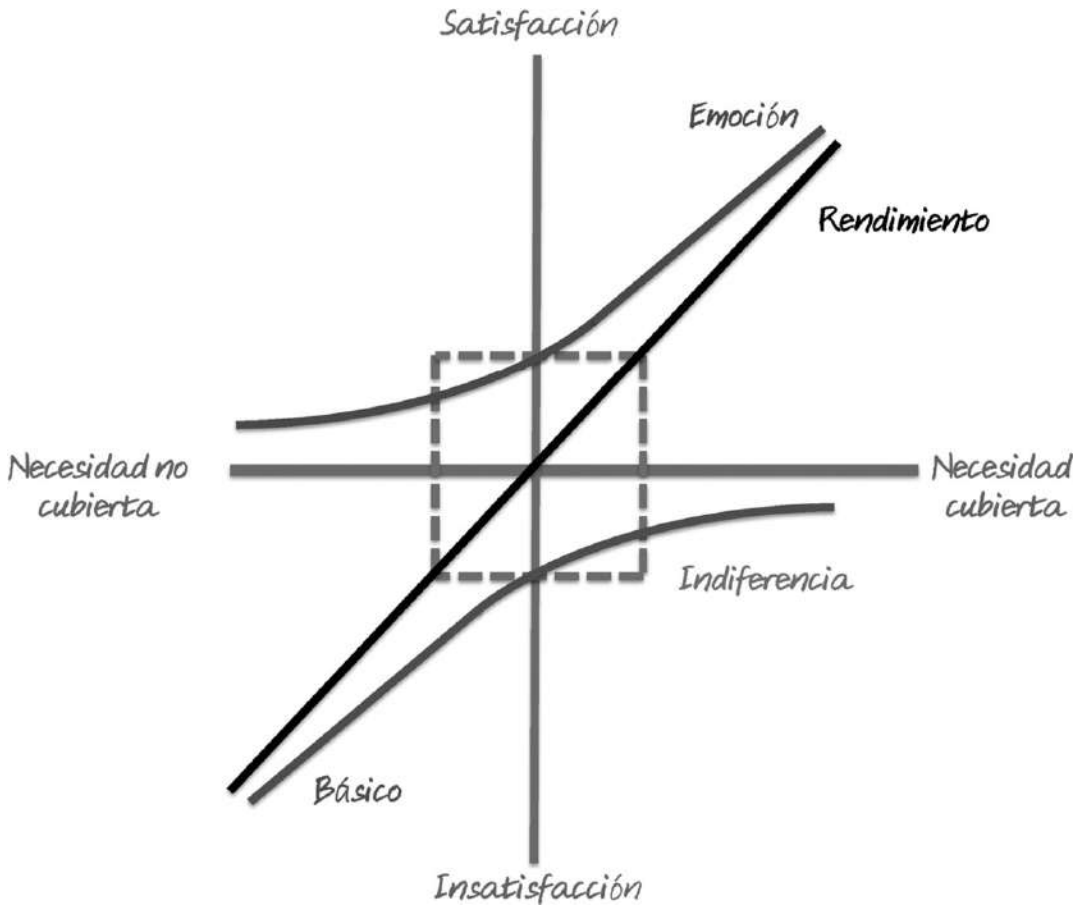


Figura 3.4. Modelo de Kano.

El reto a la hora de construir la Visión es detectar el tipo de características que se quieren incluir en ella, para maximizar los beneficios que el cliente obtendrá con esa combinación de características básicas, lineales e inesperadas.

Personas y escenarios

El concepto “Persona” representa a personajes o usuarios determinados, aunque ficticios, que permiten entender de manera clara quiénes serán las personas que utilicen el producto. El concepto “Escenario” representa a las situaciones posibles en las que las “Personas” usarán el producto. Esta técnica ayuda a enriquecer la Visión, aportando información concreta de las situaciones en las que los usuarios emplearán el producto. Esto permitirá extraer los atributos más críticos del producto, según los escenarios que se definan.

Para crear las “Personas” y los “Escenarios”, se cuenta como base con la información obtenida en las preguntas de la definición del concepto. Particularmente, en la información recopilada para quién va a usar el producto y en qué situación se va a usar el producto.

Prototipos y mockups

Cuando se está creando algo nuevo, se intenta identificar las necesidades de los clientes y usuarios potenciales escuchándolas. Este proceso se conoce como VoC (*Voice of Customer*) o VdC (Voz del cliente). En muchos casos, el cliente no sabe cuáles son sus necesidades y se utiliza un proceso denominado MoC (*Mind of Customer*) o MdC (Mente del Cliente), que intenta interpretar la necesidad del cliente, aunque este no sepa expresarla. En este caso, la creación de la Visión es parte de un proceso de descubrimiento, de adquisición de conocimiento, mediante la interacción con el cliente. Es un proceso de inspección, *feedback* y adaptación, que necesita de artefactos que exploren y presenten las posibilidades al usuario de una forma tangible, aunque no sea real. A estos artefactos se les denomina prototipos o *mockups*. Cuando estos prototipos entran en detalle en la experimentación e intentan profundizar en la tecnología que se va a usar para implementar el producto, se denominan *spikes*.

Se invita a los clientes a probar estos prototipos para obtener sus comentarios en una etapa temprana y poder incorporar esta información a la Visión del producto. El resultado es un producto más cercano y satisfactorio para el cliente final.

El storyboard, user journey o experiencia de usuario

El *storyboard*, *user journey* o experiencia de usuario es un método para conceptualizar y estructurar el contenido y la funcionalidad, definidos en la Visión. Es una manera de reflejar los pensamientos, consideraciones y experiencias del usuario. Es una forma de mezclar las “Personas” y “Escenarios” que se han definido con anterioridad, mediante los flujos de usuario, aplicando patrones y procesos de comportamiento. Es la forma de unir toda la información que se ha trabajado en la Visión, en un único modelo. Refleja cómo un usuario, que tiene unas determinadas necesidades, al estar en un escenario concreto va a seguir un patrón de actuación para satisfacer su necesidad.

Este modelo de personas y escenarios resolviendo sus necesidades permite identificar el principio de Pareto sobre nuestra Visión. Pareto identificó que la gente en su sociedad se dividía naturalmente entre los “pocos de mucho” y los “muchos de poco”. Se establecían así dos grupos de proporciones 80-20 tales que el grupo minoritario, formado por un 20% de población, ostentaba el 80% de algo y el grupo mayoritario, formado por un 80% de población, tenían el 20% de ese mismo algo. Siguiendo este principio, se podrían identificar qué funcionalidades o características se usarán por la mayoría de la población y cuáles por la minoría, pudiendo poner el foco de la Visión en esta relación.

Nota:
Un ejemplo de esta aplicación de la regla se enuncia como el 80% de los programas que tienes en el ordenador los utilizas el 20% de las veces, mientras que el 20% restante de los programas los utilizas el 80% de las veces.

Una vez se obtenga esta experiencia de usuario, como parte de definición de la Visión, será sencillo el proceso de crear la versión inicial del *Backlog*, como se verá en el siguiente capítulo.

Nota:

La Visión no es algo estático que se crea en el Sprint 0 y ya no se modifica. Como parte del proceso Scrum, es algo que se retroalimentará al final de cada Sprint y se irá enriqueciendo a medida que el proyecto avance. Esta modificación de la Visión afectará a todos sus elementos y estos se tendrán que ir actualizando.

Preparándose para el viaje

Ya se tiene al *Product Owner* y la Visión. Ya se tiene claro un primer esbozo de qué se tiene que hacer, así que es el momento de organizar el cómo se va hacer. Para esto, durante el *Sprint* 0 se tienen que preparar una serie de cosas que evitarán perder tiempo productivo una vez se hayan iniciado las iteraciones de trabajo. Estas cosas que hay que organizar son: el equipo y la logística, el *Product Backlog*, las reglas de juego y la gestión de la incertidumbre.

¿Quiénes somos? El equipo y la logística

Lo primero que se tiene que organizar es un equipo *Scrum* para la ejecución de la idea. Los equipos *Scrum* están formados por el *Scrum Master* y un equipo multidisciplinar. Con multidisciplinar se hace referencia a que es un equipo en el que se aglutinan todos los perfiles

necesarios para la realización de la Visión. En *Scrum* no se cuenta con varios equipos especializados que van realizando sus tareas de forma secuencial. La idea es que todas las operaciones se hagan por un mismo equipo. Por ejemplo, en un proyecto de desarrollo software, en vez de tener un equipo de diseño gráfico, otro de desarrollo y otro de QA (calidad) independientes que se pasan trabajo entre ellos, se tiene un único equipo con distintos perfiles.

Figura 3.5. Imagen de equipo multidisciplinar.

Nota:

Los equipos se podrían ir creando progresivamente según va avanzando la ejecución del producto o del proyecto, pero se recomienda formarlo desde el principio. Creando el equipo desde el inicio se integrará rápidamente y se le sacará más partido a alguna de las técnicas de Scrum, como la medida de la velocidad.

Aunque parecería lógico que el *Scrum Master* fuera el primer papel que se va a cubrir, en

realidad no es así. Cuando se está creando un equipo en una empresa con cierta cultura *agile*, lo ideal es crear el equipo y que sea este quien elija a su *Scrum Master*. Como se verá en capítulos posteriores, el SM es el representante del equipo y está para servirlo. El liderazgo que sustenta está basado en el ejemplo y en la confianza que el equipo mantiene en él. Elegirlo antes de la creación del equipo y que este participe en la formación del equipo crea un sentimiento de jerarquía contrario a las bases de autogestión y de "otorgamiento de poderes" o *empowerment* al equipo.

Nota:

Cuando no hay una madurez en metodologías ágiles, el Scrum Master se suele elegir antes que al equipo para facilitar el proceso de inicio del proyecto.

Ya que idealmente no se debería empezar por el *Scrum Master*, ¿por dónde se debería comenzar? Dado que en la Visión se han tratado los temas sobre los que va a girar el producto o proyecto, se puede conocer la temática o dominio sobre el que se quiere trabajar. Lo ideal sería elegir un líder tecnológico o experto en la materia. El *Scrum Master* debe conocer el dominio en el que se trabajará, pero no debe ser el líder tecnológico o experto. Su prioridad como SM le restaría tiempo para esta otra labor. Una vez se cuente con el experto, los otros perfiles del equipo podrán definirse de manera específica gracias al conocimiento que este aporte. Además del conocimiento específico necesario para formar un buen equipo *agile*, sus miembros deben tener una serie de cualidades, que harán que la capacidad conjunta del equipo se maximice:

- **Trabajo en equipo:** *Scrum* trata de trabajar en equipo. Establece que el trabajo de dos personas en equipo es más que la suma de sus dos trabajos por separado. Los miembros que no sepan trabajar en equipo quedarán aislados rápidamente.
- **Generosidad:** No se trabaja por objetivos individuales. Los objetivos son los del equipo. Por esta razón, es casi más importante ayudar a un compañero, si tiene problemas, antes que terminar una tarea propia. No falla un miembro del equipo, falla el equipo.
- **Comunicación:** Un equipo funcionando es un equipo sincronizado. La base de la sincronización es la comunicación. Miembros aislados y trabajando por su cuenta no aportarán valor al equipo.
- **Capacidad de aprendizaje:** *Scrum* se basa en el principio de revisar y adaptar. Para hacer esto, es necesario aprendizaje constante y adaptabilidad. Sin esta capacidad, no se podrá progresar en el equipo, convirtiéndose en un lastre para este.

Una vez se forma el equipo, es necesario encontrar un sitio para que trabaje. Lo ideal sería que se compartiera un mismo espacio físico y que estén colocados cerca. Esta es la mejor

manera de que se fomente la comunicación que se está buscando para sincronizar el equipo. Una sala para tener al equipo junto, donde puedan tener en las paredes todo el material relacionado con el producto o proyecto y donde no molesten al resto de la empresa en sus reuniones o discusiones, sería perfecto.

Nota:

Muchos equipos encuentran la solución para que la separación física no se convierta en una barrera para su comunicación, usando salas de videopresencia, videoconferencia o sistemas de mensajería instantánea.

Finalmente, es necesario dotar al equipo con todas las herramientas y materiales que vayan a necesitar en la ejecución del proyecto o la creación del producto, desde ordenadores hasta martillos, es decir, todo aquello que se prevea que van a necesitar.

¿Qué hay que hacer? El Product Backlog

Una vez se tiene el equipo seleccionado, un sitio para trabajar y todo el equipamiento necesario, es un buen momento para conocer en detalle qué es lo que se tiene que hacer. Es el momento de crear el *Product Backlog* o pila de producto como materialización de la Visión y el objetivo que se quiere alcanzar. El *Product Backlog* es la colección de funcionalidades o características que nuestro producto debe cumplir para alcanzar el objetivo deseado. Dada la importancia de este repositorio de funcionalidad, se ha decidido dedicar el próximo capítulo a explicar en detalle su creación y gestión.

¿Cómo lo hacemos? Las reglas del juego

Para poder tener un equipo sincronizado, es importante acordar cuál va a ser la forma de trabajar. Esta viene definida por dos dimensiones: por un lado, se debe identificar cómo trabajará el equipo desde el punto de vista de procedimientos; y, por otro lado, se debe acordar cómo trabajará el equipo a nivel ejecutivo.

Desde el punto de vista de los procedimientos, durante el *Sprint* 0, se puede definir la mecánica de trabajo que se va a seguir y las herramientas que se van a usar. Así pues, en este periodo, se suelen decidir acciones como la duración de los *Sprints*, qué herramientas de comunicación se van a utilizar, cuál va a ser el procedimiento para realizar las entregas o cualquier proceso que se tenga que definir.

Es importante recordar el principio *agile* que habla de destacar siempre a las personas y sus interacciones sobre los procesos y las herramientas. Se debe mantener siempre la máxima

simplicidad en la organización del proceso, ya que garantizará estar centrados en el foco del proyecto o producto y no perder la atención en el seguimiento de procesos complicados. Como dijo Leonardo da Vinci, “la simplicidad es la sofisticación suprema”.

Nota:

Existe una colección enorme de acrónimos que giran alrededor del concepto de mantener soluciones lo más sencillas posibles del producto, que aparecen constantemente referenciadas en las metodologías ágiles. Ejemplos de estos acrónimos son: YAGNI (You Aren’t Gonna Need It), KISS (Keep it simple and Short) o DRY (Don’t repeat yourself).

Desde un punto de vista ejecutivo, hablando de la propia ejecución de las tareas para llevar a cabo el proyecto o el producto, el equipo tiene que definir o coordinar cómo va a trabajar para maximizar su sincronización. Aquí es donde se fijan los estándares, principios, reglas o criterios que el equipo va a compartir.

Por ejemplo, si se analiza el desarrollo software, un equipo debe definir cuáles van a ser sus estándares de codificación. Haciendo esto, el resultado de programación de todos los miembros del equipo será similar y fácilmente entendible por sus compañeros. El fijar estas reglas o principios tiene como objetivo minimizar las discusiones internas o el tiempo perdido en dudas relacionadas con elementos, que no sean de la propia ejecución de las tareas del proyecto o producto.

“No tengo claro qué hacer”. Gestionando la incertidumbre

Una vez creado el equipo y definido qué se tiene que hacer, puede existir cierta incertidumbre. Esta incertidumbre viene del salto que pueda existir entre el plano de definición de los requisitos, que se ha realizado en la creación del *Product Backlog,* y la forma de ejecutarlo. Puede que no se tenga una idea clara de cómo ejecutar los elementos que se están solicitando para la creación del producto o la ejecución del proyecto. Para resolver esta incertidumbre, durante el *Sprint* 0 se puede hacer un análisis inicial de la viabilidad de la solución.

Para hacer este análisis, se puede realizar un diseño inicial de lo que se quiere construir. En términos de software, se hablaría de un planteamiento inicial de la arquitectura, pero, si se tratara de construir una casa de madera para pájaros, sería un boceto de los planos de la casa.

Nota:

Se habla de planteamientos iniciales, un esbozo para clarificar y resolver incertidumbres. Estos diseños serán evolutivos e irán cambiando Sprint a Sprint con los avances en el proyecto y los nuevos requisitos emergentes.

Con el objeto de realizar este análisis de viabilidad o resolver incertidumbre, también se pueden hacer pruebas de concepto que vayan más allá de los meros diseños. Crear un pequeño prototipo o prueba de concepto sobre alguna idea poco clara del Product Backlog puede ayudar al equipo a ver más clara la solución final y estimar de manera correcta la funcionalidad del Backlog. A estas pruebas de concepto prácticas que tienen como objetivo aprender y probar un elemento del Backlog se les suele denominar Spike, como ya se vio en la anterior sección. Los Spikes se harán de manera previa a la ejecución de una funcionalidad del Backlog, sobre la que el equipo tenga un alto grado de incertidumbre.

Definiendo el plan maestro. Creando el release plan

Es muy importante definir qué y cómo se va a hacer un producto o proyecto, pero no hay que olvidar el tener claro cuándo se va a hacer. Para poder planificar e informar de cuándo se van a hacer las cosas, existe una planificación detallada que se denomina plan de entrega o *release plan*. ¿Cuándo se tiene que crear este plan? ¿En el *Sprint* 0? La respuesta es sí y no, como ocurre en muchos de los conceptos de *Scrum*. El plan de entregas es algo que se empezará a gestar en el *Sprint* 0, pero tendrá una gran adaptabilidad durante todo el ciclo de vida del producto o del proyecto. Por norma general, se tiene que revisar el plan después de cada *Sprint*.

¿Por qué no se puede partir con un plan cerrado desde el inicio para evitar sorpresas? Pues porque precisamente *Scrum* es un método que asume que siempre ocurrirán sorpresas durante el proceso y se adapta a lo inesperado. Dado que *Scrum* parte de la base de tener requisitos que puedan cambiar con el tiempo, no tiene sentido definir un plan de entregas que no pueda igualmente cambiar.

Para empezar a hacer un plan de entregas, es necesaria una información de base para trabajar con ella.

Lo primero que es necesario es tener una base de requisitos en el *Product Backlog*, aunque con la poca definición de los temas o *themes*. Los temas, como se mencionó en el capítulo anterior y se verá más adelante en detalle, son categorías de alto nivel que engloban colecciones de requisitos.

Lo segundo que necesitamos es, precisamente, la necesidad de crear una entrega. Parece obvio, pero muchas veces se fuerzan a crear entregas de algo que no tiene sentido por el simple hecho de hacerlo. El tener una necesidad o razón para hacer una entrega (interés en salir rápido al mercado para adelantar a la competencia, hacer una entrega a cliente para generar confianza o cualquier otra razón válida) es importante para fomentar el compromiso del equipo.

Una vez se tienen estas dos premisas, es el momento de planear una entrega. Para poder hacerlo, se tendrá que jugar con las tres variables principales en la gestión de proyectos o procesos: Tiempo, Recursos y funcionalidad-características.

Nota:

A este triángulo se le denomina "el triángulo de hierro".

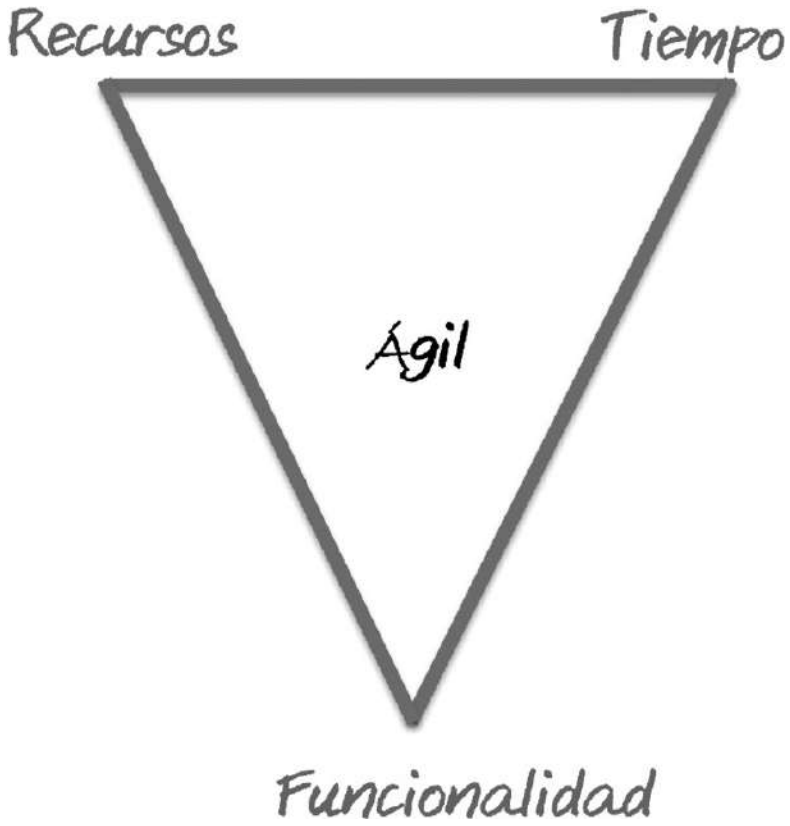


Figura 3.6. Triángulo de hierro.

Las tres variables de gestión de proyectos no son independientes. Si se modifica una de ellas, afectará a las otras dos indirectamente. Si se parte de que se va a tener unos recursos conocidos y constantes para la realización del proyecto, las dos variables con las que se podrá jugar serán el tiempo y la funcionalidad o alcance.

Nota:

Es importante tener claro que no se pueden fijar las tres variables con valores predeterminados, sino que al menos uno de los valores debe ser libre para que se adapte a los otros dos.

Hipótesis A. Se parte de que lo que se quiere hacer es una entrega en una fecha

determinada. Esto implicará que, en esa fecha, se tendrá un subconjunto de los elementos definidos en el *Product Backlog* por orden de prioridad. ¿Qué funcionalidades? Dependerá del equipo.

Hipótesis B. Se parte de que lo que se quiere tener es un subconjunto de funcionalidades determinado. Esto implicará que, para cubrir esa funcionalidad, habrá que esperar a una fecha determinada. ¿Qué fecha? Dependerá del equipo.

Depender del equipo significa que, para unos determinados recursos, y por tanto un coste conocido, el equipo puede sacar adelante una cantidad determinada de trabajo durante un *Sprint* o iteración de trabajo. Decir esto es lo mismo que hablar de la capacidad del equipo durante una limitación temporal fija y conocida. El equipo tiene para ese marco temporal lo que se denomina velocidad. La velocidad, que se explicará detalladamente en el capítulo 5, es un número que se obtiene de sumar los valores de la estimación del esfuerzo que se habían supuesto al inicio del *Sprint* para cada uno de los ítems terminados del *Backlog*. Por lo tanto, para poder definir un plan de entregas, es necesario conocer la velocidad del equipo y que los elementos del *Backlog* estén estimados.

Por esta razón, el plan de entregas es dinámico, porque depende de la velocidad, estimaciones y prioridades del *Backlog* que van cambiando *Sprint* a *Sprint*. Una vez se cuenta con el *Backlog* priorizado y estimado, a partir de la estimación individual de los elementos del *Backlog*, ya se podrá establecer con unas simples relaciones nuestro plan de entregas.

Dado un *Backlog* como el de la siguiente figura, en el que se tendrán los elementos A, B, C, D, E y F con unas estimaciones de 2, 2, 5, 8, 20, 20, respectivamente, y partiendo de la base en la que se tiene un equipo que tiene una velocidad de 10 puntos por cada *Sprint* de 2 semanas, ¿cómo se podría hacer un plan de entregas?

Figura 3.7. Backlog de ejemplo.

Por ejemplo, si lo que se quiere es hacer una entrega cada mes, eso implicaría que cada 4 semanas se avanzaría (2x 10) 20 puntos. Si se va sumando ítems del *Backlog*, se podría decir que en la primera entrega se liberarían los elementos A+B+C+D =17<20, en la segunda E=20 y en la tercera F=20.

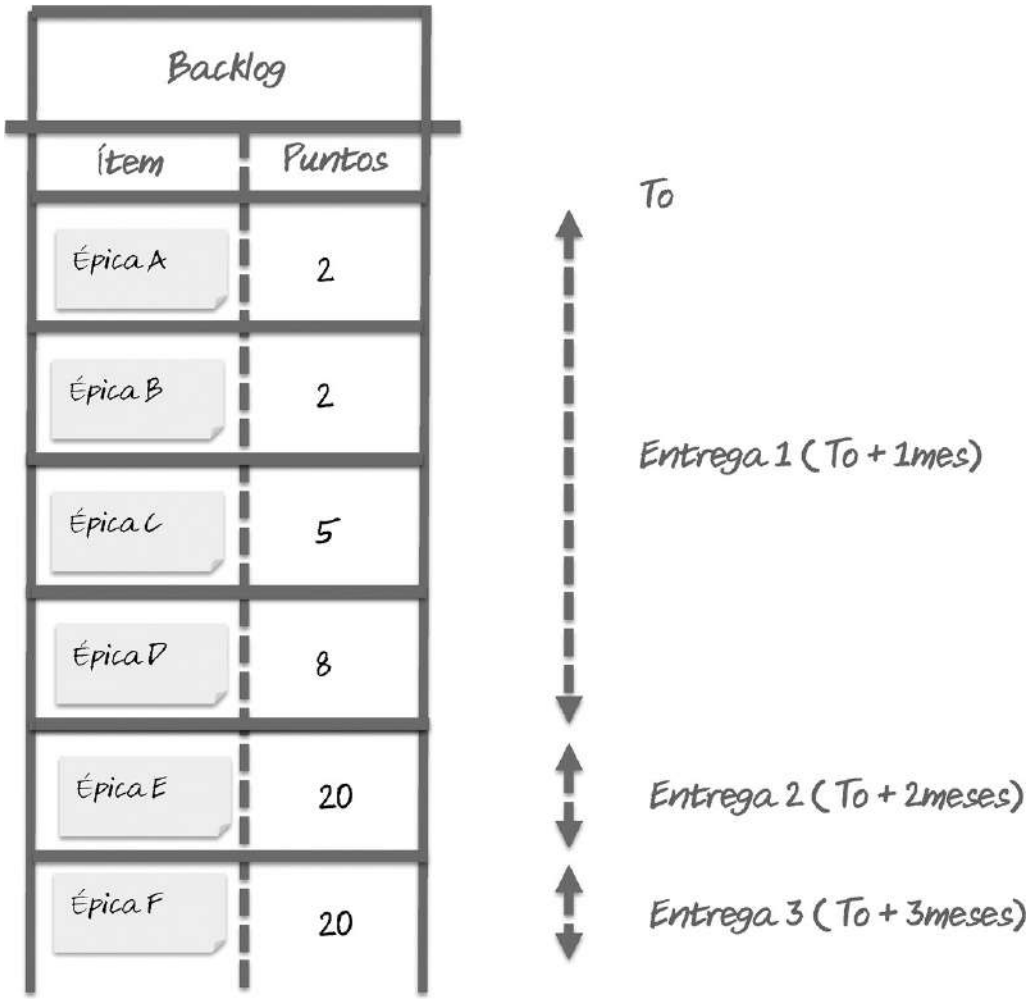


Figura 3.8. Plan de entregas I.

Si, por el contrario, lo que se quiere es fijar la funcionalidad y una entrega de A+B+C, otra de D+E y otra de F, eso implicará que se tendrá una entrega a las 2 semanas (A+B+C =9 <10), otra al mes y medio de la anterior (D+E=28 < 30) y, por último, otra después de un mes más (F=20<=20).

A la hora de crear el plan de entregas, siempre se tiene que tener en mente el concepto de producto incremental. Lo ideal es crear entregas del producto o proyecto que, de alguna manera, puedan ser autocontenidas y representen incrementos del producto que tengan sentido y aporten un valor de negocio.

En resumen, el plan de entregas se construye según la experiencia del equipo, por medio de sus estimaciones, ya que son quienes realmente conocen cuánto se tarda en hacer las cosas. Además, no será un plan fijo, se adaptará como todo elemento en *Scrum* después de cada iteración.

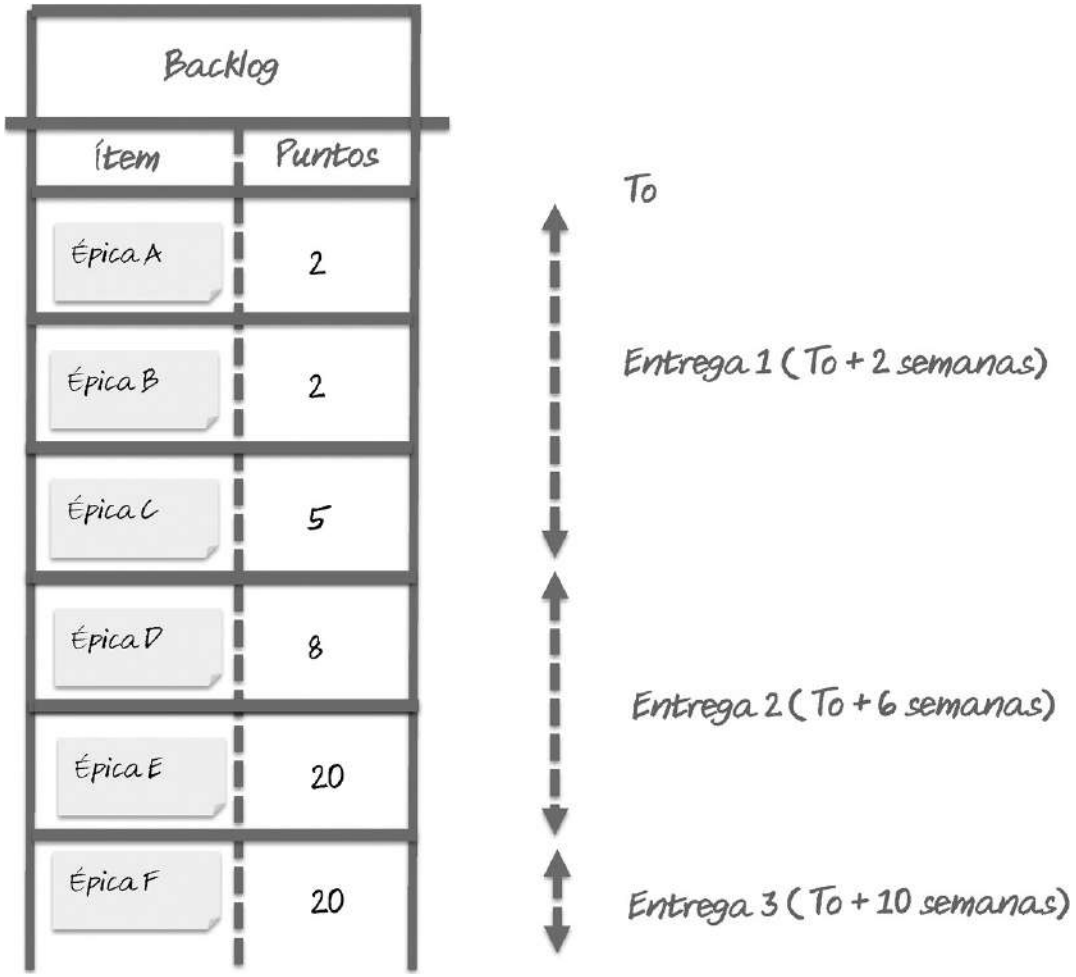


Figura 3.9. Plan de entregas II.

Conclusión

En este capítulo, se ha visto cómo poder empezar un proyecto utilizando *Scrum*, definiendo el rol del *Product Owner* como clave en el equipo *Scrum*. También se ha analizado la importancia de la creación de la Visión como base del *Product Backlog* y como herramienta de cohesión del equipo *Scrum*. Finalmente, se ha analizado cómo se debe crear un plan de entregas que represente de forma coherente el avance del trabajo.

En lo que respecta a nuestro libro, el *Sprint* 0 concluyó con una serie de acuerdos importantes para el trabajo en lo que concierne a la organización del mismo, las herramientas de comunicación, las convenciones y, en general, el diseño del marco de trabajo. También creamos un plan de entregas a partir de nuestra estimación inicial de capacidad de trabajo, un diseño preliminar del contenido, incluso una prueba de concepto a partir de textos y gráficas.

Se han dado los primeros pasos para lanzar el proyecto de escribir el libro que está en sus manos.

19 *Agile Product Management with Scrum: Creating Products that Customers Love*. Roman Pichler. Ed. Addison Wesley (2010).

20 *Strategy Maps: Converting Intangible Assets into Tangible Outcomes*. Robert Kaplan y David P. Norton. Ed. Harvard Business Press Books (2003).

21 *Crossing the Chasm*. Geoffrey Moore. Ed. Harper Business Essentials (1991).

4 ¿Qué tengo que hacer? El Product Backlog

En este capítulo aprenderá:

- La definición del *Product Backlog*.
- La composición del *Product Backlog*.
- Estrategias de priorización.
- Problemas del *Product Backlog*.

Buena parte del *Sprint* 0 está ya completado: marco temporal, recursos, condiciones, organización; pero hay una parte fundamental aún pendiente.

Necesitamos identificar las grandes divisiones del trabajo que vamos a realizar, ordenarlas, priorizarlas y documentarlas. En un proyecto tradicional, hubiéramos dedicado mucho tiempo a escribir unas especificaciones muy detalladas y precisas de nuestro libro. Estas serían seguramente inútiles porque, a medida que avanzáramos, aparecerían nuevos requisitos y cambiarían los que ya teníamos.

Vamos a aprovechar la potencia de *Scrum* para construir nuestra lista flexible, adaptable y dinámica de requisitos: nuestra Pila de Producto o *Product Backlog*. Veamos ahora cómo hacerlo.

La visión del producto o proyecto, como se ha comentado en el anterior capítulo, es el punto de partida para empezar a concebir la base en la que se va a trabajar. Lógicamente, por sí sola, es insuficiente para empezar a trabajar de manera práctica. Para iniciar el proceso de desarrollo del trabajo, es necesario definir un nuevo artefacto que permita alimentar al equipo que va a convertir la idea en una realidad. Este artefacto debe contar con definiciones concretas de lo que se quiere crear. Pero... ¿un nuevo artefacto? ¿No existen ya muchas formas de definir requisitos?

Durante años se han utilizado las definiciones de requisitos para especificar los productos o proyectos que se quieren llevar a cabo. Han funcionado de manera correcta, pero han presentado siempre una serie de problemas que han deteriorado la calidad de los resultados obtenidos. Los problemas más importantes identificados se resumen en la siguiente lista:

- Desconexión entre las personas que definen los requisitos y las personas que los llevan a cabo.
- Interpretación ambigua de los requisitos.
- Cambio de los requisitos desde que se definen hasta que se implementan.
- Necesidad de incorporar nuevos requisitos durante el ciclo de vida del desarrollo de un producto.

Con el objetivo de resolver estos problemas en las especificaciones tradicionales de requisitos, surge en *Scrum* el concepto de *Product Backlog*, PB o Pila de Producto. Este artefacto intenta definir un nexo de comunicación entre los clientes y el equipo *Scrum*. Este artefacto sirve de punto de encuentro para discutir, conversar, definir y aclarar las características que debe cumplir el producto o proyecto que se está llevando a cabo.

Nota:

Es importante fijarse en el tiempo de los verbos. Se usa "se está llevando a cabo" en lugar de "se va a llevar a cabo", ya que es un elemento que se mantiene vivo durante toda la duración del proceso de implementación de un producto o proyecto usando Scrum.

En este capítulo, se va a revisar el concepto, la estructura y buenas prácticas para explotar al máximo este artefacto de *Scrum*.

Presentando al Product Backlog

El concepto es muy sencillo. El *Product Backlog* es una lista de los requisitos que debe cumplir el producto que se quiere construir. ¿Esto es todo? Es lo básico. Pero, como dijo Albert Einstein, "lo importante es no dejar de hacerse preguntas". Si se continúa escarbando en esta definición se pueden hacer evidentes las claves del *Product Backlog*.

- **¿El PB contiene todos los requisitos del Producto?:** No, son los que se conocen en un momento dado del proceso. Los requisitos se descubren y emergen constantemente, así que el *Product Backlog* de ayer posiblemente no sea el de hoy. Los requisitos surgen para aportar valor de negocio a lo que se está desarrollando. Por esta razón, su aparición en cualquier momento es más que bienvenida, ya que mejorarán la calidad del producto.

Nota:

Normalmente el Big-Bang de los requisitos en el PB no aparece de la nada y es la consecuencia lógica de haber estado trabajando en la visión del producto durante el Sprint Zero.

- **¿Es una lista ordenada o desordenada?:** Es una lista ordenada por prioridad. De mayor prioridad a menor. Esta prioridad la define el *Product Owner* o responsable del producto, que es el dueño de la pila de producto. La definición de esta prioridad se hace en función de la relevancia desde el punto de vista del producto, la coherencia para ir creando un producto incremental o la incertidumbre de algunos requisitos.
- **¿Quiénes crean estos requisitos?:** Cualquier persona involucrada en el proyecto, con el beneplácito del responsable del *Backlog*, el PO, puede crear elementos en él. Estos requisitos se discuten, trabajan, aclaran y estiman por todo el equipo *Scrum*. Son requisitos creados de forma colectiva y colaborativa.
- ¿**Estos requisitos son de distintos tipos?:** Los requisitos pueden tener tipos, esto suele ser a gusto del equipo. Lo mínimo que debe tenerse es una división de requisitos funcionales y no funcionales (u operacionales). Los requisitos funcionales se conocen como historias de usuario que identifican una situación funcional del producto. Esta se describe desde el punto de vista de un usuario que desempeña un papel determinado.

Los requisitos no funcionales están relacionados con cualidades que son necesarias para el producto y no se pueden definir mediante historias de usuario.

Por ejemplo, si se quiere construir un columpio, una historia de usuario podría ser: "Como usuario, me gustaría disponer de un elemento que me permita balancearme respecto a un plano vertical 30º cuando me impulso con los pies". Que el columpio pueda soportar usuarios con un peso de hasta 80 kilos sería un requisito operacional.

- **¿Tienen los requisitos alguna otra organización, además del tipo?:** Cada equipo puede tener su propia organización, pero en la literatura sobre el tema se suele hablar de una estructuración en Temas (*Themes*), Épicas (*Epics*) e Historias de usuario (*User Stories*).

  Las historias de usuario se han definido como requisitos funcionales expresados desde el punto de vista de los usuarios. Las historias de usuario son ítems realizables dentro del marco de un *Sprint*. Cuando el caso de uso que representa una historia de usuario es tan grande que no puede ser abordado dentro de un *Sprint* se denomina épica. Normalmente, es necesario romperla en varias historias de usuario que puedan ser realizadas de manera más cómoda dentro de un *Sprint*.

  Finalmente, los temas son etiquetas que permiten agrupaciones de historias de usuario y épicas que hablan de un mismo tema o parte del producto.

Nota:

Tener el PB segmentado ayuda a trazar el trabajo que se está realizando o incluso dividir el trabajo en grupos por temática.

- ¿**Qué nivel de detalle tienen estos requisitos?:** Depende. Su nivel de detalle va en función de su posición en el *Backlog*. Los elementos que tengan más prioridad y, por lo tanto, estén colocados en una zona superior del repositorio estarán muy detallados. Los que estén más alejados de la parte superior estarán menos detallados y podrán ser más ambiguos.

  La razón es sencilla: los elementos que se encuentran en la parte superior del *Backlog* son los ítems que posiblemente entren en la siguiente iteración o *Sprint*, por lo que necesitan estar muy detallados para que el equipo pueda trabajar con ellos. Por el contrario, los elementos más lejos de la superficie se tardarán más en realizar, por lo que es mejor no definirlos evitando tomar decisiones sobre ellos hasta no tener que implementarlos. Esta espera garantizará que no sea necesario cambiar demasiado esos requisitos si, durante la realización del proyecto o producto, su contexto cambia. Esta estrategia se conoce como *last responsible moment* o último momento de responsabilidad.
- **¿Cómo se plasma todo esto en algo utilizable?:** Cierto, la pila del producto debe convertirse en algo real sobre lo que se pueda trabajar y discutir. El cómo se

implementa depende del equipo. Hay equipos que utilizan un tablón con los ítems en *post-it*, otros utilizan hojas de cálculo o pueden utilizarse herramientas informáticas desarrolladas para este propósito, como se puede observar en la imagen.

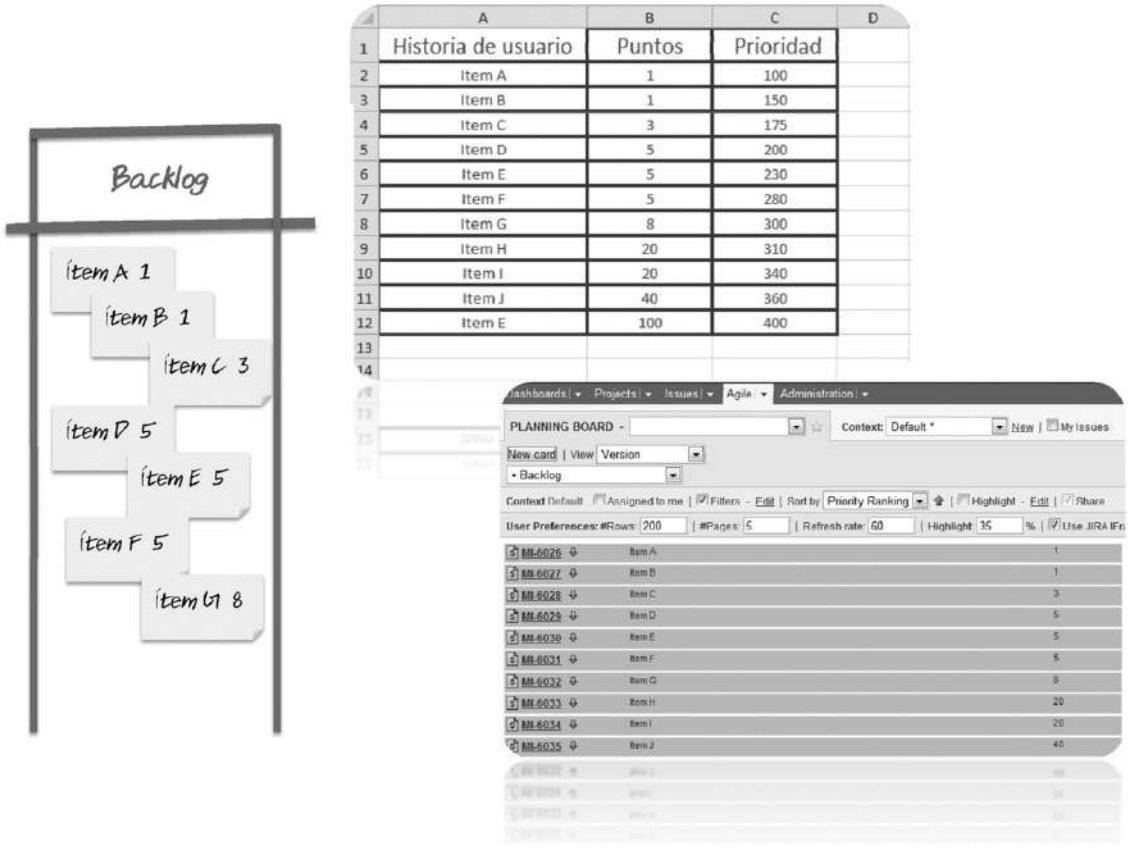


Figura 4.1. Distintas alternativas para mantener el Product Backlog.

Independientemente del formato del *Product Backlog* que se utilice, lo importante es que sea algo que esté al alcance del equipo en todo momento y con el que todos los miembros se sientan cómodos. Si no ocurre esto, el *Product Backlog* se dejará de usar como base del trabajo con consecuencias negativas sobre el proceso global.

Buceando en el Product Backlog

Dentro de un *Product Backlog* y dependiendo del tipo de proyecto, nos podemos encontrar con distintos tipos de elementos. Cada equipo puede adaptar el *Product Backlog* a sus necesidades, pero es importante que no se centre solo en requisitos funcionales y que también recoja los requisitos no funcionales.

Nota:

Los requisitos no funcionales no tienen por qué aparecer como un ítem en un Backlog específico y pueden ser un criterio de aceptación o comentario de un requisito funcional.

Para crear un buen *Backlog*, sus cimientos, los ítems que lo conforman deben ser sólidos. Una buena medida para conocer la solidez de estos criterios es analizar si cumplen la regla conocida como 3C o de las tres C. Este criterio enuncia las características que un ítem del *Backlog* debe cumplir:

- C*ard*: Su redacción debe poder entrar en una tarjeta o **c**artulina de tamaño reducido. La consecuencia es que debe poder ser resumido en un espacio pequeño sin perder sentido.
- **C***onversation*: Debe ser el resultado de la **c**onversación y negociación entre el responsable del producto y el equipo. La consecuencia es que los elementos no aparecen por arte de magia y son el resultado de iteraciones y explicaciones dentro del equipo.
- C*onfirmation*: Su cumplimiento debe ser de fácil **c**onfirmación. La consecuencia es que los elementos del *Product Backlog* deben contar con un criterio que indique cuándo han sido cumplidos y terminados.

Si los ítems cumplen estos criterios facilitarán su gestión, mantenimiento y realización, eliminando extensas reuniones de planificación o malentendidos entre los equipos de producto e implementación.

Es importante no confundir los criterios de las tres C con los criterios de aceptación. Los criterios de aceptación son supuestos, comentarios o condiciones que se imponen a la realización de un elemento del *Backlog* para considerarlo aceptado. Por ejemplo, recuperando el caso que se mencionaba anteriormente sobre el columpio y su historia de usuario: "Disponer de un elemento que me permita balancearme respecto a un plano vertical 30º", un posible criterio de aceptación podría ser que "si se ponen los pies en el suelo, el columpio debería pararse". Este criterio impone una condición o matiz para cumplir la funcionalidad buscada en la historia de usuario. Muchos equipos llevan más allá estos criterios de aceptación para los elementos del *Backlog* y, en vez de expresarlos como una simple frase, los estructuran y organizan un poco más. Por ejemplo: "Partiendo de una condición inicial en la que el columpio se esté moviendo, pueden ocurrir las siguientes cosas: si se ponen los pies en el suelo con fuerza, el columpio se tiene que parar de golpe; si se ponen los pies con fuerza en el suelo, el columpio se irá frenando hasta pararse; si se ponen los pies con poca fuerza en el suelo y luego se quitan, el columpio se habrá frenado pero seguirá balanceándose...". Como se puede observar, para definir un requisito se definen sus posibles comportamientos o las pruebas que deberían ocurrir cuando estuviera terminado. Desarrollar un requisito conociendo previamente las pruebas que deben pasarse para que se considere como cumplido se conoce en el mundo software como ATDD, *Aceptance Test Driven*

Development (Desarrollo guiado por los test de aceptación) o su evolución BDD, *Behavior Driven Development* (Desarrollo guiado por comportamiento). El concepto BDD se desarrollará en el capítulo17.

Dependiendo de los tipos de elementos, en el *Product Backlog* se puede contar con herramientas adicionales para garantizar su calidad.

Para los requisitos no funcionales se debe procurar de forma especial que contengan un criterio de aceptación concreto y realizable y que no deje lugar a la duda. Por ejemplo, para el columpio con su requisito no funcional sobre el peso soportado, se debe ser concreto en el límite soportado y no hablar en abstracto.

Para las historias de usuario, se recomienda usar la regla nemotécnica anglosajona "INVEST" para asegurar que una historia de usuario está definida con buena calidad.

INVEST define que una historia de usuario debe ser:

- I*ndependent* (**I**ndependiente): Autocontenida y no depender de la realización de otra historia de usuario.
- **N***egotiable* (**N**egociable): Mientras no esté dentro de un *Sprint,* puede ser discutida, negociada y cambiada las veces que sean necesarias.
- **V***aluable* (**V**aliosa): Aportar valor al usuario y al negocio.
- **E***stimable* **(E**stimable): Poder ser comprensible y su esfuerzo estimable por el equipo.
- **S***ize or* **S***mall* (**S**intética): Tener un tamaño óptimo para priorizarla e incluirla en una iteración de trabajo.
- T*estable* (Probable): La historia de usuario debe contar con una serie de pruebas o criterios de aceptación que permiten asegurar que satisfacen su especificación.

Una buena práctica que puede ayudar a homogeneizar la información de todas las historias de usuario suele ser definir una plantilla. Existe una plantilla que se usa comúnmente y consiste en definir una historia de usuario de la siguiente manera:

"*Como [usuario desempeñando un papel] me gustaría [deseo o funcionalidad] de tal manera que [beneficio o valor que se aporta al usuario]*".

Nota:

Es importante definir el rol del usuario, ya que permite identificar los distintos beneficios que el producto aportará a los usuarios segmentados por sus roles. También ayuda a organizar los equipos de trabajo implementando historias de usuario por roles.

El Product Backlog "Top model"

En el anterior apartado, se ha visto la radiografía de un *Product Backlog* por dentro, analizando cómo tienen que estar sus ítems para considerarlos de calidad óptima. Aunque se suele decir que la belleza está en el interior, en ese caso unas raíces fuertes no garantizan un árbol sano. En esta sección, se van a repasar las claves para tener un *Product Backlog* atractivo. Un *Backlog* DEEP. DEEP es el acrónimo inglés de **D***etailed Appropiatetely* - **E***mergent* - **E***stimated* - **P***riorized*, que afortunadamente tiene traducción en castellano con las mismas siglas: Detallado - Emergente - Estimado - Priorizado.

- Con **Detallado** se refiere a que las historias de usuario deben estar definidas de tal manera que el equipo pueda entenderlas y estimarlas. Es importante destacar la segunda parte de la cualidad, la que indica que este detalle debe ser el apropiado. Con apropiado se quiere recalcar que solo los elementos que estén más cercanos a la parte superior del *Product Backlog* son los que tienen que estar más detallados, mientras que el resto de los elementos puede permanecer con un menor detalle hasta que suban dentro del *Backlog*. La parte superior del *Backlog* indica cuáles son los elementos que con más alta probabilidad se implementarán antes, de ahí la razón de tenerlos preparados y detallados.
- Con **Emergente** se quiere destacar el carácter dinámico de un *Product Backlog*: puede cambiar en cualquier momento para adaptarse al contexto y necesidades del producto o proyecto en desarrollo. “Emergente” significa que en cualquier momento pueden aparecer nuevas historias de usuario.
- Con “**Estimado**” se pretende reflejar que un *Product Backlog* va más allá de la lista de cosas que se deben hacer para cumplir la visión de lo que se quiere crear. Los elementos de un *Product Backlog* deben estar estimados. Esta estimación habla de la comparación entre los elementos del *Product Backlog*. Si se tienen dos historias de usuario estimadas en valor 2 significa que son del mismo orden de magnitud, mientras que si existe otra de valor 1 implicará la mitad de esfuerzo que las anteriores. Aunque ese valor de estimación, en sí, no sea un valor tangible en términos de tiempo o coste, una vez se conozca cuántas historias de usuario en media se realizan en un determinado marco temporal se tendrá una potente herramienta de estimación. Se podrá usar esta predictibilidad para prever cuánto trabajo estará implementado en un momento concreto de la ejecución.
- Por último, **Priorizado** habla de la necesidad de priorizar los elementos del *Backlog*, lo cual ayudará a saber qué elementos tienen que estar detallados, dónde colocar los nuevos ítems emergidos y ayudará a conocer cuándo se tendrá una colección de funcionalidades gracias a la estimación de estas. La priorización es uno de los procesos clave para ir entregando progresivamente el valor adecuado en el incremento de cada iteración. En la siguiente sección se analizará detalladamente el proceso de priorización.

Una imagen que representaría a un *Product Backlog* bien formado sería la de un iceberg como se suele usar en la documentación relacionada con *Scrum*. Pequeño en su parte superior, con sus historias de usuario reducidas, de tamaño medio en su parte central y de un tamaño mucho más grande y desconocido por debajo del agua con sus épicas. Así debería ser el *Product Backlog*, ya que se desconocerá qué es lo que puede venir durante el desarrollo del proyecto.

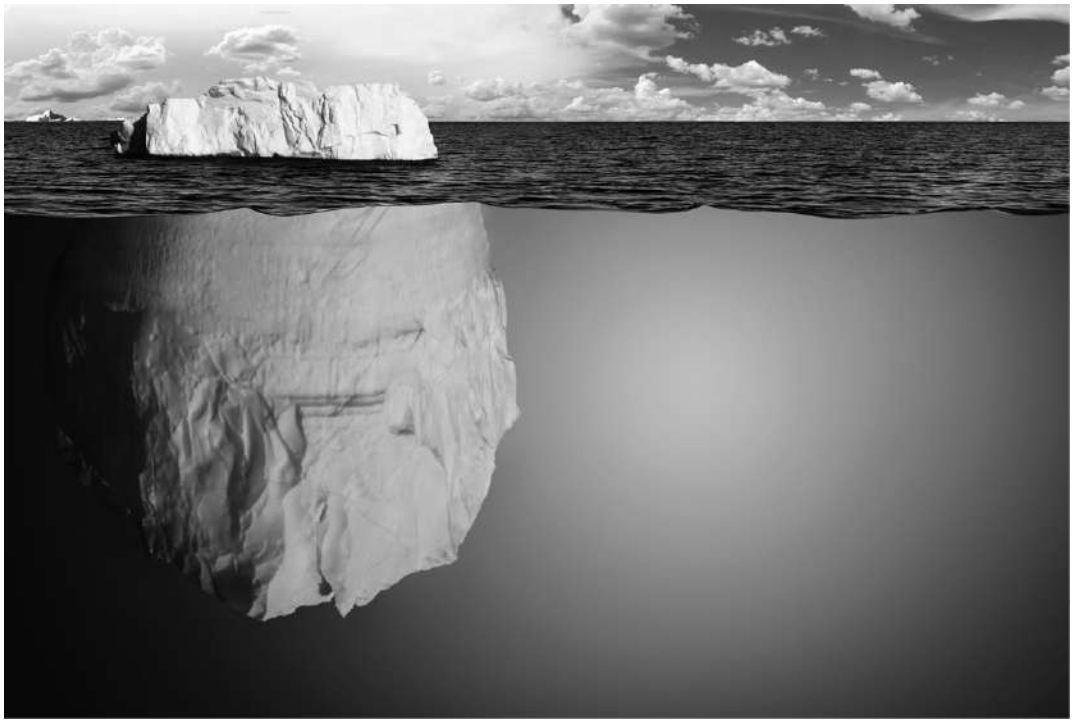


Figura 4.2. El Product Backlog como iceberg.

Mantener al *Product Backlog* DEEP en forma es la responsabilidad más importante del *Product Owner*, ya que es el instrumento que tiene a su alcance para velar por la integridad y calidad de lo que se va a crear.

Priorizando el Product Backlog. ¿Por dónde empiezo?

Como se ha visto en la anterior sección, un *Backlog* priorizado sirve para organizar el plan del equipo y conocer cuál va a ser la ruta de trabajo a corto plazo. La primera pregunta importante es saber qué hay que priorizar o a qué nivel hay que hacerlo. Lo que se tiene que priorizar son los elementos que se tienen en el *Product Backlog*. Estos elementos son las historias de usuario y los requisitos no funcionales. Como ya se ha comentado, también se tienen las épicas, que son elementos demasiado grandes para entrar en un *Sprint*, que suelen tener un etiquetado que ayuda a agruparlos, denominado tema. Una buena recomendación es

priorizar desde el punto de vista de épicas y temas. Cuando se desgrana hasta el nivel de historia de usuario que podría entrar en un *Sprint*, el detalle es tan bajo que resulta muy complicado comparar los elementos entre ellos. Quedándose en un plano de más alto nivel en los requisitos, se pueden analizar qué elementos pueden tener impacto en el producto de una manera mucho más transversal y más sencilla.

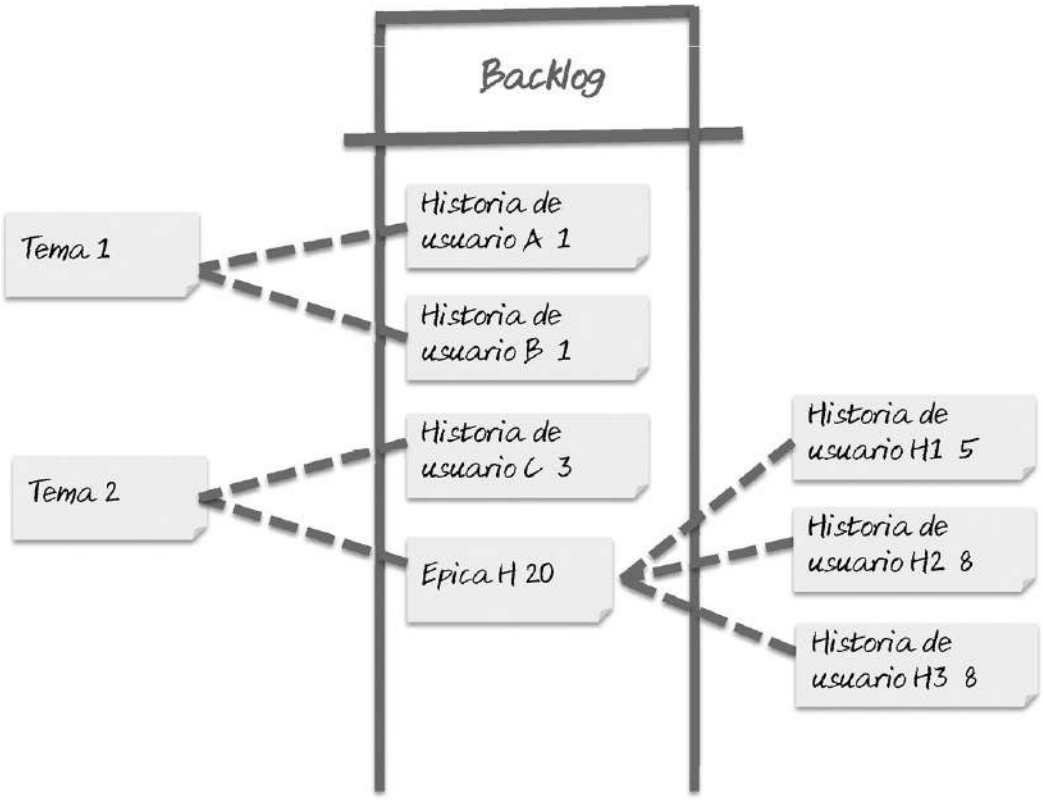


Figura 4.3. Historias de usuario, épicas y temas en un Backlog.

Por ejemplo, si se está hablando de hacer una reforma en una casa y se plantea la prioridad de colocar unos azulejos en una pared y poner el suelo del baño, posiblemente no se sea capaz de ordenar una tarea por encima de la otra. Si se habla de reformar el baño o el salón de una casa quizás sí se tenga un criterio más claro para priorizar.

El responsable de priorizar es el *Product Owner*, pero no tiene por qué estar solo en esta tarea. El *Product Owner* se puede valer de su equipo de producto, *stakeholders* o del propio equipo *Scrum* (se ha de recordar que está compuesto por el PO, *Scrum* Master y equipo de trabajo) para crear una prioridad que refleje una visión más universal del producto.

Nota:

En muchos proyectos, esta prioridad la fija una sola persona basada en decisiones subjetivas y no suele resultar efectiva. Cuanta más gente representativa para el producto participe en esta priorización y en más criterios objetivos se base, más valor se generará

con cada avance del proyecto.

Para realizar una buena priorización es importante definir un criterio sobre el que priorizar. Si no se define, no se podrá alinear la opinión de todas las personas que están participando en la priorización.

La priorización es una de las tareas más complicadas del proceso de *Scrum* y suele ser fuente de debate en los equipos. Para facilitar su ejecución, se explica en esta sección una colección de técnicas para que el lector las explore y elija la que más se adecúe a sus necesidades. Para más información sobre la priorización, es muy recomendable visionar la charla "Priorizando el *Product Backlog*[22]", impartida por Mike Cohn en el congreso "Agile 2008".

Priority poker o póker de prioridad sobre un criterio simple

Esta técnica es la más sencilla de todas. Se reúnen las personas que van a tomar parte en la priorización. Cada participante cuenta con una baraja de cartas que están numeradas del 1 al 9, siendo el 9 la mayor prioridad. Se define cuál va a ser el criterio de priorización general para el *Backlog*. Se toma cada elemento del *Product Backlog*, se enuncia y todos los participantes sacan una carta con su prioridad, sumándose el total. Cuando se ha hecho este proceso con todos los elementos del *Backlog*, se pueden ordenar todos los componentes de mayor a menor teniendo ya un *Backlog* priorizado. Nótese que la prioridad está basada en este caso en una media de la opinión subjetiva u objetiva, sobre un criterio de priorización determinado, de los participantes en la reunión.

La figura indica el orden de las prioridades en función de la suma de las puntuaciones. En este caso, ese orden sería D, E, A, B, C.

Ejemplos de estos criterios que se han comentado en la definición del *priority poker* pueden ser los dos siguientes:

- **Riesgo tecnológico:** Como se acaba de ver, la ordenación se hace con un criterio específico para definir el concepto de prioridad. Un posible criterio simple de ordenación es el riesgo tecnológico. Riesgo tecnológico se define como la incertidumbre de no conocer si existe la posibilidad de que no pueda llevarse a cabo un requisito por razones técnicas. En este caso, se dará más prioridad a todos los elementos del *Backlog* que puedan suponer un riesgo tecnológico que impida la realización del proyecto. De esta manera se asegurará la viabilidad del proyecto en un momento temprano evitando gastar recursos de manera innecesaria.
- **Valor de negocio:** Al igual que en el caso anterior, la ordenación sigue un criterio definido. En la ordenación de valor de negocio, de una manera generalista se le asigna un peso de valor de negocio a cada historia de usuario en función de ROI (Retorno de

la inversión) que se prevé que puede generar. De esta manera se desarrollarán al principio los elementos que puedan aportar mayor valor de negocio.

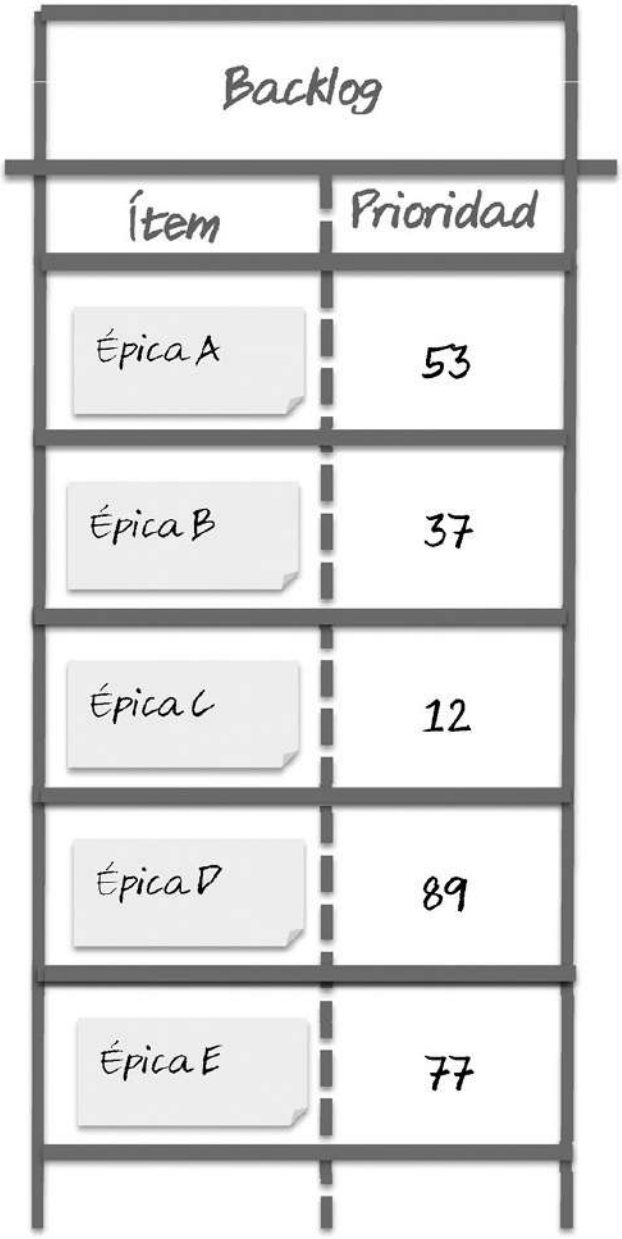


Figura 4.4. Tabla de puntos de priority ranking.

MoSCoW

En las anteriores técnicas se trata a todos los elementos del *Backlog* de una manera homogénea. La técnica MoSCoW se basa en la segmentación y agrupación de los elementos del *Backlog* para poder hacer una mejor priorización. MoSCoW en un acrónimo inglés cuyo origen está en los siguientes valores para los elementos del *Backlog*:

- M*ust* (El producto debe tenerlo): Si no se cumple este requisito, el proyecto podría ser cancelado.
- **S***hould* (El producto debería tenerlo): No es tan crítico como el anterior, pero sí muy valioso para el producto.
- **C***ould* (Estaría bien si lo tuviera. El producto podría o no tenerlo): Interesante pero no clave.
- W*on't* (El producto actual no contempla tenerlo): Estos requisitos se mantienen en PB pensando en el futuro. No se eliminan porque son una característica que podría contemplar el producto, pero no en este momento. Incluso puede darse el caso de que, en un momento dado del proyecto, se convierta en un *Must* o *Should*.

Conforme a estos criterios, los asistentes a una reunión de priorización deberían discutir qué letra asignar a los elementos del *Backlog*. Una vez las letras estén ordenadas con el orden correcto (M-S-C-W), se podrá aplicar sobre cada segmentación alguna de las técnicas anteriores para una reordenación interna.

Modelo de Kano

En el capítulo dedicado al *Sprint* 0 se analizó el modelo de Kano como metodología para la definición de la visión del producto. Este método también puede usarse para priorizar el *Product Backlog*. La priorización del *Backlog* en este caso se orienta a mejorar la satisfacción del cliente con el desarrollo que se va a realizar. Si se recuerda el modelo de Kano, la satisfacción del cliente se podía alcanzar con 3 tipos de características o atributos para el producto de las 5 que define el modelo:

- Las **básicas u obligatorias**, que son aquellas características que producen un alto nivel de insatisfacción del cliente si no están presentes, pero que, si lo están, producen al cliente un sentimiento neutro, ya que es lo que esperaba del producto. Por ejemplo, si se está redactando un libro y no se incluye un índice, generará insatisfacción al lector porque esperaba encontrarlo, pero, si lo tiene, no le va a producir ninguna satisfacción especial.
- Las de **rendimiento o lineales**, que son aquellas que producen más o menos satisfacción al cliente según haya más o menos de ellas. En el ejemplo de un libro se podría hablar de las imágenes y fotografías explicativas: cuantas más incluya, mayor satisfacción puede obtener el lector.
- Por último, están las **inesperadas o emocionantes**, que son aquellas que producen un sentimiento de novedad. Son los atributos de un producto cuya ausencia es neutra para el cliente, pero, cuando están presentes, producen un alto grado de satisfacción. Siguiendo con el ejemplo del libro, se podría hablar de un descuento 2x1 para comprar

otro libro. Si no está el descuento, al usuario le parecerá lógico, pero, si está el descuento, le producirá un alto grado de satisfacción.

Para aplicar la teoría de Kano a la priorización de un *Product Backlog* se tiene que identificar en él qué elementos son de tipo básico, cuáles de rendimiento y cuáles inesperados. Se deberían priorizar todos los básicos, casi todos los de rendimiento y alguno de los inesperados, por este orden. La manera práctica de hacer esta clasificación suele hacerse por medio de encuestas a posibles usuarios[23]. En las encuestas se deberá preguntar por los temas y épicas de dos maneras opuestas:

- La pregunta funcional: ¿Cómo te sentirías si esta funcionalidad estuviera presente?
- La pregunta disfuncional: ¿Cómo te sentirías si esta funcionalidad no estuviera presente?

Ambas preguntas se deberían responder con la escala de 5 respuestas del modelo de Kano, como se muestra en la siguiente figura.

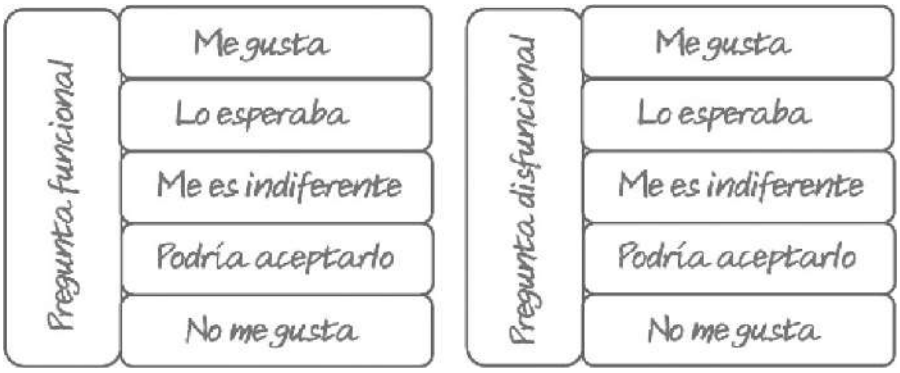


Figura 4.5. Preguntas y respuestas del modelo Kano.

Si se cruzan las 5 respuestas funcionales del modelo con las 5 disfuncionales, dará lugar a las 5 categorías del modelo: Básico, Lineal, Emocionante, Indiferente, Reversible, además de la opción Cuestionable, que habla de resultados que no son coherentes. Por ejemplo, si se tiene una épica en el *Backlog* a la que un usuario responde que esperaba encontrarla cuando se le pregunta “¿cómo te sentirías si esta funcionalidad estuviera presente?” y responde que no le gusta cuando se le pregunta “¿cómo te sentirías si esta funcionalidad no estuviera presente?”, aquí estará ante una épica básica, como indica la tabla. La tabla de referencia de los cruces se puede comprobar en la siguiente figura.

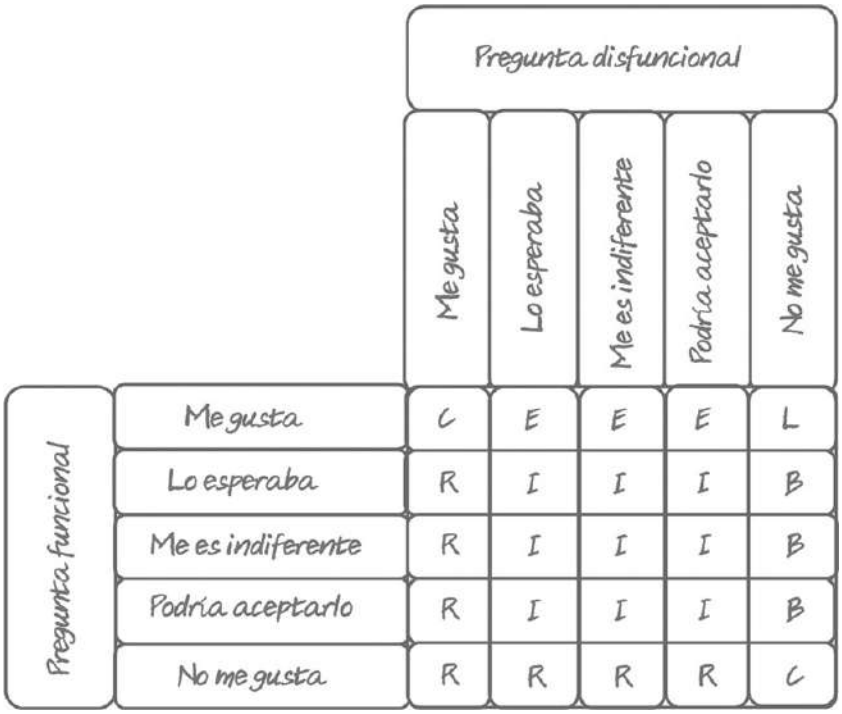


Figura 4.6. Clasificación de las respuestas en el modelo de Kano.

Finalmente, se tiene que extraer, conforme a los resultados de las repuestas, qué elementos del *Backlog* pertenecen a cada categoría para ordenarlos según la propuesta que se hacía con anterioridad. Para esto, se suman las respuestas de todos los usuarios por cada épica analizada para ver qué categoría es la que tiene más peso como indica la figura.

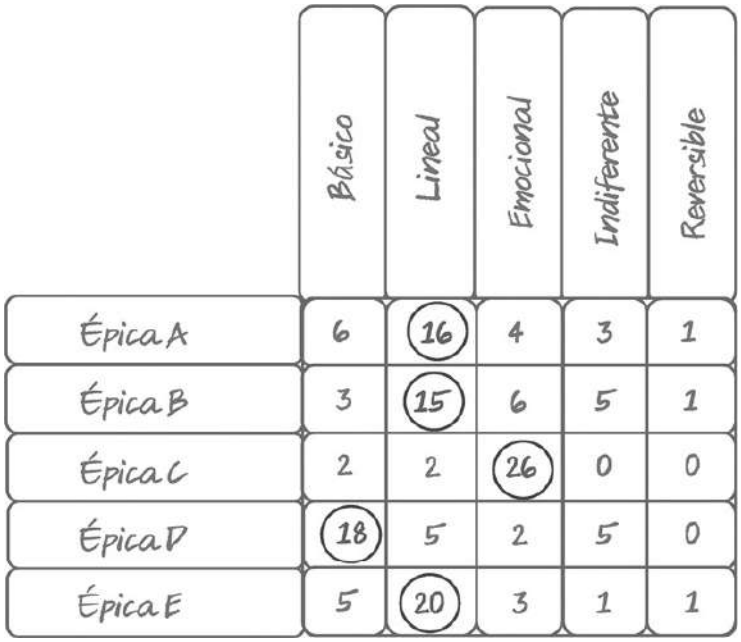


Figura 4.7. Pesos de las respuestas de los usuarios.

Recuerde:

Primero, todos los básicos; luego, gran parte de los lineales; y, finalmente, la inclusión de algún inesperado. Esto daría un orden, según el ejemplo, de D, E, A, B, C.

Criba de temas

El método de criba de temas permite priorizar los elementos del *Backlog* para ordenarlos a la hora de priorizar una nueva versión de un producto o entrega de un proyecto. Lo primero que hay que hacer es establecer unos criterios de decisión, la misión o metas que se quieren conseguir en la nueva entrega. Por ejemplo, mantener los clientes que ya tiene la empresa, ganar respeto, mejorar la imagen de seguridad de la compañía o cualquiera que se considere válido.

Nota:

Es bueno tener un buen número de criterios, pero tampoco hay que ser muy exigentes. 5 es un buen número de criterios para llevar a cabo esta técnica.

Además de definir los criterios de selección, se tiene que elegir un tema o épica base. Es un elemento que considerará obligatorio que esté en la siguiente versión o entrega que se quiera realizar. A este elemento base se le asigna un valor de 0 y se revisan el resto de los elementos del *Backlog* para el criterio de selección en comparación al base, preguntándose si la inclusión de este elemento mejoraría la entrega, si es igual de importante que el base o no aporta ningún valor especial respecto al caso base. En función de la respuesta, se le asigna valores +1, 0, -1 a ese elemento del *Backlog* que se esté comparando. Cuando se ejecuta esta comparación de todos los elementos del *Backlog* siguiendo los criterios de selección con el elemento base, se obtiene una tabla de medidas como la de la siguiente figura.

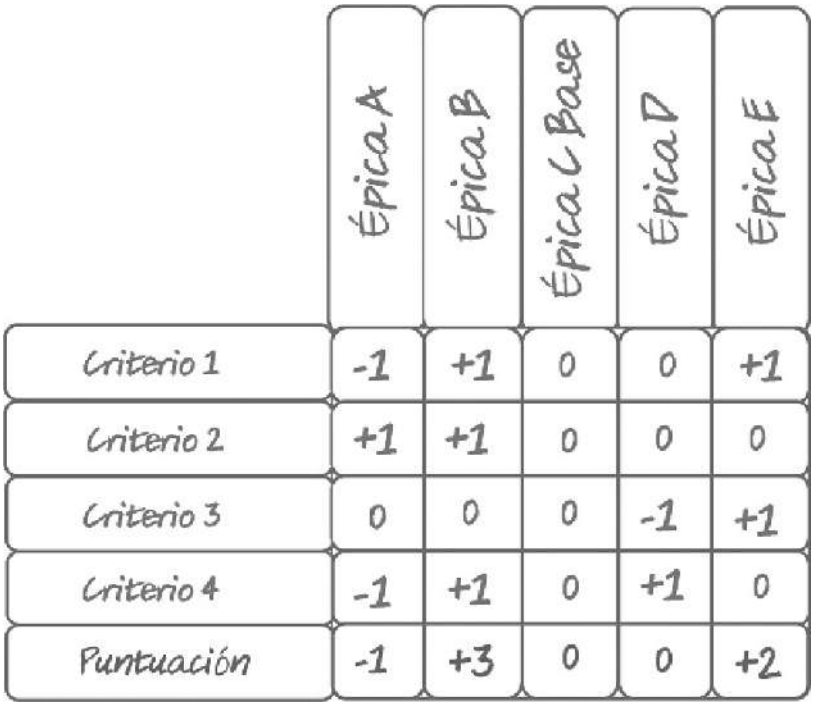


Figura 4.8. Tabla de cribado de elementos.

Su puntuación numérica ayudará a ordenarlos de forma objetiva. En este caso, la ordenación sería B, E, C, D, A.

Puntuación de elementos

La metodología de puntuación es muy similar a la de cribado. Su variante radica en asignar valores o pesos a los criterios de selección, ya que posiblemente no todos los criterios tienen la misma importancia. Al dar un peso porcentual a cada criterio y un valor numérico constante al caso base, se obtendrá un valor numérico como se muestra en la siguiente tabla, que ayudará a ordenar los elementos del *Backlog* cuando los criterios de selección pueden tener pesos distintos.

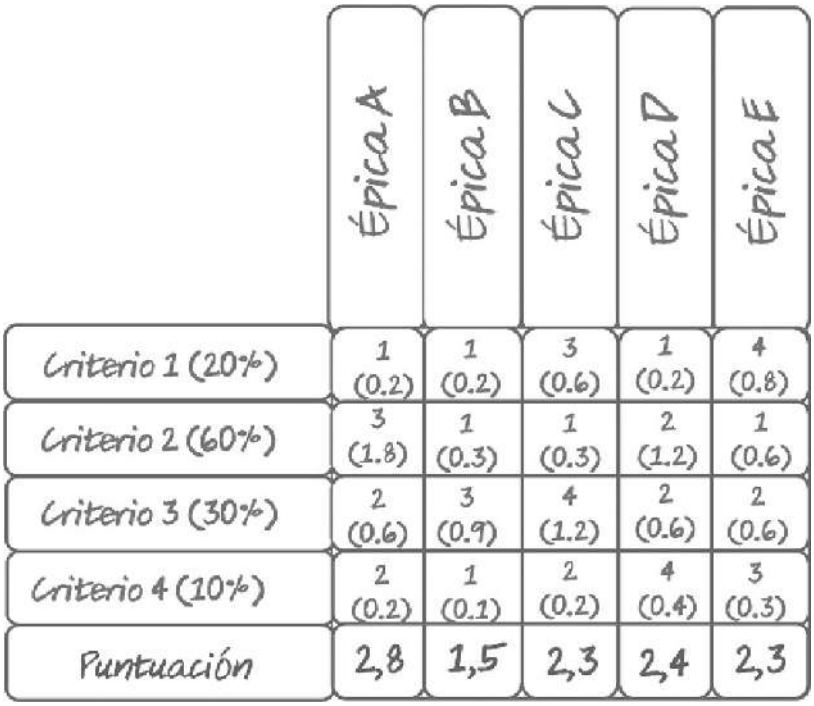


Figura 4.9. Tabla de puntuación de elementos.

Comprobando la agregación de valores en la figura, la ordenación sería A, D, C, E, B.

Peso relativo

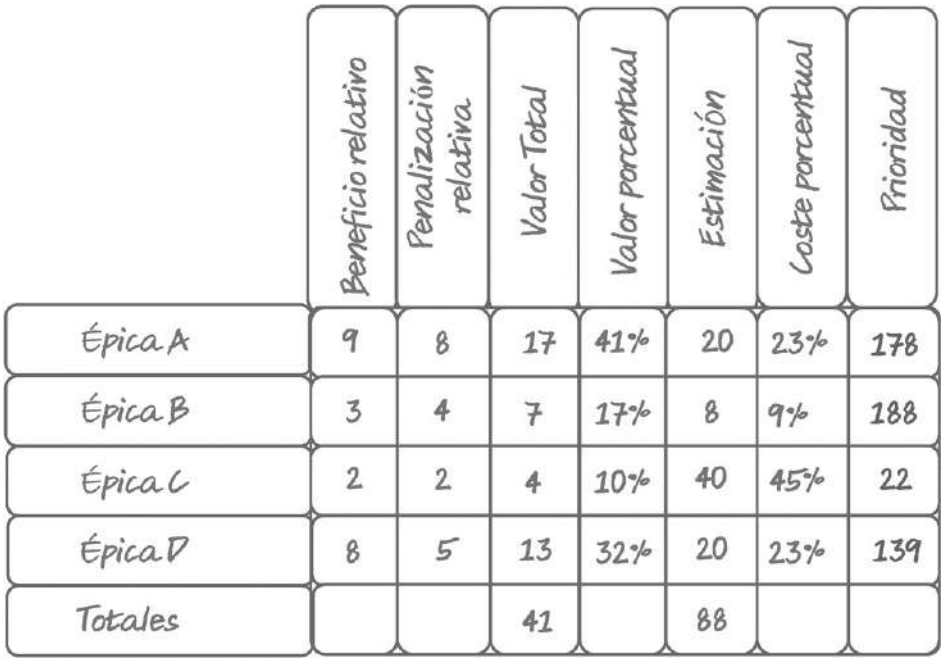
Por último, se va a analizar el método del peso relativo que se apoya en muchos conceptos de los anteriores métodos. Para todos los elementos del *Backlog* se estima el impacto de tener ese elemento del *Backlog* en el producto. Esta estimación se hace de 1 a 9, siendo 9 el impacto mayor. Se realiza el mismo proceso, pero analizando el impacto de no tenerlo. Esta valoración también se realiza desde el 1 hasta el 9.

Una vez se tienen ambas listas se calcula el Valor Total y el Valor porcentual de la siguiente manera:

- Valor total = Beneficio relativo + Penalización relativa
- Valor porcentual = Valor total del elemento / Suma (Valores totales)
  Una vez se han obtenidos estos valores, se recuperan las estimaciones que el equipo haya realizado sobre los elementos del *Product Backlog* y se calcula su Coste porcentual de la siguiente manera:
- Coste porcentual = Estimación de elemento/ Suma (Estimaciones)

Analizando la relación entre el Valor porcentual y el Coste porcentual, se podrá valorar de forma objetiva cuál sería la prioridad para poder primar los elementos que con menor coste porcentual puedan generar mayor valor porcentual. En la siguiente tabla se puede observar

cómo sería un ejemplo del análisis de los pesos relativos sobre varios elementos del *Backlog*.



	Beneficio relativo	Penalización relativa	Valor Total	Valor porcentual	Estimación	Coste porcentual	Prioridad
Épica A	9	8	17	41%	20	23%	178
Épica B	3	4	7	17%	8	9%	188
Épica C	2	2	4	10%	40	45%	22
Épica D	8	5	13	32%	20	23%	139
Totales			41		88		

Figura 4.10. Tabla de costes y valores porcentuales de los ítems del Backlog.

En función de los resultados, la clasificación en este caso sería B, A, D, C.

Me falta una E de DEEP

Como el lector habrá podido observar, cuando se ha hablado de un *Backlog* DEEP una de las E habla de la estimación, pero no se ha hablado del concepto de estimación en profundidad y de sus técnicas todavía. En el capítulo del *Sprint Planning* se dedicará una sección en exclusiva a hablar de este tema, por lo que se remite al lector a ese capítulo para que pueda cerrar el conjunto de las buenas características que debe cumplir un saneado *Product Backlog*.

Manos a la obra. Creando un Backlog

Una vez han quedado claros los conceptos sobre cómo debe construirse un buen *Product Backlog*, se va a hacer un repaso sobre el ejemplo de creación de este libro. Como se ha visto en el capítulo anterior, para construir un buen *Product Backlog* es necesario haber trabajado previamente sobre la visión de lo que se quiere construir. En nuestro caso, la visión o concepto es: la creación de un libro que resuma los conceptos básicos de *Scrum*, así como las

metodologías ágiles, y que pueda utilizarse, por ejemplo, como material de formación para impartir un curso de introducción a estas metodologías. El hecho de tener un concepto o visión ya ayuda indirectamente a sentar las bases del *Product Backlog* gracias a la definición de los temas o *themes* que agruparán en el futuro los elementos del *Backlog* de más bajo nivel. En nuestro caso, estos temas fueron: Métodos ágiles, *Scrum* y XP. Lógicamente los temas siempre son elementos muy amplios, pero ayudan a segmentar la información que se tendrá en el *Product Backlog*.

Una vez se tienen el concepto y los temas, es un buen momento para trabajar en la *user journey o* experiencia de usuario. La *user journey* es una agrupación de las experiencias que un usuario podría experimentar con el concepto que se quiere desarrollar. Da igual que sea un libro, una casa o un programa software. Esta colección de experiencias es muy importante, ya que de su análisis posterior se podrán extraer todos los requisitos funcionales que se quieren ofrecer con el concepto en desarrollo y todos los requisitos no funcionales que se desean que el producto cumpla.

En nuestro caso, se analizó la experiencia de uso que se quería ofrecer con el libro al lector. De todas las posibles alternativas para presentar la información, se optó por la descripción de una secuencia lineal de todos los procesos que tienen lugar en la metodología *Scrum* para que el lector o usuario pudiera seguir fácilmente el proceso. Sobre esa lectura lineal, se irían describiendo los distintos roles y artefactos que irán apareciendo en cada parte. Sobre esa secuencia lineal se decidió presentar una introducción de metodologías ágiles y acabar con un detalle de XP para ampliar el conocimiento. Este guion o *storyboard* es lo que se tomó como base o *user journey* para definir los elementos del *Backlog* segmentados en los temas.

Las historias de usuario que se definieron inicialmente fueron: Introducción, Revisión del proceso completo, *Sprint* 0, *Sprint Planning*, Iteración, *Review*, Retrospectiva, XP, *Scrum* en la empresa. Estas historias de usuario son muy amplias y luego se tuvieron que dividir en las historias de usuario que representaban las secciones de cada capítulo. Estos elementos son las épicas o *epics* que se han comentado en este capítulo.

Además de las propias historias de usuario, se tenía una colección de requisitos de usuario, muchos de los cuales se añadieron como criterios de aceptación y otros como propios ítems del *Backlog*. Estos requisitos no funcionales han representado criterios de estilo para la realización del libro o códigos de trabajo para la organización de las secciones. A continuación, se muestra un ejemplo de historia de usuario redactada: la historia que se denominaría introducción.

Como lector me gustaría que hubiera un capítulo de introducción en el que se sentaran las bases y conceptos de Scrum, de tal manera que cuando volvieran a aparecer en el libro puedan ser fácilmente interpretados.

Y, por supuesto, la historia de usuario debe contar con sus criterios de aceptación para que quede definida de manera completa:

- Debe describirse el contenido de las secciones del libro.
- Debe aparecer un resumen del concepto para recurrir a él de forma rápida.
- Se deben dar datos de los autores.
- Debe aparecer una referencia a cómo extender el conocimiento del libro.

Una vez se hubo definido las épicas, se priorizó el orden de ejecución de estas según el criterio de ordenación sobre la facilidad de cada uno de los autores para la redacción de los capítulos. Así el *Backlog* quedó ordenado conforme a los capítulos que tenían una estimación más baja. Fueron estos primeros capítulos los que se detallaron dividiéndolos en historias de usuarios más reducidas. Con el *Backlog* ya preparado después del *Sprint* 0, todo estaba listo para iniciar la primera iteración y comenzar a redactar el libro.

Doctor, mi Product Backlog está enfermo y no sé qué le pasa

Una pila de producto saludable y en forma es la mejor manera de garantizar un buen resultado del producto o proyecto que se esté desarrollando. También da una imagen de lo bien o mal que se está realizando el proceso *Scrum*.

En esta sección, se verán los síntomas que se pueden percibir en el *Backlog* de que algo no está marchando bien y algún posible remedio.

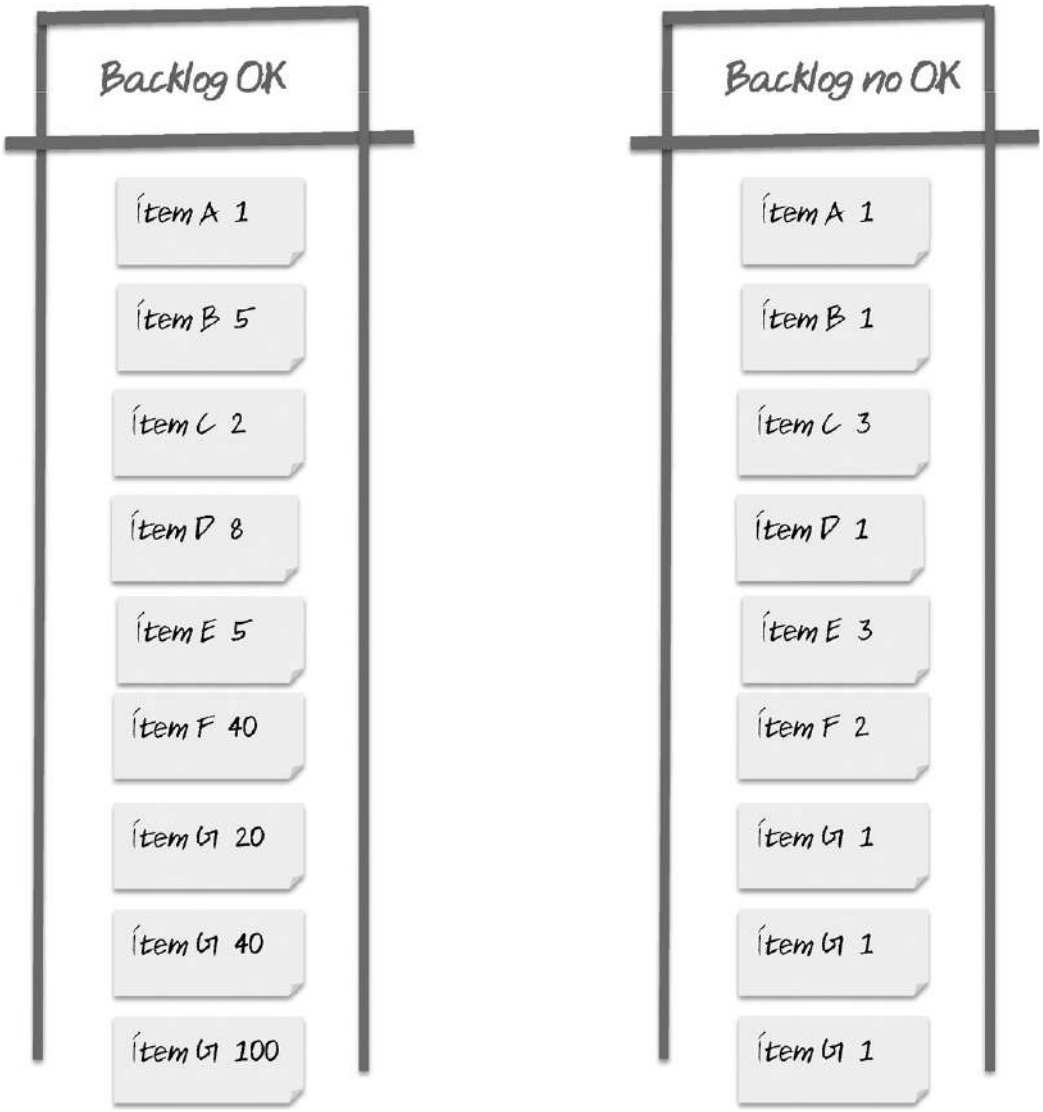


Figura 4.11. Doctor Backlog.

Un Product Backlog con sobrepeso

Si se encuentra en un momento dado con un *Product Backlog* que tiene cientos de elementos por hacer, muy divididos y detallados, deben saltar las alarmas de que algo no está bien. A simple vista, este síntoma es amigable, incluso alguien podría pensar que es positivo, pero detrás de esto se encuentra una especificación de requisitos encubierta. El tener una lista de requisitos pasada al *Backlog* hace que todo el producto o proyecto en el que se quiere trabajar esté definido desde el inicio con detalle. Elimina la potencia de *Scrum* a la hora de destapar los requisitos emergentes o las correcciones incrementales. En definitiva, lleva a un modelo tradicional en cascada sin quererlo.

Un problema frecuente que se asocia a esta enfermedad es la dificultad de priorización. Al tener un *Backlog* tan definido y una gran fragmentación de historias de bajo nivel, es muy difícil priorizar unas sobre otras.

Nota:

Cuando se tiene un Backlog excesivamente troceado y que dificulta su priorización, se puede hacer una reunión de segmentación asociando los ítems del Backlog a temáticas y priorizando las temáticas en bloque.

El mejor remedio es siempre recordar el principio DEEP (Detallado, Emergente, Estimado, Priorizado) y la foto de un *Product Backlog* en forma de un iceberg.

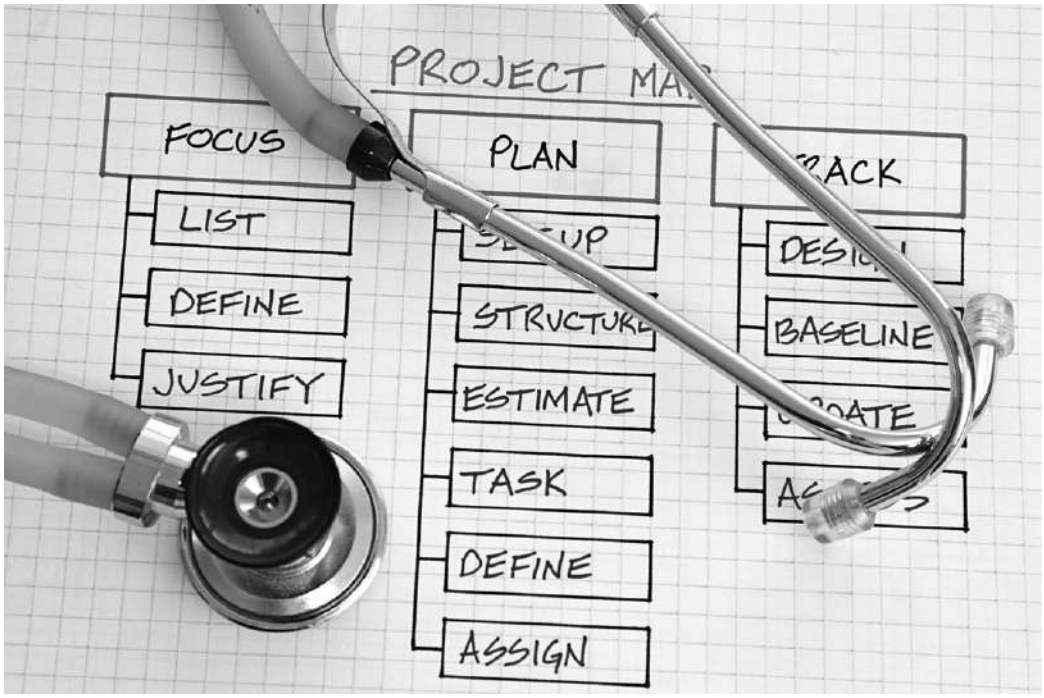


Figura 4.12. Backlog correcto vs. Backlog incorrecto.

Los ítems del *Backlog* que estén arriba serán los más prioritarios y detallados, mientras que, según se baja, los ítems serán más ambiguos, permitiendo que nuevos requisitos emerjan y puedan subir en prioridad.

Historias de usuario siamesas

Si cada vez que se va a realizar una estimación del *Backlog* o una planificación de un *Sprint* siempre aparece el comentario: “Para poder hacer el elemento A, necesitamos también el elemento B”, tienes un problema de historias de usuario siamesas. Las dependencias en el *Product Backlog* son un problema, ya que imposibilitan tratar las historias de usuario de manera autónoma. Este condicionante afectará a la priorización, ya que, por obligación, y no por necesidades de negocio, un elemento tendrá que estar por encima del otro. También afectará a la hora de definir el trabajo en un *Sprint*, ya que este factor implica decisiones

obligatorias a la hora de asumir una carga de trabajo.

La solución es dividir en partes más pequeñas y hacer una sesión con todo el equipo para analizar las dependencias existentes e intentar aislar el punto común y sacarlo a una nueva historia conjunta. Lógicamente, hay que tener cuidado en este proceso para evitar que una dependencia entre dos elementos se convierta en una dependencia a tres bandas.

Si las dependencias no se pueden reparar, la solución que queda es priorizar ambos elementos, de tal manera que siempre puedan ser llevados a cabo en el mismo *Sprint*.

Catarro mal curado en una historia de usuario

Si aparecen constantemente elementos del *Backlog* que no se cierran *Sprint* tras *Sprint* y siempre aparecen abiertos, se está claramente frente a la dolencia de las tres C, que se ha comentado anteriormente. Se tienen que revisar los principios de Tarjeta, Conversación y Confirmación, porque alguno de los tres debe estar fallando.

El criterio de Tarjeta o *Card*, como se pudo ver en este capítulo, indica que la redacción del elemento de la pila de Producto debe ser lo suficientemente pequeña como para caber en una tarjeta de indexación. Cumplir este criterio puede producir un problema de indefinición. Aplicarlo no significa que solo se cuente con un espacio limitado reduciendo redacción, sino que, si se necesita más espacio para definirlo, es porque hay que dividir el elemento en varios más pequeños.

El criterio de Conversación o *Conversation* indica que tiene que haber existido una conversación en el equipo *Scrum* sobre cada elemento, que el responsable de producto no ha lanzado el requisito “por encima de la valla” y luego se ha desentendido de él. Mediante la conversación y discusión se deberían resolver las ambigüedades del ítem del *Backlog* y debería ser sencillo acometerlo sin incertidumbre.

Finalmente, el criterio de Confirmación o *Confirmation* define que los elementos del *Backlog* pueden ser claramente validados. Esto implica que existe una definición y criterios por los que se puede saber cuándo un ítem está claramente terminado.

Si para todas las historias de usuario se cumple el criterio de las tres C, no debería existir una razón para que no se dieran por terminadas al final de un *Sprint*.

Infección operativa

Ocurre cuando la pila de producto no define la estrategia del producto, sino que habla de la táctica del producto. Si el *Product Backlog* no habla de qué hay que hacer, sino que define cómo hay que hacerlo, entonces se está ante una de las peores infecciones que puede sufrir nuestro *Product Backlog*: la infección operativa. Suele empezar de forma inofensiva, sacando la especificación de un par de ítems de la lista para que los validen ciertos roles en una

organización. Cuando esa especificación vuelve validada, para ejecutarla, se crean tareas operacionales en el *Backlog*. El equipo siente una cierta comodidad con el proceso y se empieza a repetir periódicamente dejando el *Backlog* como un campo yermo de requisitos y plagado de tareas operacionales. Se ha creado un *Sprint Backlog* para todo el producto o proyecto y se rompe el principio DEEP para el *Backlog*.

La solución a este problema reside en llevar un buen control sobre las tareas operativas e intentar dejarlas siempre en el ámbito de los *Sprint*s, dejando el *Product Backlog* libre de estas tareas operativas.

Plaga de requisitos

Si el *Product Backlog* empieza a llenarse de forma muy rápida de requisitos y no se puede manejar de una manera cómoda, puede ser un síntoma de un problema.

Nota:

El concepto cómodo es muy relativo, pero, si ha tenido que empapelar su entorno de trabajo con elementos de su Backlog, quizás no lo esté manejando de manera cómoda.

La plaga de requisitos implica que el *Backlog* está siendo usado no como una base de datos coherente de requisitos de un producto o proyecto, sino como una lista de peticiones. En la pila no se tendrá un plan o un *roadmap*, se está usando este artefacto de *Scrum* como la carta a los Reyes Magos, donde volcar cualquier cosa que se les ocurre a las personas implicadas en el proceso. Es importante recoger siempre los comentarios en todas las reuniones que la metodología define, pero también es importante hacer algo con ese *feedback* y que no se enquiste en la pila de producto de forma indefinida.

Una posible solución es la combinación de las técnicas de priorización por orden con la técnica MoSCoW, vista en este capítulo. Esta técnica permite identificar posibles *Won't* o elementos que no se van a implementar y que deberían separarse del *Backlog* para su correcto saneamiento.

Consejo:

Cree una lista de lluvia de ideas independiente para que todo el mundo pueda participar en la propuesta de ideas para el proyecto. Posteriormente, esa lista puede ser ordenada, segmentada y trabajada para que pueda ser progresiva y coherentemente volcada en el Backlog del producto.

Carencia de las vitaminas de mis requisitos no funcionales

Si con todos los elementos que hay en la pila de producto, los incrementos de producto, aunque cumplen lo que se espera de ellos a nivel funcional, no terminan de convencer, tal vez sea porque el *Product Backlog* no esté completamente definido. La usabilidad, seguridad o el rendimiento son valores que se relacionan con cualidades del producto que van más allá de la funcionalidad y fortalecen al producto o al proyecto. Su falta de definición produce debilidad en el producto o proyecto y futuros problemas.

La solución es prestar tanta atención a las historias de usuario como a los requisitos no funcionales y priorizarlos de forma coherente a las necesidades de los incrementos del producto en cada momento de su evolución.

Nota:

Para definir correctamente valores tan generales como la usabilidad de un producto es necesario ser muy cuidadoso con la última C del criterio de las tres C: la confirmación. Deben definirse criterios de aceptación ejecutables que permitan validar de forma concreta que se cumple el requisito no funcional.

Inmunización tardía

Si su pila de producto no se trabaja para inmunizarse en sus primeros momentos de vida, cuando crezca padecerá grandes enfermedades. Inmunizarse en este contexto significa que hay que priorizar todos los elementos del *Backlog* sobre los que se tengan incertidumbres para trabajar sobre ellos. Priorizando los elementos o usando *Spikes*, como se vio en el capítulo 3, se limitan los problemas que puedan derivarse sobre ellos. Si estas historias de usuario no se afrontan al inicio de un proyecto o producto, podrían aparecer a mitad de este y plantear un problema que incluso podría afectar a la viabilidad del mismo.

La solución es realizar un taller de incertidumbre con el equipo para analizar las incertidumbres del *Backlog*, priorizando las incertidumbres y creando los *Spikes* necesarios para resolver esas incertidumbres.

Nota:

La muerte prematura de un producto o proyecto por la detección temprana de un problema en el Backlog no es una derrota, sino que es un caso de éxito, ya que permite mover al equipo a otro producto viable.

Muerte dulce por contrato

Se ha dejado para el final la peor de las enfermedades que puede sufrir su *Backlog*: la muerte por contrato. Su descripción es sencilla: ocurre si su *Backlog* no es una herramienta de trabajo dinámica del grupo, sino que funciona como un contrato. Si el *Backlog* se establece con el cliente y las partes implicadas se aferran a él para exigir y reclamar, se marchitará y languidecerá sin poder dar sus frutos. Nunca permita que el *Backlog* sea un acuerdo formal o contractual de requisitos. No está pensado para esto y su uso indebido hará que no funcione la metodología en su conjunto.

Un Backlog para gobernarlos a todos, un Backlog para encontrarlos, un Backlog para atraerlos a todos y atarlos en las tinieblas

Cuando un proyecto o producto es muy grande, posiblemente sea necesario que varios equipos trabajen en paralelo y el PB tendrá que estar preparado para esta situación. No se deben tener *Product Backlogs* separados. Siempre que sea posible, se debería mantener un único *Product Backlog* del que se pueda sacar partes de él para alimentar a los distintos equipos durante una iteración.

Cada *Product Backlog* extraído del PB maestro puede tener su propio *Product Owner* que lo gestione, pero siempre estará supeditado al *Backlog* maestro y a un PO que jerárquicamente organice el trabajo de los otros *Product Owners (*ver el apartado dedicado a *Scrum of Scrum*s en el capítulo 12*).*

Las prioridades deben gestionarse desde el punto de vista del *Product Backlog* maestro y se trasladará a *los Product Backlog* relativos. En términos de prioridad, cada equipo no puede trabajar con su criterio, ya que se podría romper la priorización del trabajo entre todos los equipos y aparecer problemas de bloqueos por dependencias.

En resumen

En este capítulo, se ha aprendido cómo debe ser un *Product Backlog* y los elementos que lo componen. Los conceptos DEEP e INVEST deben ayudar a mantener un *Product Backlog* saneado, actualizado y en buena forma. Esta será la única forma en la que su utilidad revierta en un correcto funcionamiento del proceso y el aporte de un valor constante al producto o proyecto que se esté desarrollando.

El trabajo de escribir el libro que tiene en sus manos fue avanzando con la construcción y poblado inicial de un *Product Backlog* que describiera a grandes rasgos los principales requisitos del trabajo. Aquellas partes del proyecto con mayor prioridad que debían iniciarse

antes contaban con mayor detalle. Por ejemplo, escribir un índice detallado; elaborar una serie de apartados especiales (índice alfabético, glosario...) que se fueran poblando a medida que se iba creando el texto; identificar fuentes de imágenes; o acordar convenciones de vocabulario.

Nuestro *Product Backlog* no es completo, y seguramente no sea estable, pero no importa. La naturaleza de *Scrum* nos permite hacer cambios e introducir modificaciones considerando la construcción de nuestro producto, el libro, como una acción incremental.

Ya estamos listos para los próximos pasos y puede arrancar la primera iteración o *Sprint* de nuestro trabajo.

---

22 http://www.infoq.com/presentations/prioritizing-your-product-backlogmike.

cohn

23 La muestra de usuarios para las encuestas no tiene por qué ser muy grande, con 10 repuestas se puede trabajar cómodamente.

5 Antes de empezar: Sprint planning

En este capítulo aprenderá a:

- Preparar el *Sprint Backlog*.
- Estimar historias para seleccionarlas.
- Hacer la planificación detallada en tareas.
- Reconocer el papel del *Scrum Master*.
- Mantener el *Product Backlog*.

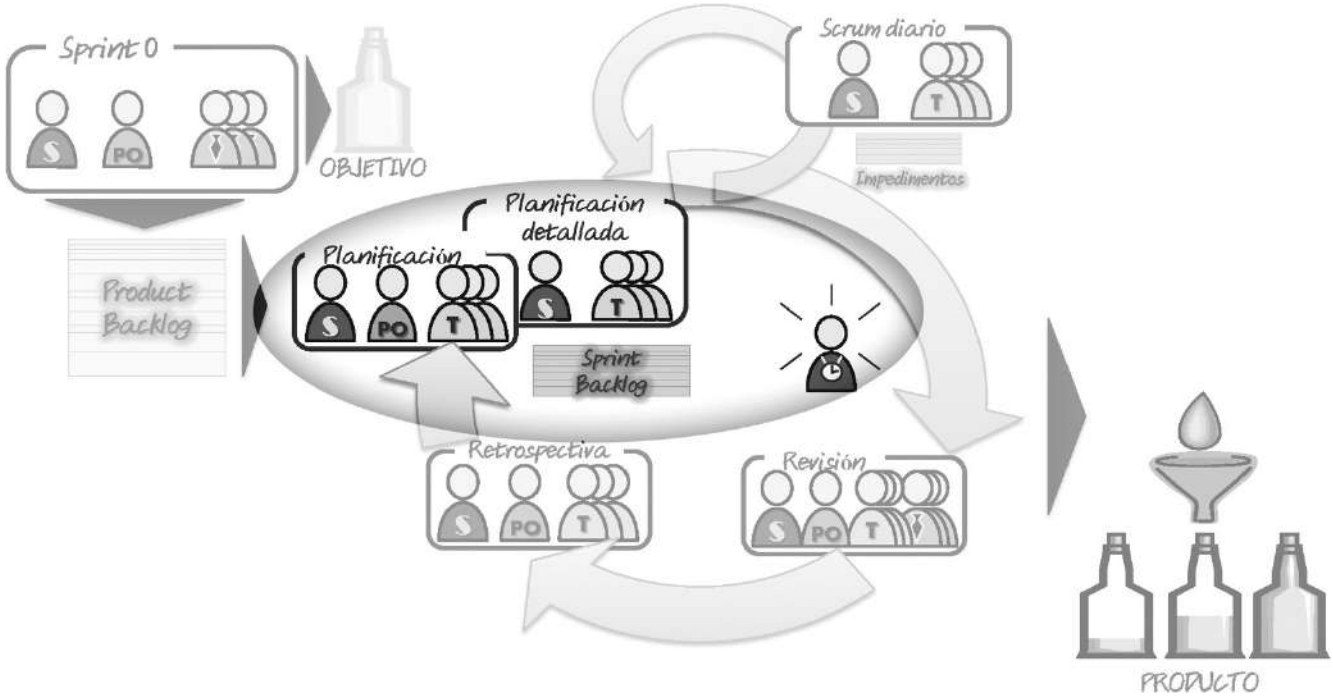


Figura 5.1. La planificación en el ciclo Scrum.

Introducción

Ya hemos fijado el alcance, el contenido y la forma de trabajo de nuestro proyecto, en este caso la escritura de este libro. Hemos dedicado un buen tiempo en el *Sprint* 0 a la organización del trabajo, a seleccionar las herramientas y a establecer las reglas. Hemos trabajado en un *Product Backlog* que recoge los trabajos que hemos identificado hasta ahora, aunque sea a grandes rasgos. Ya tenemos los elementos necesarios para poder empezar a trabajar, pero esto no se puede hacer a la ligera.

Scrum y las metodologías ágiles, aunque eviten formalismos y burocracia, no son una forma de trabajo sin control. Por eso se organiza la actividad del proyecto en una serie de iteraciones o *Sprints* cuya duración se fija inicialmente en el *Sprint* 0. Al principio de cada iteración hay que discutir su contenido, su alcance y cómo verificar que se han llegado a los objetivos planteados.

Para la primera iteración del proyecto de escribir un libro sobre métodos ágiles, tenemos que determinar qué tareas vamos a abordar y por parte de quién, y qué resultados tenemos que ofrecer para considerar que se han alcanzado esos objetivos.

Hemos dedicado nuestro *Sprint* 0 a identificar todas las principales tareas que hay que realizar.

Sin embargo, sabemos que es muy posible que se nos haya pasado alguna por alto o que

puedan aparecer otras nuevas a medida que avancemos. No importa, uno de los beneficios del uso de *Scrum* es la facilidad para incorporar nuevos requisitos o tratar con cambios.

Así que ha llegado el momento de iniciar nuestro primer *Sprint* o iteración del ciclo de trabajo. Tendremos que identificar el objetivo principal para el *Sprint* y las tareas concretas que vamos a desarrollar en él. ¿Cómo lo haremos?

En este capítulo, se va a ver el proceso de planificación o **Planning** del *Sprint*. Se parte de una revisión y priorización de los elementos del *Product Backlog* (historias de usuario, épicas, requisitos del proyecto), a las que se añade criterios de aceptación para determinar cuándo se han cumplido. Esta tarea de revisión y priorización la lidera el **Product Owner**, que puede contar con la ayuda del **Scrum Master**, que entre otras muchas cosas (como se verá más adelante) es el facilitador del trabajo, vigilante del cumplimento de los principios de *Scrum* y puede actuar como intermediario entre PO y equipo. El PO puede apoyarse en otras personas en esta tarea, por ejemplo, en miembros del equipo destacados por su conocimiento o experiencia.

A continuación, ya como parte de la planificación del *Sprint*, se realiza una reunión denominada *Sprint Planning*. En ella, el equipo revisa, junto con el PO, las tareas del *Product Backlog*. Se valora la complejidad de cada tarea y se selecciona, siguiendo la prioridad, las que podrán realizarse en el transcurso del *Sprint*.

En una segunda etapa de la planificación, el equipo traduce las historias a lenguaje de proyecto y las subdivide en unidades menores o tareas. Con todo ello se construye el **Sprint Backlog** o repositorio de los trabajos que se realizarán durante la iteración.

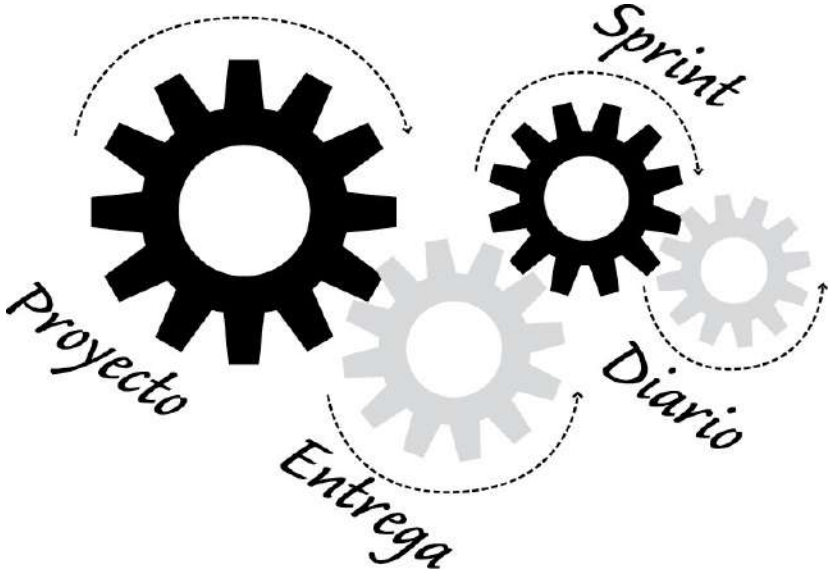


Figura 5.2. Un proyecto puede tener una o varias entregas, organizadas en Sprints, que se realizan en un ciclo diario.

En este capítulo se va a dedicar también un espacio a hablar del **Scrum Master**, uno de los roles principales en el proceso *Scrum*, que tiene una influencia decisiva en el éxito del

trabajo.

Sprint planning: qué es, para qué sirve

El trabajo inicialmente definido en el *Sprint* 0 del proyecto se divide en *Sprints* o iteraciones, que a su vez se pueden agrupar en una o varias *Releases* o entregas (al menos habrá una final) de acuerdo con la longitud del proyecto, las necesidades del cliente y la naturaleza del trabajo.

Esta división del trabajo es muy útil por varias razones: permite abordar tareas más reducidas y controlables, reduciendo la incertidumbre; facilita la obtención de resultados intermedios, lo que va a servir para detectar desviaciones o malas interpretaciones de los requisitos mucho antes; y permite definir una velocidad constante de avance del proyecto, siendo predecible.

Cada una de las iteraciones o *Sprints* se divide fundamentalmente en tres etapas: planificación del contenido, desarrollo del trabajo y revisión de resultados. Este capítulo se centrará en la primera de esas etapas, en la planificación de la iteración o *Sprint Planning*.

Todo *Sprint* se comporta como un pequeño proyecto en sí mismo y, por ello, debe tener un alcance y objetivo claros, más allá de cumplir unas determinadas horas de trabajo.

Una de las primeras tareas de la planificación es precisamente identificar ese objetivo, algo que deberá hacer el *Product Owner*, para lo que puede contar con la ayuda del *Scrum Master*. Podría ser “desarrollar la interfaz de entrada de datos de una aplicación”, “definir el método de liquidación de una cuenta corriente”, “tener el diseño de toda la instalación eléctrica de un edificio” o “preparar la cartelería de una campaña de publicidad”.

También es habitual definir ese objetivo principal una vez se conocen los requisitos o las historias de usuario que se van a abordar, en función de su contenido y alcance.

No siempre va a ser posible tener un objetivo cerrado que se ajuste a las dimensiones del *Sprint*, pero siempre será posible contar con una definición simple de a qué se dedica, por ejemplo: “fue el segundo *Sprint* que dedicamos a desarrollar los componentes software de la base de datos” o “el *Sprint* en el que se definió el índice y se escribió la introducción del libro”.

Nota:

Por supuesto que este objetivo debe consensuarse finalmente con el equipo. Scrum no funciona por medio de la imposición. El PO tiene la potestad de modular el trabajo, pero es el conjunto de los involucrados (SM y, sobre todo, equipo) quien finalmente define la actividad.

Otra tarea muy importante previa al proceso de *Planning* es la priorización de las tareas del *Backlog*. El repositorio o *Backlog* del proyecto contiene la lista de trabajos definidos usando el lenguaje de negocio, que suelen expresarse usando el formato de historias de usuario (*User Stories*). El *Backlog* del proyecto es un elemento dinámico y cambiante: pueden aparecer nuevas historias que recojan cambios o novedades en los requisitos del trabajo, de la misma forma que pueden desaparecer.

Una de las dimensiones más importantes del *Product Backlog* es la prioridad. Aunque todas las historias de usuario son importantes para alcanzar el objetivo último del trabajo, se deben ordenar de más a menos prioritarias, ya que no es posible abordarlas todas simultáneamente. Esta priorización define el orden en el que se hará el trabajo en el proyecto y, con él, el contenido aproximado de cada *Sprint*.

El gestor del *Product Backlog* es el *Product Owner*, así que él es quien se encarga de que esté:

- **Actualizado:** Eliminando los elementos que se hayan considerado irrelevantes con el tiempo y añadiendo los nuevos que se vayan identificando.
- **Completo:** Cada una de las historias debe contener información suficiente como para comprender qué se espera de ellas y, así, poder definir las acciones concretas (tareas) necesarias para realizarlas. Esta información debe incluir necesariamente criterios de aceptación claros para determinar si el trabajo ha concluido o no.
- **Ordenado:** De acuerdo con el orden de prioridad que le asigna el *Product Owner*.

Por supuesto, hay otras características que debe cumplir un buen *Product Backlog*, expresados por la regla “DEEP”, explicada en el capítulo anterior. Contiene requisitos funcionales (del producto o de la actividad) y no funcionales (necesarios para el trabajo en sí mismo y que definen sus características).

Así que el punto de partida antes de iniciar la planificación del *Sprint* es cuando el PO o *Product Owner*, que puede contar con *Scrum Master*, revisa el *Product Backlog* para ordenar sus elementos de acuerdo con su criterio de priorización. Esa ordenación determina qué trabajos deberían abordarse en el transcurso del *Sprint*, por lo que debe hacerse cuidadosamente y considerando la relación entre las distintas historias. Si se quiere que el *Sprint* se dedique sobre todo a, por ejemplo, definir la ventilación de un edificio de oficinas en un proyecto de arquitectura, lo normal es que las tareas relacionadas con esos aspectos tengan mayor prioridad.

Figura 5.3. La planificación en Scrum es una actividad colaborativa.

Nota:

Una de las responsabilidades más importantes del Product Owner es poblar y priorizar el Backlog. Sin embargo, una queja habitual de muchos equipos es el encontrarse con un Product Owner que nominalmente acepta la aplicación de Scrum, pero delega estas tareas en el Scrum Master. Los miembros del equipo y el propio SM no deben consentir esa situación: hace que el PO se desligue de los principios de Scrum al verlo como algo ajeno, que la aplicación de estas prácticas sea puramente cosmética y que al final se pierdan los beneficios que ofrece esta forma de trabajo.
Si es usted un Product Owner no debe delegar nunca la gestión del Product Backlog. Es aquí donde reside el control del trabajo, no en la jerarquía o la gestión de las reuniones. El rumbo del trabajo diario y del conjunto del proyecto se marca desde el Product Backlog.

Esta priorización se puede hacer de varias formas. La más cómoda es la ordenación: poniendo las historias más prioritarias antes o por encima de las demás. A fin de cuentas, un *Backlog* no deja de ser una pila donde todos sus elementos están dispuestos uno sobre otro y se accede en primer lugar al que está por encima de todos los demás.

Otra forma de priorizar es asignar un código o valor de priorización: puede haber historias “A”, “B” o “C”, o con prioridad 100, 50, 20...

También se puede aplicar la técnica conocida como MoSCoW, descrita en el capítulo

anterior.

Cualquier método vale, pero el primero posiblemente sea más simple y fácil de entender.

Nota:

Si la priorización es numérica, no hay que olvidar usar números muy separados. En cualquier momento puede aparecer un nuevo elemento que haya que colocar entre ambos y habrá que asignarle algún valor. Aunque lo habitual es que sea el Product Owner quien idee y genere las historias, hay ocasiones en las que otras personas pueden añadirlas. En este caso, el Product Owner debe estar al tanto y comprender perfectamente el contenido de la historia, ya que es él quien debe explicarla al equipo.

Con el *Product Backlog* priorizado, podemos continuar adelante. El punto de partida para iniciar un *Sprint* es la planificación. Esa planificación se hace en dos etapas: una de selección de historias, y otra de subdivisión en unidades más pequeñas o tareas. La primera requiere el concurso del PO, mientras que la segunda es una actividad más propia del equipo. En ambas puede participar el *Scrum Master*.

Véase ahora cómo se hace esa primera fase de selección de historias para poblar el *Sprint Backlog*.

Sprint Backlog

El objetivo del proceso de *Sprint Planning* es rellenar el **Sprint Backlog** o pila de **sprint**. Se trata de un repositorio que recoge los trabajos que van a realizarse en una iteración o *Sprint* determinado. Es decir, que cada *Sprint* tiene un *Sprint Backlog* distinto. Este repositorio contiene las historias de usuario y, sobre todo, las tareas, que el equipo, que es quien gestiona este *Backlog*, ha identificado en el momento de la planificación de detalle (una de las dos reuniones dentro del proceso de planificación del *Sprint*).

El *Sprint Backlog* es propiedad del equipo, que es quien lo gestiona y actualiza, mientras que el *Product Backlog* está asignado al *Product Owner*. Para poblar el *Sprint Backlog* se parte del *Product Backlog*, seleccionando historias en función de la priorización hecha por el *Product Owner*. El equipo va estimando en orden cada historia, que va añadiendo al *Backlog* hasta que se alcanza una suma de *Story Points* o puntos de historia (ver más adelante en este capítulo) en las historias ligeramente superior a la velocidad habitual del equipo (más adelante se hablará de este concepto de velocidad). Las historias demasiado “grandes”, es decir, que tienen una complejidad tan alta que impide realizarlas en el tiempo fijado para *Sprint*, se subdividen hasta convertirse en unidades más manejables. La siguiente figura describe estas relaciones:

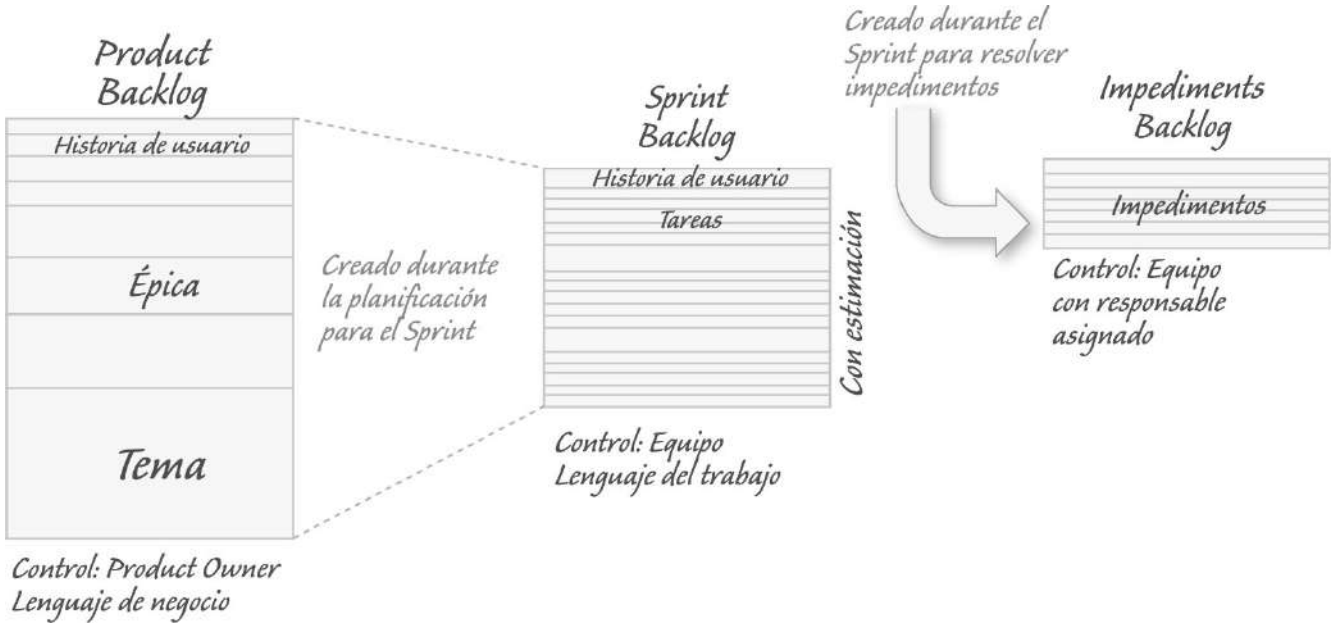


Figura 5.4. Los distintos repositorios o backlogs de Scrum.

Cada historia seleccionada se divide, a su vez, en tareas. Estas tareas tienen que estar descritas en el lenguaje del dominio técnico del trabajo, ser pequeñas, detalladas y también pueden contar con una estimación del tiempo necesario para su realización.

Aunque hay equipos que prefieren usar medidas relativas para comparar las tareas, es bastante habitual que se tome como unidad de referencia el tiempo, ya que las tareas concretas se pueden estimar con más exactitud. También la descripción es más concisa, desglosando las acciones que deben completarse en cada tarea, y no en forma de requisitos, como ocurre con las historias de usuario.

Nota:

Por regla general, se asume que cada tarea debe poder ser realizada por una persona y su duración debería ser entre medio y tres días.

Si lo comparamos con el *Product Backlog*, el contenido del *Sprint Backlog* es mucho más rico y detallado. Para empezar, cada historia seleccionada para el *Sprint* debe contener un claro y preciso criterio de aceptación del resultado que se espera, que será introducido por el PO. Este criterio de aceptación es uno de los elementos fundamentales de la operativa del *Sprint*. Sirve de guía a los miembros del equipo a la hora de desarrollar el trabajo y alcanzar los resultados (sobre todo si el *Product Owner* no está accesible) y permite validar si se han alcanzado o no los objetivos del trabajo.

En paralelo, cada tarea va a tener una “definición de hecho” o *Definition of done* que introduce el equipo para saber cuándo ha alcanzado el resultado esperado para esa tarea concreta.

Nota:

Lo ideal es que el criterio de aceptación esté de antemano en todas las historias del Product Backlog, pero por distintas razones no siempre es posible. Lo que sí que es obligado e inexcusable es que cada historia del Sprint Backlog tenga un criterio lo más detallado y completo posible, y las tareas su correspondiente definición de hecho. Es lo que permite saber si ha conseguido su objetivo o no.

Las historias del *Sprint Backlog* también tienen un valor de complejidad que dé una idea del esfuerzo requerido para su desarrollo. Son los llamados *Story Points*, que ya hemos mencionado y sobre los que volveremos más adelante para ver cómo se obtienen.

Las historias deben ser independientes (aunque puedan tener relaciones con otras) y pueden tener un historial de su evolución y, sobre todo, información y anotaciones que documenten su resolución. Cada miembro del equipo que trabaje en una historia puede ir añadiendo información relativa a su desarrollo y, si la herramienta usada lo permite (véase en el siguiente capítulo el apartado correspondiente a herramientas), incluir todo tipo de documentos, diagramas, planos, código o fotografías. Toda esta información permite seguir el trabajo mientras se desarrolla y documentarlo a su finalización, de forma que sea fácil entender por otras personas qué se hizo, por qué y cómo.

Nota:

En entornos de desarrollo software, por ejemplo, el llamado "diseño detallado" puede verse sustituido por la información introducida para documentar cada historia de usuario del Sprint Backlog.

Figura 5.5. Un posible tablero o panel con columnas para los estados de las tareas: Task (pendientes de iniciar); In progress (realizándose); Stopped (impedidas) y Completed (terminadas).

El *Sprint Backlog* tiene tres dimensiones principales.

En primer lugar, está la prioridad, que refleja la del *Product Backlog*. Las historias seleccionadas para el *Sprint* durante el proceso de planificación (como veremos más adelante) se extraen del *Product Backlog* de acuerdo con el orden de prioridad fijado por el PO. Ese orden se mantiene en el *Sprint Backlog* de forma que se deben abordar, en primer lugar, las historias (y sus tareas) más prioritarias y seguir esa secuencia a medida que avanza el trabajo.

Otra dimensión (implícita) es la de detalle, de forma que dentro de cada historia de usuario tendremos el desglose de las tareas concretas que la resuelven. De hecho, el trabajo del equipo se refiere a las tareas que componen una historia. No se puede dar por resuelta una historia de usuario si no se han completado todas las tareas en las que se descompone. Esto es muy importante. La última dimensión también importante y se refiere al estado. Una historia (y sus tareas) pueden estar: pendientes de iniciar, en curso, terminadas e impedidas, además de otras posibilidades que se comentarán ahora. Véase su significado:

- Por definición, antes de empezar cualquier historia o tarea, estará en estado **pendiente**, es decir, que nadie la ha seleccionado aún del *Sprint Backlog* para resolverla. Las historias pendientes se ordenan por prioridad, de forma que la primera que se debe acometer es la más prioritaria de las pendientes, salvo que haya alguna razón de peso o impedimento que obligue a tomar otra menos prioritaria.
- Cuando un miembro del equipo de trabajo ha seleccionado una tarea para completarla, la tarea pasa a estar **en curso**, en desarrollo o cualquier otra denominación que implique que se está trabajando sobre ella. Cuando se aborda la primera tarea de una historia, se asume que la historia está también en curso, y no solo esa tarea concreta. Las tareas en curso están asignadas siempre a una persona concreta del equipo que es quien se responsabiliza de su cumplimiento, aunque en determinados casos sea preciso involucrar a otros miembros.
- Si la tarea se considera realizada de acuerdo con la definición de hecho (*Deffinition of Done*), pasa al estado de **terminada**. En cuanto a las historias de usuario, no pueden considerarse terminadas si no lo están todas las tareas que las componen. Por supuesto que esa conclusión es desde la perspectiva del miembro del equipo y de acuerdo con los criterios de aceptación que se haya introducido. Más adelante, en la reunión de *Review*, el PO validará el resultado y determinará si la historia está o no completada.
- Una tarea (y con ella su historia) puede estar **impedida**, lo que significa que hay algún elemento que no permite desarrollar el trabajo sobre ella. La naturaleza de ese impedimento puede ser de lo más variada: falta algún elemento externo, como documentos, diseños, o software; herramientas no disponibles; o imposibilidad de contactar o ausencia de una persona relevante para la tarea. En realidad, lo más importante es documentar el impedimento, identificar la manera de resolverlo y asignar un responsable (generalmente el SM) para resolverlo y continuar.

Además, puede haber otros estados en función de la naturaleza del proyecto. Por ejemplo, si se está desarrollando un producto tecnológico, como puede ser una aplicación software, puede incluirse un estado adicional relacionado con QA o el aseguramiento de calidad. En ese caso, este estado indicaría que, además de concluido, el resultado de la tarea o la propia historia de usuario ha sido validado por el área de calidad. Se podrían definir otros estados adicionales en función de la naturaleza del trabajo y las necesidades del proyecto, pero estos que se han comentado son los básicos.

El tablero de tareas

Aunque más adelante, en el capítulo dedicado a herramientas se hablará extensamente de las herramientas usadas en *Scrum*, a continuación se menciona brevemente una de ellas porque ayuda a comprender muy bien el concepto del *Sprint Backlog*.

Aunque hay muchas aplicaciones y medios informáticos para gestionar el proceso *Scrum*, una de las más simples es tan bien una de las más útiles. Se trata del panel, tablero de tareas o *task board*, un sistema extremadamente sencillo de mostrar toda la información de un vistazo.

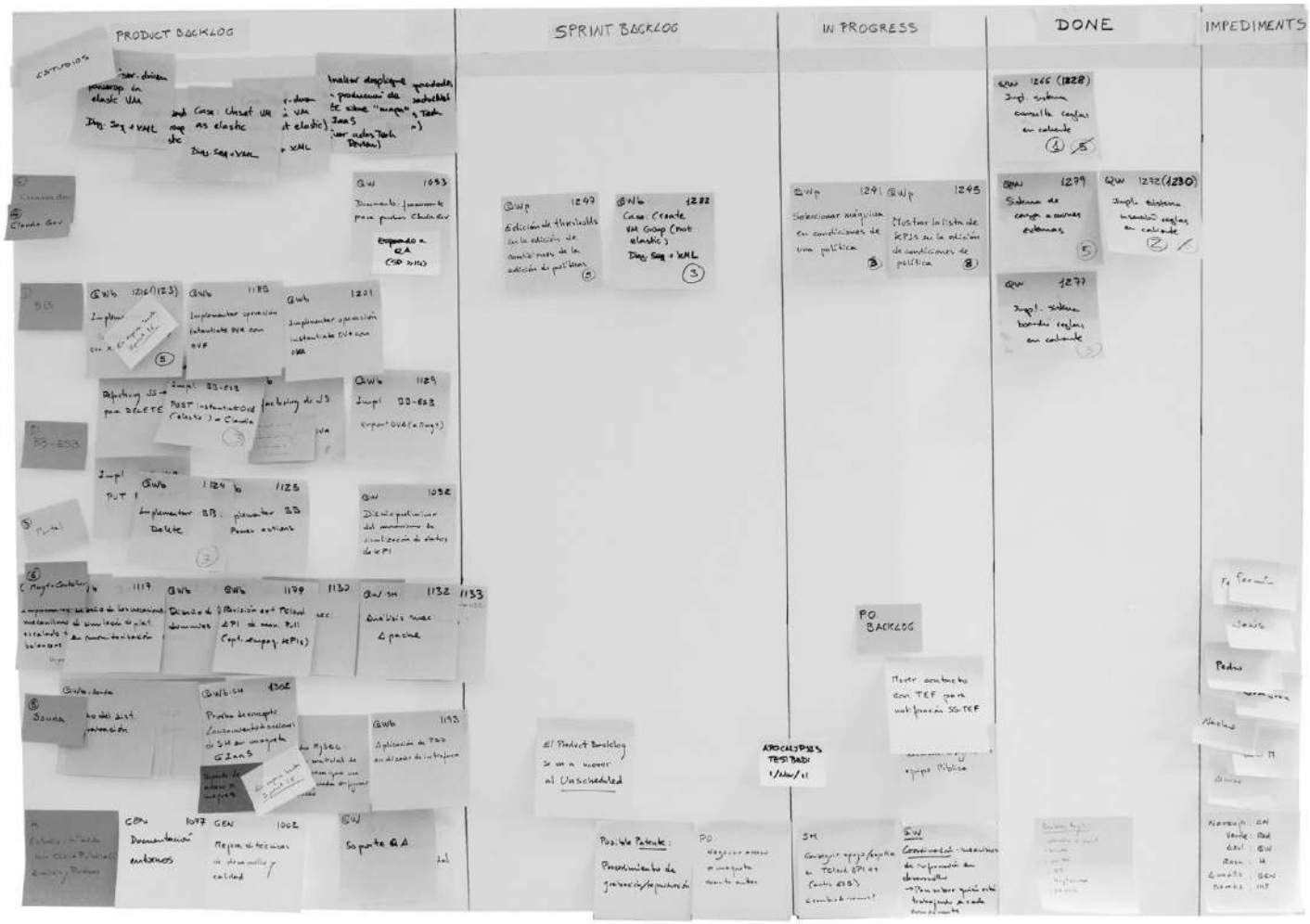


Figura 5.6. Tablero usado en un proyecto real por los autores.

Tableros como el anterior, puestos en un lugar visible para todo el equipo, permiten tener en todo momento un estado actualizado de la evolución del *Sprint*. Se compone de etiquetas autoadhesivas, carteles, textos escritos, etc., que contienen las historias de usuario y, si el formato lo permite, las tareas en las que se dividen. Estos elementos se disponen en columnas que representan cada uno de los estados (pendiente, en curso...).

Uno de los elementos clave de *Scrum* es facilitar a todo el equipo un modo de conocer en todo momento y rápidamente el estado del proyecto. Aunque las herramientas informáticas son muy útiles, nada sustituye a un tablero puesto en un lugar bien visible, donde es muy fácil cambiar el estado de las tareas con solo mover una etiqueta de sitio.

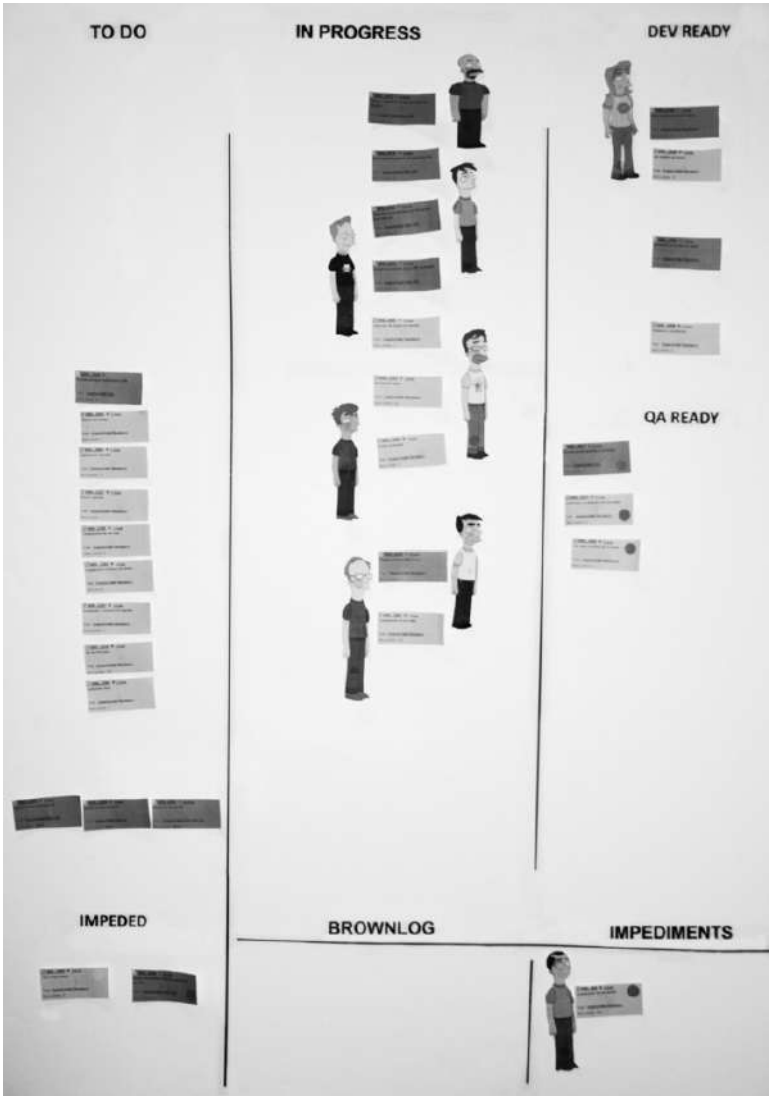


Figura 5.7. Otro ejemplo de tablero real.

Truco:

La información que se puede mostrar va a depender mucho del medio. En un post-it a duras penas nos cabrá algo más que el título, alguna indicación de componente y, por ejemplo, la medida de complejidad. Una forma fácil de añadir información es jugar con los colores de las notas, con el color del texto, usar símbolos o alguna otra convención (mayúsculas y minúsculas, la disposición horizontal o vertical para expresar la prioridad).

Cómo obtener el Sprint Backlog: selección de historias

Ahora que se tiene un objetivo general para el *Sprint* y que se ha realizado la ordenación de todas las historias pendientes, ha llegado el momento de definir el trabajo del *Sprint*, incluido en el repositorio de la iteración o *Sprint Backlog*. Como ya se ha comentado, al igual que el *Product Backlog*, el *Sprint Backlog* es una sucesión de historias de usuario contenidas en alguna de las herramientas de *Scrum*. Ese es el guion del trabajo que se va a realizar y cada uno de sus elementos, expresados en lenguaje de negocio, van a desgranarse en otros más pequeños, las tareas, descritas con el lenguaje del dominio de aplicación del trabajo (el marketing, la arquitectura, el diseño industrial o el desarrollo software).

Poblar el *Sprint Backlog* es el trabajo fundamental del proceso de planificación del *Sprint* y ese trabajo se desarrolla en un par de reuniones que inauguran cada iteración.

Nota:

Dentro del ciclo de trabajo marcado por cada Sprint, hay una sucesión de reuniones que encadenan el final de una iteración y el comienzo de la siguiente. Tras la Review o revisión y la Retrospectiva, que marcan la conclusión de un Sprint, las reuniones de planificación o Planning a las que nos referiremos en este capítulo abren el inicio de una nueva iteración.

La reunión de planificación o *Sprint Planning* se divide, en realidad, en dos reuniones. La primera hace una primera selección del trabajo que se va a abordar, mientras que la segunda entra en el detalle de las tareas concretas que se van a realizar. Estas reuniones están habitualmente, aunque no exclusivamente, coordinadas por el *Scrum Master*.

La primera reunión de planificación o *Sprint Planning* busca identificar el subconjunto de historias que se va a abordar en un *Sprint*. Como en todas las reuniones de *Scrum* es muy necesario:

- Contar con la debida preparación. Las reuniones se preparan en contenido, entorno y herramientas. En este caso, será el *Scrum Master* por regla general quien la convoque con suficiente antelación. Implicará a todos los asistentes: equipo, *Product Owner* y otros representantes del cliente, del área calidad si fuera necesario, *coach* si lo hubiera,

incluso otras personas potencialmente interesadas. También se encargará de facilitar la disponibilidad de las herramientas necesarias.

- Garantizar la presencia del *Product Owner*. Es crucial en la reunión de planificación como veremos más adelante.
- Establecer unos límites temporales determinados, que se deberán cumplir estrictamente (*time-boxing*).
- Garantizar que el objetivo, mecánica y resultados esperados de la reunión son conocidos de antemano.

Truco:

Aunque el equipo de trabajo tenga una amplia experiencia y conozca la teoría de Scrum, no está de más hacer un recordatorio periódico de los principios y mecánica. Con el tiempo, los equipos adquieren hábitos y costumbres que relajan los principios de Scrum, incluso van contra ellos. El Scrum Master puede dedicar unos minutos en cada Sprint a repasar algunos de los aspectos clave de Scrum. Arrancar una reunión recordando su cometido y objetivos es una buena forma de ayudar a alcanzarlos.

Véase ahora cómo es el proceso de llenado del *Sprint Backlog* o repositorio de la iteración. Es muy importante no perder de vista que el objetivo principal es hacer una selección de las historias de usuario que se van a completar a lo largo del *Sprint*. Eso es lo que hay que conseguir al final del proceso y todo lo demás (estimación, cálculo de velocidad, reuniones) no son más que herramientas para alcanzar ese objetivo.

La reunión de planificación, como todos los grandes hitos de *Scrum*, cuenta con el *Scrum Master* como facilitador, aunque un equipo muy maduro puede ser capaz de avanzar sin requerir, o solo mínimamente, esa figura. El SM se encarga de garantizar que la reunión no se desvíe de su propósito, que no se superen los límites temporales (*Time Boxing*) y que acudan todas las personas relevantes. Merece la pena dedicar un momento a hablar de ellas.

Es imprescindible que al *Sprint Planning* acuda tanto el *Product Owner* como el equipo de trabajo. El primero es quien va a dar la dimensión del alcance de cada historia de usuario, y el segundo es quien debe entenderla para poder abordar su realización. ¡No es posible lanzar el *Sprint Planning* sin el *Product Owner*! Es algo que debe quedar muy claro. Si por algún motivo fuera imposible su presencia, debe proveer un sustituto capaz de reemplazarle o haber previsto la eventualidad añadiendo todo tipo de detalles al *Product Backlog*, incluso habiendo revisado su contenido, aunque sea parcialmente, al equipo.

Nota:

Dentro de una organización, si el equipo no pone inconveniente, pueden asistir personas de otras áreas a las distintas reuniones. Se trata de facilitar el aprendizaje, de manera que

grupos menos maduros en Scrum o personas que están introduciéndose puedan aprender de la forma de proceder de equipos maduros. En cualquier caso, el carácter privado o público de cada reunión, sea del tipo que sea, lo decide el equipo, con el apoyo del SM.

De la misma forma, en una reunión cuyo propósito es definir el trabajo del equipo y garantizar que este conozca con precisión lo que se espera de cada historia de usuario, es imprescindible la presencia del equipo o del mayor número posible de miembros.

En el caso particular del *Sprint Planning*, todas aquellas personas que puedan hacer contribuciones valiosas y aportar valor están invitadas. En especial, todos los representantes del cliente más allá del *Product Owner*; los llamados *stakeholders*. El propósito es ayudar al equipo a entender todos los matices de los requisitos del trabajo que se quiere realizar en el *Sprint*.

La mecánica de la reunión de planificación es en sí misma bastante simple.

Como ya se ha comentado, se parte de un *Product Backlog* revisado y priorizado. Si ha habido algún cambio sustancial en él, debería empezarse por hablar de esos cambios y su motivo. A continuación, el PO procederá a leer la primera historia de usuario del *Backlog*, la de mayor prioridad. El objetivo es que el equipo entienda los requisitos de la historia, que tengan una idea preliminar de las tareas concretas que deberán realizar y de la complejidad del trabajo con vistas a poder estimar su tamaño y dificultad (si es que no se había estimado previamente).

La lectura debe servir para aclarar todas las dudas que pueda tener el equipo, motivo por el que está presente el *Product Owner*. Además, es muy importante que cada historia tenga unos criterios de aceptación claros y detallados, como ya hemos dicho. Estos criterios de aceptación servirán para contrastar al final del *Sprint* si puede darse o no por completada la historia.

Una vez ha explicado el *Product Owner* qué es lo que espera y se han aclarado todas las dudas, es el momento de la estimación. De acuerdo con su propia experiencia, los miembros del equipo asignarán un valor de complejidad, llamado puntos de historia o *Story Points*, tal y como se verá en el siguiente apartado. Este número es importante: se suma hasta que alcance un valor cercano a la velocidad media del equipo. La velocidad representa la cantidad de trabajo que un equipo es capaz de realizar en un *Sprint* y depende de muchos factores: desde la madurez del equipo a la comprensión del problema, pasando por la capacidad para estimar y medir ese valor.

Cada historia comprendida, valorada y seleccionada pasa a formar parte del *Sprint Backlog*, manteniendo el orden de prioridad establecido originalmente por el PO. Cuando la suma de los puntos de historia de las historias seleccionadas alcance a completar (e incluso superar ligeramente) la velocidad media del equipo, será el momento de concluir la primera etapa de la planificación. Para completar el *Sprint Backlog* todas esas historias deberán aún ser divididas en tareas.

Nota:

Si un equipo ha alcanzado en los últimos Sprints velocidades de, por ejemplo, 28, 33, 36, 39, 32 y 30, podemos afirmar que se mueve en un rango de entre 30 y 36. Cuando se hace la estimación y se valoran las historias, las podemos ir añadiendo hasta que la suma de los puntos de todas ellas esté en el entorno de la velocidad media. En este ejemplo, cuando se alcance un valor de, por ejemplo, 35 puntos, el equipo puede considerar que se puede cerrar el Sprint con esas historias. También puede aplicarse el criterio del "tiempo de ayer": si en el último Sprint la velocidad alcanzada fue de 42, podemos pensar que el próximo puede estar alrededor de este valor si el entorno permanece estable.

Truco:

Es conveniente valorar unas cuantas historias adicionales. Puede ser que el Sprint haya sido especialmente productivo (o la estimación poco acertada), que hayan quedado historias impedidas y se agoten en el Sprint Backlog, por lo que se debe continuar con las historias más prioritarias del Product Backlog. Tenerlas valoradas será de ayuda a la hora de calcular la velocidad final del Sprint.

La estimación de trabajo que puede realizar el equipo es una parte destacada del proceso de planificación, pero la más importante es el compromiso. El equipo puede decir al PO: "Creemos que podemos llegar a alcanzar esta cantidad de trabajo", pero sobre todo debe decir: "Pero nos comprometemos a realizar al menos estas tareas". El compromiso es uno de los principios de *Scrum* y es compartido por todo el equipo como si fuera una única persona.

Cuidado:

La presión de fechas y costes pueden traducirse en un Product Owner que trate de forzar al equipo a adoptar estimaciones más agresivas, comprometerse a una cantidad de historias superior o reducir la calidad para cumplir plazos. Todas estas medidas están llamadas al fracaso: si un equipo no suele realizar más de 30 puntos de historia por Sprint, es difícil que de forma sostenida duplique esa cantidad. Además, es un hecho constatado que la presión se traduce en una **reducción de la productividad** *y que solo funciona en periodos muy cortos de tiempo. El otro aspecto, la calidad,* **debería ser innegociable***: reducir la calidad supone arriesgarse a tener la falsa impresión de completar tareas, que luego habrá que revisar, corregir e incluso rehacer a un coste muy superior al de aplicar la calidad debida desde el principio. No se engañe a sí mismo.*

El trabajo que se compromete a realizar el equipo es la otra cara del objetivo del *Sprint* que haya establecido el PO. Sin embargo, ese compromiso debe ser expresado libremente por

el equipo, sin que deban influir las presiones (prisas, urgencias) del PO.

Antes de hablar de la planificación detallada, vamos a contar cómo se calcula la velocidad.

La velocidad y su estimación

Como ya se ha indicado, en la primera etapa del *Sprint Planning*, y con la participación del *Product Owner*, se revisa cada historia del *Product Backlog* siguiendo el orden de la priorización. El PO la explica, detalla los criterios de aceptación que servirán para validar el trabajo realizado y atiende a las preguntas, dudas y aclaraciones que plantee el equipo.

Como estas historias son distintas en alcance, complejidad y dificultad, es preciso valorarlas para determinar qué cantidad de trabajo se va a realizar finalmente en el *Sprint*. Esa valoración es una estimación que hace el equipo a partir de su experiencia previa, del conocimiento que tiene del trabajo y de su propio olfato profesional. No hay medios objetivos de "medir" historias de usuario, ni ninguna otra herramienta que pueda ayudar más allá del conocimiento y la experiencia de cada miembro del equipo.

Ese valor de complejidad, dificultad o cantidad de trabajo se mide en puntos de historia o *Story Points*. Hay varios puntos de vista sobre su significado, ya que mientras hay equipos que los usan como expresión de complejidad hay muchos autores que los definen como esfuerzo, que es la visión que se mantendrá en este libro. En cualquier caso, es muy importante que todos los miembros del equipo estén de acuerdo en su significado y la forma de medirlos.

Cuando expresan cantidad de trabajo o esfuerzo, puede usarse un valor estimativo que solo sirve como medida de comparación (una historia con 2 puntos requerirá menos esfuerzo que una de 8), pero no indica una cantidad de horas o días de trabajo. Esta medida abstracta permite hacerse una idea del trabajo que requiere una historia frente a otras, aunque no se concrete en una duración determinada. Hay una forma de hacer esta estimación que se basa en tomar como referencia una historia pequeña, en la que haya acuerdo sobre el esfuerzo necesario para hacerla. Esa historia recibe un valor de "1", y las demás se estiman comparándolas con esta.

Otra forma de estimar las historias es usar el número de "días ideales" (*Ideal Days*) o teóricos que requeriría una historia para ser llevada a cabo. Estos días ideales se calculan considerando el tiempo que una sola persona necesitaría para llevar a cabo el trabajo asociado a esa historia sin interrupciones ni distracciones. A lo largo de su jornada de trabajo, las personas atienden a muchas actividades que no están directamente relacionadas con el proyecto en el que trabajan: reuniones internas, formación, acciones de RR. HH., compromisos sociales, otros proyectos... El porcentaje de tiempo que se dedica de manera efectiva al trabajo concreto varía en función de la persona, su rol y la organización en la que trabaja, aunque se suele estimar entre un 50% y un 80%. Hay autores que llaman a ese

porcentaje “factor de foco” o *Focus Factor*.

Nota:

En entornos de desarrollo software donde se utilicen técnicas de programación en parejas (Pair Programming), el día ideal debe corregirse para considerar el tiempo en el que **dos personas** *trabajan juntas en una misma actividad.*

La estimación es una acción colectiva, en la que participa todo el equipo (o al menos las personas directamente implicadas en la historia de usuario). Se puede hacer de muchas formas, pero una técnica muy habitual y vistosa es el llamado *Planning Poker* o *Estimation Poker*, en el que se usan unas cartas marcadas con números para que cada miembro del equipo vote usando su criterio. Luego del consenso y compromiso entre todos se asigna una valoración a cada historia.

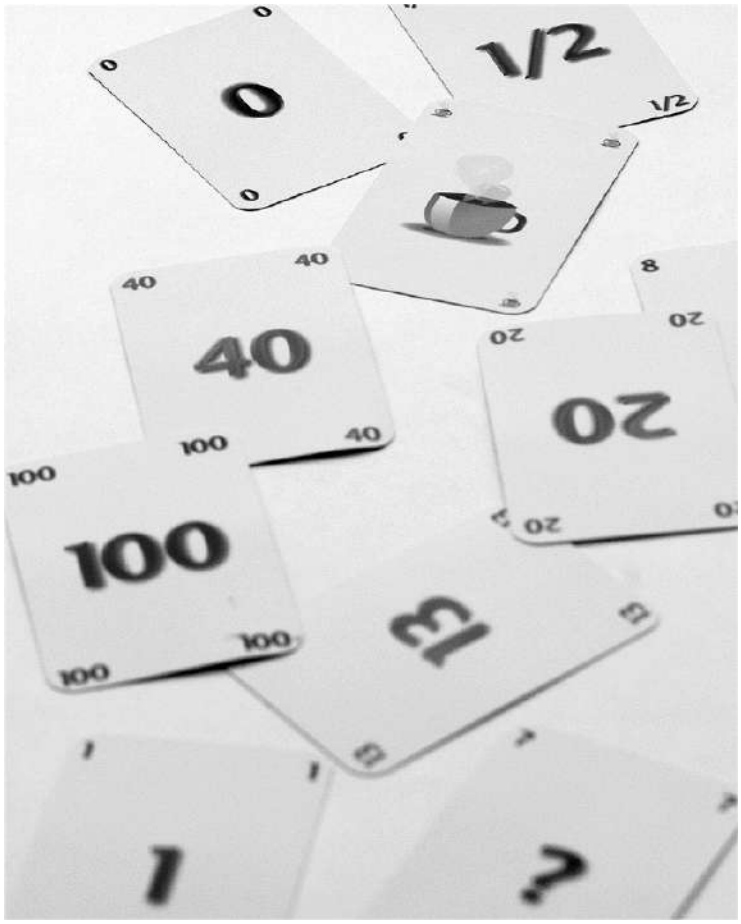


Figura 5.8. Cartas para el Planning poker.

Estas cartas se numeran usando una serie basada en la de Fibonacci: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100. La idea de dejar diferencias cada vez mayores en los últimos números sirve para evitar estimaciones ajustadas cuando la incertidumbre crece. Cuando no es fácil acotar el

esfuerzo necesario para un trabajo, es muy difícil estimar sin incurrir en grandes errores.

La mecánica del juego es simple: tras conocer en qué consiste cada historia y hacer las preguntas que precisen, los miembros del equipo estiman el esfuerzo sacando a la vez una carta con su valoración. Si esta es coincidente, se acepta. Si no lo es, cada persona defiende el valor que ha estimado y se discute entre todos hasta alcanzar un valor de consenso, que se traslada a la historia.

Puede pensarse que solo deberían opinar aquellas personas que puedan tener participación directa sobre el desarrollo de la historia, sin embargo, conseguir que vote la mayoría del equipo puede aportar puntos de vista que no se hubieran tenido en cuenta de otro modo.

El *Planning Poker* no es la única técnica de estimación, pero sí una de las más conocidas y difundidas. Las técnicas alternativas inciden en buscar formas de representar una escala y comparar tamaños. A continuación, se mencionan algunas de ellas:

- Camisetas: Se usa una escala de tallas de camiseta para representar el tamaño aproximado de una historia y para compararlas entre sí: XS, S, M, L, XL, XXL... Se fija un límite para el tamaño máximo que se puede abordar en el *Sprint*, por ejemplo, llegar hasta la talla L. Aquellas que lo excedan deberán subdividirse.
- Perros: Se usa una graduación de razas de perros, por ejemplo, entre chihuahua y gran danés, y se elige el tamaño máximo abordar (por ejemplo, un pastor alemán).
- Coches: O vehículos, con el mismo sistema, entre un mínimo (Smart) y un máximo (limusina).

Como se ve, todas estas alternativas buscan metáforas de tamaño que sean fáciles de entender y manejar. Luego se puede usar una tarjeta o medio gráfico de proponerlas o simplemente enunciarlas.

Sumando todos los puntos de las historias seleccionadas en un *Sprint*, se obtiene la velocidad estimada de la iteración, es decir, la que se alcanzaría si se completaran todas las historias. Sumando todos los puntos de las historias completamente terminadas del *Sprint* se obtiene la velocidad real del *Sprint*.

La velocidad es un concepto de *Scrum* que define la capacidad del equipo para realizar sus actividades. Es una herramienta que ayuda a estimar y medir el proceso, pero debe usarse con todo tipo de precauciones. El motivo es su naturaleza arbitraria y subjetiva: los intentos de medir el proceso de desarrollo de un trabajo, sobre todo de naturaleza intelectual y con un cierto grado de incertidumbre, han fracasado históricamente. Incluso en campos tan controlables como el desarrollo de software, todos los intentos de medida objetiva han fallado. Con todos sus defectos, se verá que la estimación es una herramienta muy útil para hacer más predecible el trabajo de un equipo.

Nota:

Solo se contabilizan los puntos de una historia completa. Si la estimación original era de 8 puntos para una historia que solo se ha completado a medias, no se suman 4 puntos, se suma 0. El motivo no es caprichoso: una historia de usuario representa un requisito completo de los clientes (ponerle frenos al coche, preparar la cartelería de una campaña, añadir la gestión de usuarios de un programa, diseñar la calefacción de un edificio...), por lo que únicamente tiene sentido cuando se verifica completamente, no cuando solo hay ciertas partes realizadas.

Dado que se hace referencia a valores estimados subjetivamente, no tiene sentido tratar de obtener cantidades muy precisas. Hay que saber vivir con la incertidumbre y la idea de que hay un considerable error incluido en estas estimaciones. Sin embargo, es mucho mejor trabajar con estimaciones imprecisas que con ninguna en absoluto.

Figura 5.9. Equipo real jugando al Planning Poker.

El uso de la medida de velocidad debe ser interno y circunscrito al seguimiento del trabajo del equipo y como ayuda para su estimación. Tratar de usarlo como medida objetiva (por ejemplo, para comparar a un equipo con otro) solo dará lugar a que surjan malas prácticas, como podría ser sobrevalorar la complejidad de cada historia para así mostrar una velocidad más alta. Es decir, la medida de la velocidad de un equipo en *Scrum* solo sirve para comparar al equipo consigo mismo y ver si mejora o no aumentando esa velocidad.

Hay una serie de herramientas que permiten hacer un seguimiento del cumplimiento de las estimaciones. Aunque se hablará en detalle de ellas más adelante, merece la pena dedicar ahora un espacio al llamado *burn-down chart*, una gráfica que muestra la evolución del

trabajo realizado de manera efectiva comparado con el previsto. Tal y como se puede ver en la siguiente figura, el eje horizontal representa el tiempo, mientras que el vertical se dedica a la cantidad de trabajo (puntos de historia, número de tareas) que va a realizar. Una línea une la cantidad total de trabajo en el inicio del *Sprint* con el valor 0 al final, lo que señala que todo el trabajo comprometido se ha realizado. Esa línea recta representa la evolución ideal del trabajo en el *Sprint*. Día a día, se irá actualizando con la cantidad de trabajo completado en las unidades que se hayan fijado. Esta gráfica, además de medir la evolución del trabajo del equipo, permite detectar en etapas muy tempranas desviaciones sobre el plan previsto y adoptar acciones correctivas. Y este es uno de los grandes beneficios de *Scrum*.

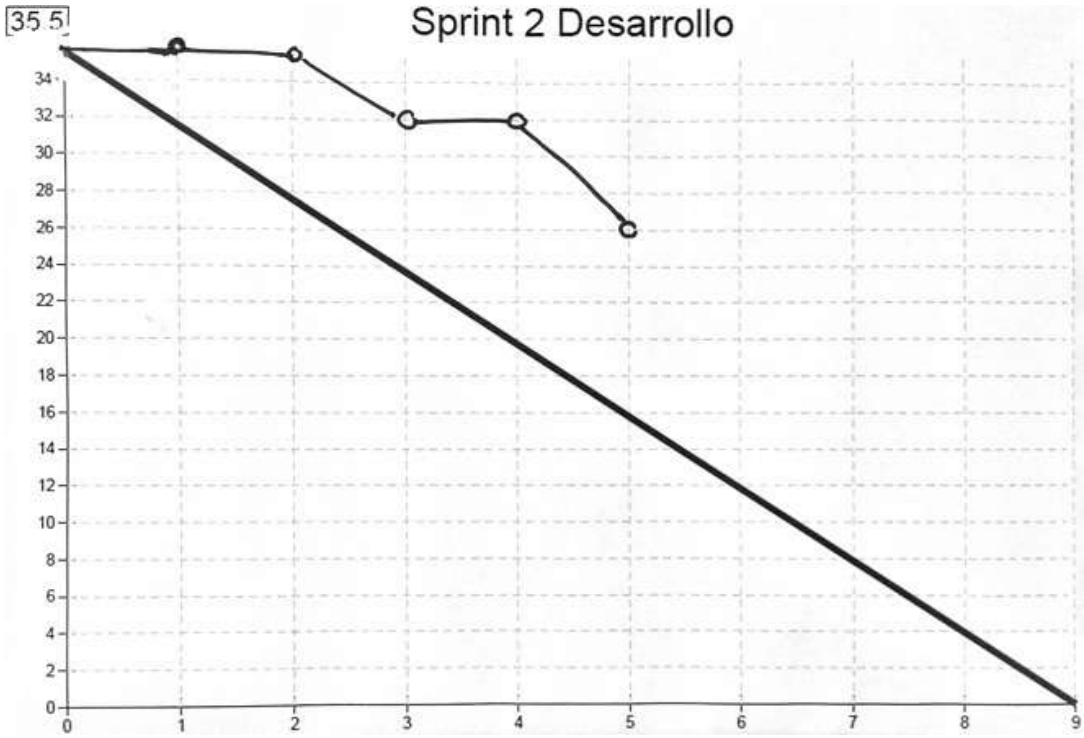


Figura 5.10. Burn-down chart, gráfica que representa la evolución del trabajo realizado frente al estimado.

El cálculo de velocidad, como otro buen montón de tareas (mantenimiento de los *Backlogs*, organización de reuniones, seguimiento del uso de *Scrum*...) corre a cargo del *Scrum Master* o SM, un rol que hemos mencionado ya numerosas veces. Antes de continuar, vamos a dedicar un espacio a hablar de este papel tan importante en *Scrum*.

Pilotando el Sprint: el Scrum Master

Junto con el *Product Owner*, el otro gran papel diferenciado en el mundo *Scrum* es el del SM o *Scrum Master*. Se trata de un papel difícil, ya que, aunque reúne bastantes

responsabilidades, no está dotado del “poder” para ordenar a otros miembros del equipo, sino que su capacidad de influir viene dada más bien por el ejemplo o inspiración que pueda inducir.

El SM ejerce un liderazgo moral, nunca jerárquico. Pero, aunque no tenga una autoridad formal sobre el equipo, sí que tiene mucho que decir sobre cómo trabaja.

El *Scrum Master* es, por encima de todo, el responsable del proceso y de garantizar que se sigue *Scrum*. Se trata de un rol intermedio entre el *Product Owner*, responsable del éxito del producto, y del equipo que se responsabiliza del desarrollo exitoso del trabajo.

Ante todo, hay que fijar que el papel de SM no es el de un jefe de proyecto: con *Scrum* ese concepto desaparece. Si hay una palabra que define al *Scrum Master* es facilitador, y sería muy parecido a un director de orquesta: los verdaderos protagonistas son los músicos, pero el director se encarga de que el talento y trabajo de todos ellos funcione conjuntamente de la mejor manera posible. Armoniza el trabajo de los demás, dando protagonismo a unos cuando otros flaquean y consiguiendo que todo el equipo realice un trabajo conjunto sin estridencias.

Lo que debe quedar claro es que no es el secretario del equipo y asistente del *Product Owner*. Es cierto que suele convocar reuniones y hacer todo lo necesario para que asistan las personas implicadas, pero no está a las órdenes del PO ni debe ser su subordinado para tareas administrativas.

Tampoco es el líder, ni siquiera técnico del equipo. Puede ser una persona de un dominio técnico dentro del área de trabajo, pero eso no le cualifica necesariamente como SM.

Tampoco es un controlador que persigue a los miembros del equipo o al PO: *Scrum* favorece la responsabilidad y la iniciativa, y eso es incompatible con el papel de supervisor y controlador.

Figura 5.11. El Scrum Master es un facilitador, como un director de orquesta que ayuda a los músicos a tocar conjunta y armónicamente.

El cometido principal del *Scrum Master*, encargarse del seguimiento correcto de los principios de *Scrum*, no es una tarea abstracta y se manifiesta en aspectos muy concretos, como por ejemplo:

- Velar por la **productividad** del equipo, lo que se traduce de manera práctica en tratar de aislar al equipo de interferencias externas que puedan distraerle y en resolver los impedimentos que puedan aparecer en el trabajo. Un impedimento es cualquier condición externa que impida llevar a cabo una historia de usuario o una tarea: puede ser no contar con una herramienta o persona, falta de información, necesitar un permiso, alguna contribución externa, etc. Poder resolver impedimentos requiere que el SM tenga un conocimiento profundo de la organización en la que trabaja.
- Debe procurar que fluya la **comunicación** y la **colaboración**. Por eso asiste a todas las reuniones y procura garantizar la asistencia de las personas necesarias para su éxito. También busca este objetivo fuera de las reuniones, en el trabajo del día a día.
- Además, es responsable de introducir y fomentar las **prácticas ágiles**. Por ello, debe conocer profundamente los principios, los métodos y sus variantes y hacer un esfuerzo por difundir este conocimiento entre el equipo y PO. No es necesario que organice e imparta cursos, a veces acciones tan simples como describir el propósito de cada reunión antes de empezar es de gran ayuda.
- Supervisa el **Backlog**, asegurándose que todas las historias están correctamente descritas, priorizadas y estimadas. Se trata de una acción derivada de velar por la productividad, evitando así posibles impedimentos. No hay que olvidar que el responsable del *Backlog* es el *Product Owner*, al que no debe sustituir.
- Analiza la **velocidad** obtenida haciendo un seguimiento de su evolución con las herramientas apropiadas. Si decae o no crece (pudiendo hacerlo), tendrá que lanzar iniciativas que mejoren la productividad.
- Es también el **intermediario** entre el mundo exterior (PO, otros equipos) y el equipo de trabajo. Esto forma parte de su misión de fomentar la productividad protegiendo al equipo de interferencias externas.
- Tiene un papel destacado en la preparación y **formación** del equipo, el PO e incluso los clientes para que adopten las mejores prácticas de *Scrum* en su trabajo. El *Scrum Master* es el primer *coach* del equipo.

¿Es el *Scrum Master* un rol especializado o puede rotarse entre los miembros del equipo? Se trata de un dilema que aparece con frecuencia en la literatura especializada y en los equipos de trabajo. Los autores de este libro nos inclinamos por considerarlo un rol diferenciado y especializado debido, sobre todo, a la formación y experiencia necesarias para

llevarlo a cabo. Eso no impide que haya casos en los que sea viable y no cause conflicto, pero depende fuertemente de la persona que vaya a realizar ese trabajo. Lo que resulta absolutamente incompatible es combinar los papeles de *Scrum Master* y *Product Owner*: no es posible.

Nada impide que un *Scrum Master* lo sea de varios proyectos: únicamente la cantidad de trabajo y la capacidad de la persona determinará que sea o no posible. En este caso, obviamente, se está hablando de un SM especializado que no realiza otra actividad.

¿Qué es lo que define a un buen *Scrum Master*? Hay numerosas listas de atributos y características, pero, para simplificar, se facilita esta lista de los seis atributos principales que selecciona Mike Cohn[24]:

- **Responsable:** El papel del *Scrum Master* es peculiar desde la óptica de la forma tradicional de desarrollar proyectos. No es un jefe de proyecto, ni una persona revestida de autoridad formal, ni siquiera es responsable del éxito final del trabajo (salvo que es compartida con el equipo). Sin embargo, es responsable del proceso y su éxito y, por ello, de que el equipo aplique correctamente *Scrum*, y eso sin la coerción que implica una autoridad formal.
- **Humilde:** El *Scrum Master* no debe restarles protagonismo a los miembros del equipo. No debe destacar ni distinguirse. Convence por medio del ejemplo y la inspiración. Debería decir: “Mira lo que he ayudado a conseguir”, y no: “Mira lo que he hecho”.
- **Colaborador:** *Scrum* se basa en la colaboración y el *Scrum Master* debe ser el abanderado de este principio, fomentando la colaboración y buscando la forma de frenar las actitudes contrarias y egoístas. Por ello, ayuda a crear una atmósfera positiva de colaboración, haciendo que los debates sean constructivos y no deriven en disputas con vencedores y vencidos.
- **Comprometido:** Formalmente, puede parecer que es tanto el *Product Owner* o el equipo quienes tienen los compromisos más fuertes con el éxito del trabajo. Sin embargo, es imposible que el proyecto llegue a buen puerto si el propio *Scrum Master* no adopta una actitud comprometida con el proyecto, sus fines y la forma de llevarlo a cabo. Eso se manifiesta en procurar resolver con rapidez los impedimentos que surjan y ayudar a mantener permanentemente actualizada la información del trabajo, como las gráficas de avance o el *Sprint Backlog*. Como muestra de su compromiso, el SM debería mantenerse ligado al proyecto hasta su conclusión.
- **Influyente:** Al no contar con autoridad formal, el *Scrum Master* no tiene más remedio que convencer por medio del ejemplo y de su capacidad para persuadir a otros. Esto obliga a que un buen *Scrum Master* deba dotarse de unas armas más políticas que metodológicas, técnicas o científicas.
- **Entendido:** Precisamente, una forma de influir en otros es desplegando un conocimiento profundo tanto de *Scrum* y aspectos metodológicos como del campo de aplicación en el que se esté desarrollando el trabajo. El objetivo es entender la

naturaleza de los problemas que pueda tener el equipo en su trabajo.

Nota:

Existe un debate sobre el grado de conocimiento que debe tener el SM acerca del área de conocimiento del proyecto. En general, puede decirse que, aunque su principal cometido es Scrum y la aplicación de este método, es muy conveniente tener un conocimiento suficiente, incluso dominio, del área de conocimiento específico del trabajo. Este es también un argumento a favor de que el SM sea parte de la organización en la que trabaja y no una persona externa que pueda contratarse temporalmente: el conocimiento que posee es estratégico para la organización, por lo que no debería externalizarse ni dejarse fuera de ella.

El papel del *Scrum Master* es decisivo en el desarrollo del proyecto. Esto es especialmente cierto cuando se inicia la adopción de *Scrum*: un mal SM y una mala experiencia de trabajo con *Scrum* puede condicionar la aceptación como método de trabajo. Por ello, es crítico el encaje del SM en el equipo y su capacidad para conectar y transmitir.

A modo de resumen, estas son las responsabilidades concretas del *Scrum Master*:

- Garantizar la aplicación de los métodos ágiles y su conocimiento.
- Facilitar la productividad del equipo, resolviendo impedimentos.
- Trabajar muy cerca del *Product Owner* para garantizar que el *Product Backlog* está priorizado y completo, y que hay criterios de aceptación claros para todas las historias de usuario.
- Medir la velocidad del equipo e identificar puntos de mejora.
- Ayudar a verificar el cumplimiento de los compromisos y que se hayan completado historias y tareas.
- Fomentar el conocimiento de *Scrum*, formando incluso al equipo y cliente. El SM es el primer *coach*.
- Tomar decisiones que luego consensuará sobre el proceso y la aplicación de los métodos.
- Supervisar los *Backlogs* y su contenido y el cumplimiento de las condiciones para dar por cerrada una historia o tarea.

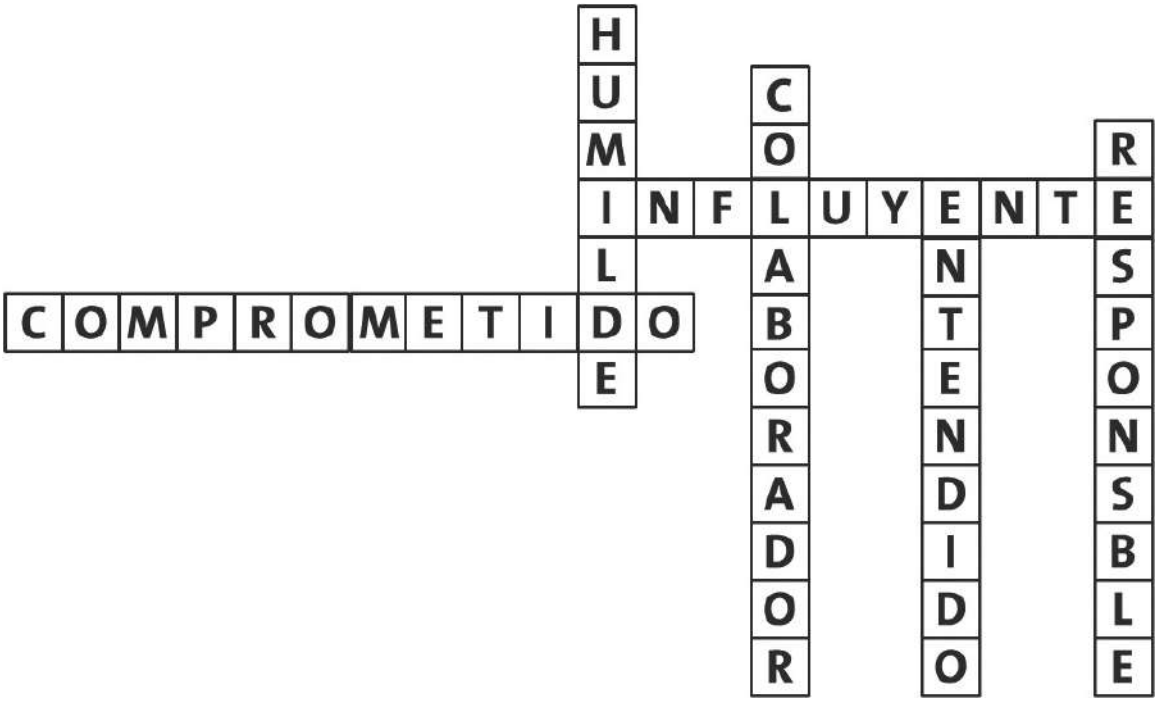


Figura 5.12. Los seis atributos del Scrum Master.

El objetivo último del *Scrum Master* es dejar de ser necesario y hacer que el equipo se auto-organice y aplique perfectamente *Scrum* sin su contribución. Lo ideal es que un equipo de trabajo maduro y productivo no necesite de un SM.

La planificación detallada

Se había dejado el proceso de *Sprint Planning* en el momento en que se había alcanzado un número suficiente de historias como para completar el trabajo del equipo durante el *Sprint*. Ese es un hito importante, pero no se ha terminado aún: hay que desgranar esas historias en unidades más manejables y cercanas al trabajo final del equipo. Se trata de ahondar en la idea de *Scrum* de ir refinando sucesivamente en lugar de hacer planes detallados desde el principio.

Por término general, la planificación del *Sprint* se divide en dos reuniones diferenciadas. La primera, el *Sprint Planning*, de la que ya se ha hablado, sirve para poblar el *Sprint Backlog* con una selección de historias del *Product Backlog*. En ella, es preciso contar con la presencia del PO y de todas las personas que puedan contribuir a hacer que el equipo comprenda perfectamente qué se espera de su trabajo.

La segunda reunión, la planificación detallada o táctica para el *Sprint*, busca dividir las historias de usuario seleccionadas en el *Sprint Backlog* en partes más manejables llamadas tareas. En esta segunda reunión (o segunda parte de la primera), no es necesaria la presencia

del PO. De hecho, el equipo de forma autónoma es capaz de realizar esta división por sí mismo. El objetivo es obtener tareas: entidades pequeñas y manejables, descritas en el lenguaje del dominio del trabajo, (arquitectura, desarrollo software, marketing o el que corresponda) y no en lenguaje de negocio.

Una tarea tiene una descripción muy simple del trabajo concreto que se quiere hacer. Lo ideal es que las tareas puedan realizarse por una persona en un tiempo limitado de entre medio día a tres días, aunque en esto, como en tantas otras cosas en *Scrum*, puede tomarse con una cierta flexibilidad. Una tarea describe un trabajo concreto que debe desempeñar el equipo y sus resultados no tienen por qué ser revisados por el PO. El conjunto de las tareas de una historia sí que da lugar a resultados de producto y sí que tienen que ser validados por el *Product Owner* posteriormente en la reunión de revisión o *Review*.

Es muy conveniente recoger el resultado de esta división en tareas en la herramienta que esté utilizando el equipo para reflejar el trabajo y su evolución, por ejemplo, en un panel con *post-it* usando notas de distintos colores y/o tamaños, asociadas a las de las historias. Las tareas también deben documentarse y contener, además de un nombre suficientemente descriptivo, toda la información que pueda ser relevante para su conclusión.

Aunque a algunos equipos les pueda parecer un trabajo innecesario y redundante, lo cierto es que hacer una división en tareas desde el principio del *Sprint* ayuda mucho a desarrollar y hacer seguimiento del trabajo, aunque luego pueda cambiar y ajustarse.

La división en tareas se hace en una reunión independiente en la que los miembros del equipo que vayan a abordar una historia determinada se ponen de acuerdo entre sí para dividirla. Cuando se consigue la división en tareas de todas las historias, puede darse por concluida la fase de planificación del *Sprint*.

El conjunto de las reuniones de planificación no debe suponer un tiempo excesivo y el que se estime para ello debe ser rigurosamente respetado. No hay reglas establecidas para fijar la duración de estas reuniones, aunque, cuando no hay otra referencia, se puede calcular una hora por cada semana de duración del *Sprint*. La duración definitiva la definirá la práctica y experiencia acumulada. Por regla general, en proyectos nuevos, en sus primeras fases y con equipos inmaduros o que no hayan trabajado antes juntos, las reuniones de planificación duran más que cuando se trata de un proyecto rodado con equipos experimentados. Lo mismo cabe decir de los distintos actores y su participación: con el tiempo el equipo es cada vez más autónomo y requiere menos explicaciones y tiempo para estimar y planificar.

Mantenimiento del Backlog: refinamiento

El conjunto de historias de usuario y las tareas en las que se dividen conforman el *Sprint Backlog*, la definición del trabajo que se va a desarrollar durante la iteración o *Sprint*.

A partir de este resultado, arranca el trabajo propiamente dicho durante el desarrollo del

Sprint. En los siguientes capítulos, veremos cómo discurre este proceso y cómo se evalúan sus resultados.

Sin embargo, hay una actividad que, aunque no está incluida entre las típicas de *Scrum*, conviene hacer inmediatamente antes de cada planificación: es el llamado refinamiento o mantenimiento de *Backlog* (anteriormente se hablaba del *Backlog grooming* pero este término está ahora en desuso). Se trata de garantizar que el *Product Owner* revisa el contenido del *Product Backlog* para depurarlo, añadir las historias necesarias, eliminar las que ya no lo sean, realizar las divisiones necesarias de épicas y temas y, por encima de todo, garantizar que la priorización de historias es la correcta. En realidad, es este un trabajo continuo que no debería tener un momento prefijado para hacerlo, pero concretarlo en una reunión es una forma de garantizar que se hace.

El Refinamiento se realiza antes de cada planificación, excepto para la primera tras el *Sprint* 0, momento en el que se considera que el repositorio del proyecto está perfectamente ordenado y priorizado.

En los casos en los que el refinamiento es una actividad regular y diferenciada, puede convertirse en una reunión a la que asiste todo el equipo *Scrum* (PO, SM y miembros del equipo) con objeto de alcanzar la mejor comprensión posible de cada elemento del *Product Backlog*. También puede ser algo que haga el *Product Owner* con la ayuda opcional del *Scrum Master*.

El resultado debe ser el mismo: un *Product Backlog* revisado y priorizado que haga más productivas las reuniones de planificación, y facilite las labores de estimación y subdivisión en tareas.

Consejo:

Sea o no un procedimiento formal de Scrum, un buen PO deberá dedicar periódicamente un tiempo a revisar el Product Backlog y garantizar que está completo y priorizado. Es una de las principales responsabilidades del Product Owner y tiene un impacto directo en el trabajo del equipo.

En resumen

En este capítulo se ha visto cómo funciona el arranque de una iteración de un proyecto o *Sprint*.

Todo comienza con el mantenimiento del *Product Backlog*, revisando su contenido, enriqueciendo la descripción de las historias y, sobre todo, asegurando que se encuentran debidamente priorizadas.

A continuación, se celebra una reunión de planificación, en la que participa todo el equipo

Scrum (*Product Owner*, *Scrum Master* y equipo de trabajo). En esta reunión y siguiendo el orden de prioridad, el PO va extrayendo historias del *Product Backlog*, las lee y describe y aclara las dudas que pueda tener el equipo sobre su contenido y los criterios para considerarlas terminadas (la definición de hecho o *Definition of Done*). El equipo estima la complejidad y esfuerzo para llevar a cabo cada tarea con alguna de las técnicas disponibles. El proceso se repite hasta que se alcanza un valor cercano a la capacidad de trabajo teórica o histórica del equipo. Las historias que puedan realizarse en el *Sprint* pasan a formar parte del *Sprint Backlog*, con lo que esa reunión puede darse por finalizada.

Con posterioridad, se celebra una reunión de planificación detallada, en la que no es precisa la presencia del PO. En ella, se va a proceder a dividir cada historia de usuario del *Sprint Backlog* en unidades más manejables llamadas tareas. Estas tareas deben poder realizarse por una persona (o dos en entorno de trabajo por parejas) en un tiempo limitado de entre medio y tres días. Cuando se hayan dividido todas las historias del *Sprint Backlog*, puede darse por finalizada la reunión de planificación detallada y, con ella, la fase inicial del *Sprint*. A continuación, podrá empezar el trabajo propiamente dicho.

En el caso de nuestro libro, antes de cada *Sprint* realizamos una breve reunión en la que revisamos la lista de historias de nuestro *Backlog* para ver cuáles abordamos. Por ejemplo, en uno de los últimos *Sprints* teníamos entre las acciones más prioritarias la revisión de los capítulos ya escritos y la selección de imágenes. Como son unas acciones muy genéricas, previamente se dividieron en historias de usuario que abarcaran individualmente cada uno de los capítulos. Uno de nosotros, actuando como *Product Owner* por delegación de la editorial, decidió que tenían más prioridad los primeros capítulos porque nos permitirían entregar el contenido inicial del libro a la editorial para la maquetación. Todo ello es el trabajo previo de refinamiento del *Backlog*.

En la reunión de planificación, estimamos cuántos capítulos podíamos revisar y añadir imágenes, y luego procedimos a dividir estas historias en tareas concretas como “hacer una foto de un *Planning Poker*, darle formato y generar el fichero para la editorial”. Cuando todas las historias seleccionadas para el *Sprint* estaban elegidas y divididas, pudimos empezar a trabajar en ellas.

---

24 En *Leader of the Band*, que se puede encontrar en .

6 Manos a la obra: Desarrollo del Sprint

En este capítulo aprenderá:

- Qué actividades se desarrollan en un *Sprint*.
- Cómo se organiza un equipo.
- Qué herramientas se pueden usar en un proyecto con *Scrum*.

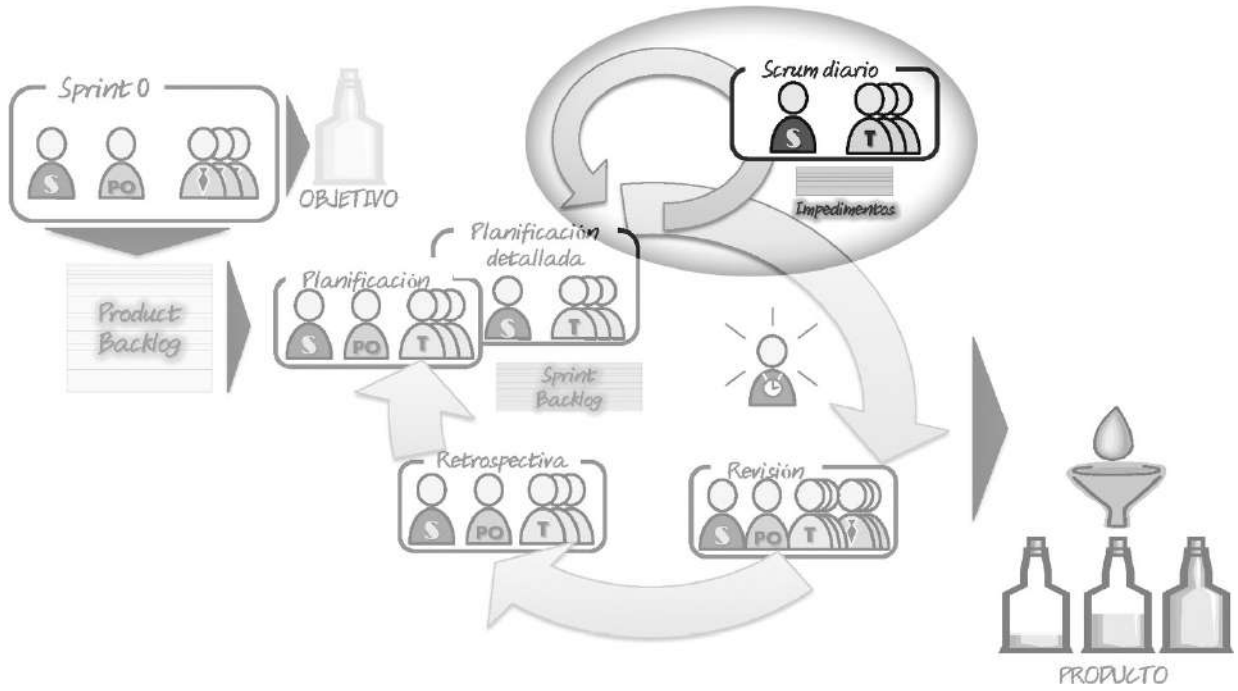


Figura 6.1. El desarrollo del Sprint en el ciclo Scrum.

Introducción

Para el trabajo de nuestro libro, hemos iniciado un nuevo *Sprint* con la planificación de lo que vamos a hacer en él. Puesto que ya llevamos un tiempo trabajando juntos y conocemos nuestro ritmo (velocidad) y el uso de las herramientas, hemos sido un poco ambiciosos. Nos hemos propuesto terminar de revisar y componer tres capítulos (lo que significa realizar correcciones y escoger imágenes), además de escribir otros tres. Esas historias de usuario han sido divididas y las distintas tareas introducidas en el *Backlog* del *Sprint*. Así que, terminada la planificación podemos empezar a trabajar, contrastando periódicamente nuestros avances y superando los problemas que vayamos encontrando.

Veremos en este capítulo cómo es el desarrollo de un *Sprint*, qué implicaciones tiene trabajar aplicando *Scrum* para un equipo, de qué forma se hace el seguimiento de su trabajo y cómo se resuelven los problemas que puedan ir surgiendo. Finalmente, hablaremos de las herramientas utilizadas como apoyo al desarrollo de proyectos con *Scrum*.

El Sprint

Ya se ha dicho que los proyectos que apuestan por *Scrum* se dividen temporalmente en una serie de iteraciones o *Sprints*.

En un proyecto convencional, tras dedicar mucho tiempo y esfuerzo a diseñar el trabajo, cada vez con más detalle, se realiza todo el desarrollo del proyecto de manera continuada para entregar al final del proceso un producto acabado.

La realidad es que en ese tipo de proyectos rara vez se cumplen plazos y costes o lo hacen sacrificando calidad. Pero como, por encima de todo, los requisitos rara vez permanecen estables, es necesario realizar cambios y correcciones sobre un producto terminado que no está preparado para ello.

Scrum aporta una aproximación incremental: el producto se construye poco a poco en ciclos limitados, al final de los cuales hay una versión parcial pero funcional del producto. Por ejemplo, en un proyecto de desarrollo de una aplicación para un dispositivo móvil, se tendría un programa limitado, pero funcionando al final de cada *Sprint*. Ganaría funcionalidades y complejidad en cada iteración, lo que facilita asumir requisitos cambiantes y nuevas características para el producto.

Una forma de entender la diferencia entre *Scrum* y un proyecto convencional es pensar en un pueblo que está creciendo: al principio hay pocas casas y, al final de cada *Sprint*, habría más, pero siempre sería un pueblo, cada vez más grande y con más "funciones" (farmacia, escuela, campo de deportes...). Una pirámide sería un proyecto convencional en el que, hasta que no se pone la última piedra en la cima, no se tiene una pirámide, solo un trapecio cada vez más alto.

Cuando el producto es suficientemente grande, pueden programarse *Releases* o entregas (véase el capítulo 3), que son unas agrupaciones de *Sprints* al final de las cuales se entrega un producto con un conjunto principal de funciones o características. Tanto si hay *Releases* intermedias como si no, al final del proyecto se obtendrá una entrega final con todo el trabajo desarrollado y validado.

La duración del **Sprint** es una decisión importante que tiene influencia en el desarrollo del trabajo. Se fija en el momento de iniciar la primera iteración, incluso antes (en el *Sprint* 0), pero esa decisión no es inamovible: puede ajustarse en función de las necesidades del proyecto y el equipo.

Por término general, se considera que un *Sprint* debe durar entre una y cuatro semanas. Menos tiempo parece escaso para poder desarrollar un conjunto mínimo de funcionalidades. Más tiempo parece demasiado para garantizar la sincronización entre las necesidades del cliente, expresadas en el *Product Backlog*, y la actividad del equipo. Sin embargo, puede haber casos en los que tenga sentido salir de esas recomendaciones.

¿Qué consecuencias tiene una u otra duración de *Sprint*?

Los **Sprints** cortos permiten detectar de forma temprana posibles problemas en el curso del desarrollo. Son más indicados en las etapas iniciales del proyecto, en actividades de innovación o poco definidas, cuando los requisitos pueden ser más inestables y sujetos a variación. También es conveniente cuando el equipo es inmaduro, se desconocen las técnicas

y herramientas usadas o sus miembros tienen poca costumbre de trabajar juntos. Al sucederse con mayor rapidez los momentos de planificación y revisión de resultados, pueden detectarse antes elementos o condicionantes negativos y actuar a tiempo para corregirlos.

Un *Sprint* corto tiene también un coste. Hay que considerar que el tiempo dedicado a planificación y revisión tiene mayor impacto que en *Sprints* más largos, aunque su duración sea proporcional a la duración de cada iteración.

Por eso, cuando hay una relativa estabilidad de requisitos, el entorno es conocido, el equipo maduro y sus miembros acostumbrados a trabajar juntos, puede optarse por **Sprints** más largos. Esto supone depositar una confianza mayor en el equipo, ya que la revisión de resultados se demora más y, con ella, la posible corrección que haya que aplicar.

Esto no quiere decir que tras la planificación se dé un cheque en blanco al equipo para que avance, haga y deshaga por su cuenta: los resultados se pueden ir conociendo a medida que evoluciona el *Sprint*. Lo que sí hay que procurar es reducir, en la medida de lo posible, las interrupciones que puedan desviar al equipo de su objetivo. Es una forma de conseguir el "latido", la estabilidad y carácter predecible del proceso que nos permitirá estimar mejor y evitar los sobresaltos y sorpresas de última hora habituales en tantos proyectos.

Decidir la duración del *Sprint* es una tarea compartida por todo el equipo *Scrum* (PO, SM y equipo de trabajo), aunque sea el *Scrum Master*, por su conocimiento de las técnicas *Scrum*, y el equipo, por su mayor dominio del área de conocimiento del trabajo, quienes tienen más que decir a la hora de alargar o reducir cada iteración.

En cambio, el número y frecuencia de *Releases* es una decisión en la que tiene más peso el *Product Owner*, ya que tiene componentes de gestión y del negocio. Sea corto o largo el *Sprint*, hay que centrarse en la consecución de sus objetivos. Esto supone cuidar determinados aspectos:

- **La estabilidad:** No se puede cambiar su contenido o variar su objetivo, salvo que esté muy justificado y tenga un impacto beneficioso real sobre el trabajo en curso (por ejemplo, cuando se determina que una historia de usuario es innecesaria o cuando es necesario añadir una para poder realizar otras).
- **Obtención de resultados:** El objetivo de cada *Sprint* es obtener un producto parcialmente desarrollado, que crezca incrementalmente. Si no ha habido cambio al final de la iteración, habrá que preguntarse seriamente si se ha planteado correctamente el trabajo. Por supuesto, no se trata de añadir cualquier cosa al producto, sino de enriquecerlo y mejorarlo.
- **Mejora continua:** Porque el tiempo dedicado al *Sprint* es también tiempo para identificar y aplicar mejoras en la forma de trabajar, en la productividad y en la calidad. Por ello, como se verá en el capítulo sobre Retrospectivas, se dedica un tiempo al final de cada *Sprint* para detectar lo que se hace bien, y lo que se puede mejorar.
- **Anticipación:** Un *Sprint* es el paso previo al siguiente y, de la misma forma que un corredor estudia el recorrido para preparar sus próximos pasos, hay que aprovechar el

tiempo de la iteración para anticipar los cambios y retos de la siguiente.

Equipo de trabajo

Antes de profundizar en cómo se realiza el trabajo durante el *Sprint*, parece un buen momento para hablar de quién lo lleva a cabo: el equipo de trabajo.

Se trata del rol más importante en un proyecto, ya que de su motivación, dedicación, buen hacer y entrega depende en gran medida el resultado del trabajo. Además, *Scrum* es una metodología que pone su foco en las personas y sus relaciones, por encima de convenciones, documentos, etapas o trámites.

La forma tradicional de desarrollar el trabajo se basa en la jerarquía, la autoridad formal y la estructuración. Frente a ella, el equipo en *Scrum* se auto-organiza, tiene la responsabilidad final por el éxito del trabajo y es capaz de asumir cualquier actividad dentro de las necesarias para desarrollar el proyecto.

El equipo debe tener un elevado grado de compromiso. Debe ser capaz de auto-organizarse, frente a un equipo tradicional donde un jefe de proyecto asignaba a cada persona tareas concretas y fijas. El equipo también debería ser capaz de realizar todas las actividades requeridas en el trabajo.

Con la ayuda del *Scrum Master*, el equipo será capaz de seguir la evolución de su productividad y hacer un proceso continuado de mejora, lo que incluye también sus conocimientos y habilidades, y la aplicación de las mejores prácticas en su trabajo.

¿Cómo es el equipo de trabajo en *Scrum*? Pues lo podemos definir por estas características (ideales):

- **Diversificado (**cross-functional**):** La idea es que el equipo de trabajo sea capaz de realizar todas las funciones necesarias para el desarrollo del trabajo, en lugar de contar con equipos especializados para cada una de ellas. Se gana en continuidad en el trabajo, estabilidad en el equipo y una visión amplia y enriquecedora de la forma de abordar el proyecto. Esto significa que, por ejemplo, un equipo dedicado al desarrollo software contará en él con perfiles especializados como la gestión de calidad, la interacción de usuario, el trabajo con bases de datos o la programación Web.
- **Autónomo y auto-organizado:** El equipo debe ser capaz de avanzar sin tener que contar con la presencia e intervención continua de líderes o jefes. Ya se ha comentado que el *Scrum Master* no es un líder ni un gestor, sino un facilitador, y es que su objetivo final es ser prescindible y que el equipo pueda realizar su trabajo *Scrum* de manera autónoma. Esa autonomía debe permitir al equipo ser capaz de hacer un seguimiento de sus propios progresos y actuar cuando detecte desviaciones con respecto a sus propias estimaciones.

- **Carecer de jerarquías y niveles:** No obstante, no siempre es posible, aunque los grados no deberían interferir en el trabajo y todos los miembros del equipo deberían ser vistos como iguales.
- **Responsable:** En general, el compromiso y la responsabilidad son unos de los principios más destacados de *Scrum* y la autonomía que se propugna para el equipo debe alcanzarse por este medio. El equipo es especialmente responsable del producto desde el punto de vista técnico y de su calidad.
- **Estable y dedicado:** Una vez más es un ideal que no siempre es fácil de alcanzar. En principio, el equipo debería ser el mismo a lo largo de todo el proyecto, lo que permite reforzar el compromiso y ganar en estabilidad y predictibilidad. También debería ser un equipo dedicado, de forma que sus miembros, salvo contadas y puntuales excepciones, tengan como principal (y único) trabajo el relacionado con el proyecto. A veces hay perfiles muy especializados cuya presencia no puede garantizarse salvo puntualmente, pero esa debería ser la excepción y no la norma.
- **Comprometido:** El compromiso es uno de los principales ejes de *Scrum* y es algo que afecta de lleno al equipo. Este compromiso se manifiesta en la mejora continuada de la calidad y la productividad, en obligarse a completar una determinada cantidad de trabajo en cada *Sprint*, a comunicar, participar y ayudar al resto del equipo. No se puede aplicar *Scrum* sin el compromiso del equipo.

Figura 6.2. Un equipo comprometido y autónomo es el pilar sobre el que se apoya Scrum.

Nota:

*Hay que recordar que en Scrum se habla de dos equipos: el equipo de trabajo propiamente dicho y el "***equipo Scrum***", que incluye al Product Owner, al Scrum Master y al equipo de trabajo.*

El equipo tiene, además, varias responsabilidades en *Scrum*.

La primera, obvia y más importante, es realizar el trabajo del proyecto. Este trabajo debe hacerse con la máxima calidad y dentro de un marco de mejora continuada. El equipo debe identificar, con ayuda del *Scrum Master,* formas de mejorar su productividad, de ser más eficientes y de aumentar la calidad del producto. Por supuesto, debe atender a los requisitos de los *stakeholders*[25] expresados a través del *Product Owner*.

Además de esta, hay otras responsabilidades, no menos importantes:

- El equipo es quien se encarga de la **estimación** de los elementos del *Backlog*. En el capítulo anterior, ya se han descrito diversas técnicas para hacerlo. Ante un requisito expresado como historia de usuario en el *Product Backlog*, el equipo debe estimar el esfuerzo necesario para llevarlo a cabo con los puntos de historia que, como se ha comentado, no se corresponden necesariamente con una unidad de tiempo determinada, sino que son más bien una forma de comparar historias entre sí. Esta estimación se realiza durante la primera fase de la reunión de planificación y requiere que el equipo atienda a la descripción que hace el *Product Owner* de cada historia y haga todas las preguntas precisas para comprender perfectamente lo que requiere de su trabajo.
- La **división de las historias en tareas** es una de las responsabilidades del equipo que se realiza durante la segunda fase de la planificación o planificación detallada. Esta actividad es muy importante porque supone la traducción del lenguaje de negocio (historias de usuario) al lenguaje del dominio de aplicación del proyecto, el de las tareas: desarrollo software, arquitectura, diseño naval, marketing, banca, diseño editorial, jardinería... Esta división persigue definir con un mayor grado de detalle el trabajo del equipo, lo que supone: que las tareas tengan una duración delimitada (idealmente entre medio y cuatro días de trabajo); que tengan una clara definición de cómo dar por cumplida una tarea o "definición de hecho" y que se pueda estimar una duración para ellas, ahora sí en tiempo.
- Comprometerse a **realizar una determinada lista de historias** en cada *Sprint*. Por un lado, puede estimar qué sería capaz de realizar por medio de la estimación, pero el equipo además debe comprometerse a completar una parte significativa de ese trabajo.
- **Participar y colaborar**. Aunque todos los miembros del equipo deberían asistir a todas las reuniones a las que se les convoca, no es estrictamente obligatorio que participen activamente en todas ellas. Sin embargo, eso no debe ser la actitud habitual,

sino puntual. De todos los miembros del equipo se espera un alto grado de participación y colaboración con su grupo y con el conjunto del proyecto para la obtención de los objetivos de productividad, de calidad y del proyecto.

- **Producto**. Los miembros del equipo son los responsables principales del objeto del proyecto en el que trabajan desde el punto de vista técnico o de su dominio de aplicación. Aunque la responsabilidad del proyecto es compartida por todo el equipo *Scrum* (PO, SM y equipo), el equipo de trabajo lo es directamente de cómo esté construido el producto, de la misma forma que el *Scrum Master* lo es del proceso y el *Product Owner* es responsable del éxito del producto desde el punto de vista de negocio. Esa responsabilidad exige que los miembros del equipo se involucren activamente en el diseño y construcción y no sean meros ejecutantes de las órdenes de los demás.

La dimensión y ubicación del equipo

Hay mucha literatura al respecto del tamaño del equipo. Lo cierto es que no se trata de un tema menor: para equipos muy pequeños, *Scrum* puede suponer una innecesaria carga adicional de trabajo que se podría evitar aplicando otro tipo de mecanismos. Equipos muy grandes pueden volverse inmanejables, aunque eso ocurre con *Scrum* y con cualquier otro método.

Por término general, se recomienda un tamaño de equipo entre los 4 y 10 o 12 miembros[26]. Aunque como recomendación que es, está sujeta a matices y excepciones. Un equipo de tres, como el que ha elaborado este libro, puede gestionarse con *Scrum*, aunque un equipo de dos puede difícilmente considerarse equipo.

Son admisibles equipos más grandes, aunque los problemas de comunicación y comprensión (además de la logística), crecen exponencialmente con el tamaño del equipo. A partir de cierto número de personas hay que considerar seriamente su división y escalar *Scrum*. Este modelo (del que se hablará más en el capítulo 12 de este libro) significa, en síntesis, crear varios equipos con su propia estructura, ciclo y *Scrum Master*, y luego crear sobre ellos una pequeña organización que se encargará de garantizar la unicidad de objetivos y la sincronización de trabajos.

La división del equipo puede deberse no solo a su tamaño. Puede ocurrir que haya una distribución geográfica que dificulte el flujo de información y la colaboración. O que haya una diferenciación muy clara de actividades y un tamaño de equipo en cada una de ellas que facilite esa división.

La separación geográfica es un factor que dificulta el trabajo en equipo, algo clave para *Scrum* (y, en realidad, en cualquier actividad en colaboración). Se ha demostrado muchas veces que la productividad aumenta cuando la comunicación es más fluida y eso se consigue especialmente si el equipo comparte un mismo espacio físico. De hecho, uno de los

paradigmas del método ágil de desarrollo software llamado XP (*eXtreme Programming*, Programación Extrema), del cual se habla en otro capítulo, es la programación por pares en la que dos personas trabajan sobre un mismo fragmento de código codo con codo, sentada una junto a la otra.

Así que una recomendación recurrente en muchos de los textos sobre *Scrum* y en la propia experiencia de los autores es procurar la mayor proximidad física posible del equipo. Lo ideal sería poder disponer de una sala de proyecto que acogiera a todos los miembros, con espacio común para reuniones y debates, y un lugar donde poder colocar un panel donde se refleje el estado del *Sprint* con las historias y tareas en curso.

Los medios tecnológicos actuales son una gran ayuda y conviene apoyarse en ellos para superar las barreras geográficas: mensajería, herramientas de colaboración o videoconferencia serán una gran ayuda, aunque nunca pueden suplir la presencia física en una misma sala del equipo.

Truco:

Siempre hay forma de sortear las limitaciones del entorno del proyecto, en este caso la proximidad. En un proyecto distribuido, pero con los miembros relativamente próximos geográficamente, se puede habilitar un día a la semana para trabajar todos juntos en un mismo espacio ocupado temporalmente. En general, alternar en la medida de lo posible presencia física con comunicaciones electrónicas es una solución que mitiga los inconvenientes de mantener un equipo de trabajo disperso.

El ciclo dentro del Sprint: Daily meeting o Scrum diario

El periodo de tiempo entre el momento en el que termina la planificación y cuando comienza la revisión es el dominio del equipo de trabajo. Durante ese tiempo se van a ir realizando las tareas identificadas en la planificación detallada, sin olvidarse de la coordinación y flujo de la información.

Concluida la planificación, arranca el trabajo y, en él, la mecánica es simple y repetitiva: cada miembro del equipo selecciona la siguiente tarea que va a abordar de acuerdo con el resto de los miembros (para evitar que, por ejemplo, dos personas seleccionen la misma) y siguiendo la priorización del *Backlog* establecida por el PO. Usando las herramientas que se hayan seleccionado para el trabajo, se marca la tarea para que aparezca en curso, progresivamente se refleja su evolución y finalmente se informa cuando haya completado.

En realidad, cada día de trabajo dentro de un *Sprint* puede considerarse como un pequeño ciclo, de la misma forma que lo es cada *Sprint* dentro de las entregas o *Releases* si las hubiera, y estas con respecto al conjunto del proyecto.

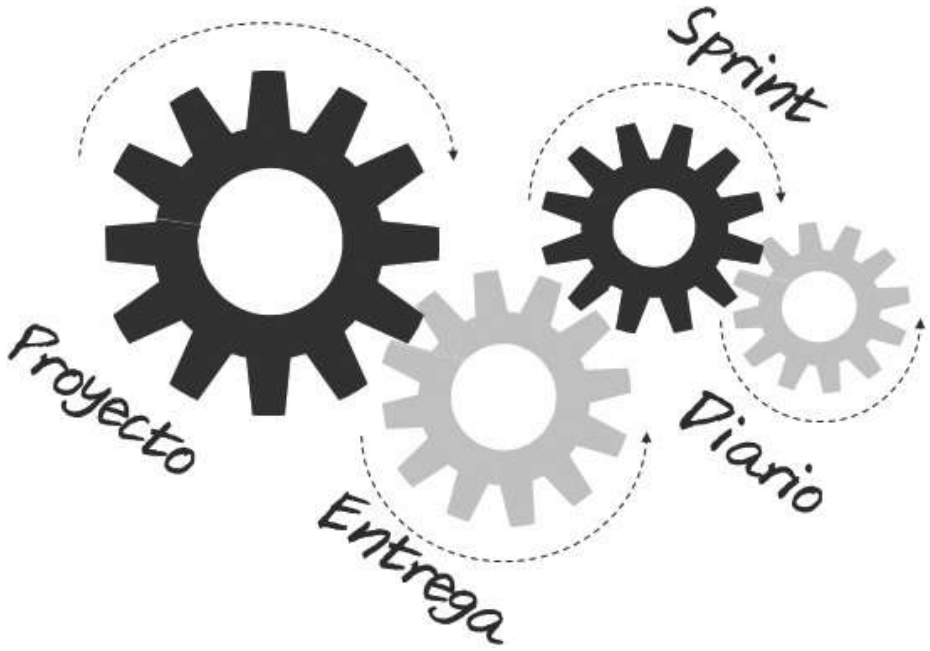


Figura 6.3. Los engranajes de los ciclos de Scrum.

Para facilitar la sincronización entre todos los miembros del equipo, mantener el ritmo y "tensión" y para fomentar la comunicación interna, *Scrum* propone la **Daily Meeting**, reunión diaria, **Scrum** Diario o **Daily Scrum**. Es una reunión informal que se realiza diariamente, en la que participan todos los miembros del equipo y el *Scrum Master*, y a la que puede asistir el PO.

En ella, cada miembro del equipo, de manera voluntaria (sin turnos o listas), pasará a comentar: qué actividades ha realizado, qué actividades va a realizar a continuación y qué impedimentos ha encontrado para continuar su trabajo. Estos son los puntos que debe tratar y ninguno más. No hay que entrar en el detalle de lo que ha hecho (salvo que sea imprescindible, muy relevante para los demás o tenga implicaciones en el trabajo de otros); no hay que discutir las soluciones que se han adoptado o podrían adoptar; no hay que divagar sobre si el impedimento tiene su origen allí o allá, solo sobre cómo resolverlo y quién puede hacerlo. Hay que centrarse en el objetivo de la reunión y hacerla breve y productiva. Si quedaran temas pendientes, las personas directamente afectadas las tratarán con calma, evitando así distraer a todo equipo. Si afectara a todo el equipo, se trataría igualmente fuera de la *Daily* para mantenerla breve y centrada en su propósito.

Nota:

Aunque la Daily Meeting es, por definición, una reunión que se celebra cada día, puede ocurrir que haya circunstancias que obliguen a elegir otra frecuencia. Scrum es muy flexible y permite hacer ajustes y adaptaciones como estas, aunque eso sea también un peligro y haya equipos que fuercen tanto Scrum que acabe por conservar solo el nombre (véase el **capítulo 9** *y los llamados "Scrum buts"). De todas formas, aunque el equipo y*

Scrum Master (y con el visto bueno del coach cuando lo haya) acuerden reunirse cada dos días, por ejemplo, hay que garantizar el flujo constante de la información y el conocimiento actualizado del estado del Sprint.

En la reunión diaria el SM participa como facilitador. Hay que recordar que no es un gestor, por lo que no hay que esperar que sea el *Scrum Master* quien dirija la reunión dando paso a cada miembro del grupo. Su cometido es más bien el de "empujar" y facilitar su desarrollo y algo muy importante: tomar nota de los impedimentos que haya para seguir su solución.

La *Daily Meeting* debe ser una reunión muy breve, en la que deben dejarse de lado las discusiones en detalle para que sean tratadas en otro momento. Hay que ser muy rigurosos con esto. Se corre el peligro de que las reuniones diarias se conviertan en debates detallados e inacabables sobre aspectos puntuales. Estas situaciones suelen acabar con la "desconexión" de los miembros del equipo que no participan activamente en la discusión y, con ello, pérdida de tiempo y desmotivación. Hay que ser muy rigurosos: cuando la información que se proporcione exceda lo necesario para una reunión diaria, el SM o cualquier miembro del equipo debe cortarla y dar paso a otro punto o a otra persona.

Truco:

Un riesgo habitual es que las reuniones diarias se alarguen más de lo conveniente. El Scrum Master deberá buscar formas de dinamizarlas e insistir en su contenido, duración y alcance. Hay muchos trucos para esto, pero casi todos pasan por reducir la comodidad que invita a los asistentes a extenderse. Por ejemplo, realizarlas de pie es una forma muy efectiva de que cada miembro no sienta la tentación de extenderse sin razón. Los autores conocen casos más extremos en los que se ha optado por sujetar un objeto pesado mientras se habla, pero irónicamente eso deja en situación de ventaja a los miembros más fuertes físicamente del equipo.

Cuidado:

Hay equipos que optan por un sistema de "multas" para controlar situaciones como retraso o falta de asistencia en reuniones, extenderse más de lo debido, interrumpir, etc. Se trata de un tema delicado que hay que tratar con cuidado. En primer lugar, no debe imponerse un sistema de castigo si no es con el consentimiento explícito de todos los miembros sin excepción del equipo, y con unas reglas muy claras. En caso contrario debe olvidarse. Y la naturaleza de la multa debe ser casi simbólica: por ejemplo, un euro cada vez que se llegue manifiestamente tarde y con el objetivo de que ese dinero revierta en todo el equipo, por ejemplo, para pagar unos dulces o una cena.

Las *Daily Meetings* permiten tomar el pulso del trabajo y detectar de manera temprana problemas que de otro modo podrían crecer y convertirse en obstáculos serios. Con ellas, se garantiza un conocimiento actualizado del estado de los trabajos por parte de todos los miembros del equipo, lo que es una forma de incrementar su grado de compromiso, la sincronización y la auto-organización.

La información recibida debe actualizarse, preferiblemente en un panel o herramienta que sea fácilmente accesible para todos. Además de esta información de estado y evolución que permite conocer cómo se está desarrollando el *Sprint* y que permite ir confirmando las estimaciones de velocidad hechas en la planificación, hay un resultado muy importante de cada reunión diaria: los impedimentos.

Backlog de impedimentos

El **Impediments Backlog**, pila de impedimentos o **Backlog** de impedimentos es un repositorio que recoge todo aquello que impide alcanzar los objetivos del proyecto. Un impedimento puede ser la carencia de una determinada información o herramienta, la imposibilidad de reunirse con una persona clave, de acceder a un libro o contar con un material de apoyo. Pero también un impedimento es todo aquello que amenace con degradar la calidad del producto final.

Los impedimentos son identificados por cualquier persona del equipo *Scrum*, aunque lo habitual es que sean los miembros del equipo de trabajo quienes los encuentren y destaquen, ya que es ahí donde con mayor facilidad pueden surgir obstáculos.

Aunque los impedimentos pueden aparecer y deben inventariarse en cualquier momento, la reunión diaria tiene como uno de sus objetivos hacerlos aflorar.

Un impedimento debe ser descrito, si es posible, identificar su solución y asignar un responsable, aunque por término general es el *Scrum Master* quien se encarga de seguir su resolución.

Para que quede constancia de su existencia y estado, los impedimentos se guardan en una pila, repositorio o *Backlog*, similar al del producto o al del *Sprint*.

Este repositorio es gestionado por el *Scrum Master* y se mantiene actualizado a lo largo del *Sprint* con todos los impedimentos que se detecten y que se manifiesten en las reuniones diarias. Es muy importante priorizar los impedimentos para su resolución de acuerdo con el impacto que tengan en las actividades del proyecto.

Cada miembro del equipo debe mencionar en la reunión diaria qué actividades ha realizado, qué actividades piensa realizar y qué impedimentos ha encontrado. De la misma forma, hay que hacer una revisión de los impedimentos abiertos y dar cuenta de los resueltos.

Los impedimentos bloquean la resolución de tareas e historias de usuario y ese estado debe quedar reflejado en las herramientas que use el equipo. En un tablero o panel habrá que dejar una columna especialmente habilitada para dejar ahí las etiquetas que representan a las

tareas impedidas.

Cuidado:

Los impedimentos no pueden quedar sin responsable. El Scrum Master lo es de su seguimiento, pero no necesariamente de su resolución. La mejor manera de conseguir que un impedimento no se resuelva es dejarlo sin responsable encargado de su resolución.

Herramientas para Scrum

Scrum define una serie de artefactos que son en sí mismos herramientas que ayudan en el control y seguimiento de un proyecto: los distintos *Backlogs* y el *burn-down chart*. Pero, además, hay una serie de aplicaciones software que hacen más fácil la vida al equipo *Scrum*. En esta sección se hará un repaso a algunos de ellos.

Herramientas básicas: artefactos y paneles

En *Scrum*, la herramienta más útil es también la menos sofisticada: un panel con una serie de etiquetas pegadas en él. Posiblemente haya otra aún más simple y útil, que es facilitar la comunicación entre los miembros del equipo, aunque seguramente haya muchas personas que se resistan a considerarla una herramienta.

La forma más simple de representar y manejar los artefactos es usar paneles o tableros con etiquetas adhesivas o *post-it*. Cada una de esas etiquetas representa una historia de usuario o una tarea, que se pueden disponer en el tablero de acuerdo con el orden más conveniente para equipo y proyecto, aunque el más recomendable es siempre la prioridad que fija el *Product Owner*.

Este panel se usa, sobre todo, como forma de representar el *Sprint Backlog* y recibe entonces el nombre de tablero de tareas o *task board*. Se trata de un sistema extremadamente sencillo de mostrar toda la información de un vistazo si se ubica en un lugar visible para todo el equipo. Permite tener en todo momento un estado actualizado de la evolución del *Sprint*.

En el panel, que puede ser móvil o la propia pared, se dibujan unas áreas o columnas que representan los distintos estados de historias de usuario y tareas (pendiente de iniciar, en curso, terminada, impedida). Unos *post-it*, etiquetas autoadhesivas, carteles, textos escritos, etc. que contienen las historias de usuario y, si el formato lo permite, las tareas en las que se dividen.

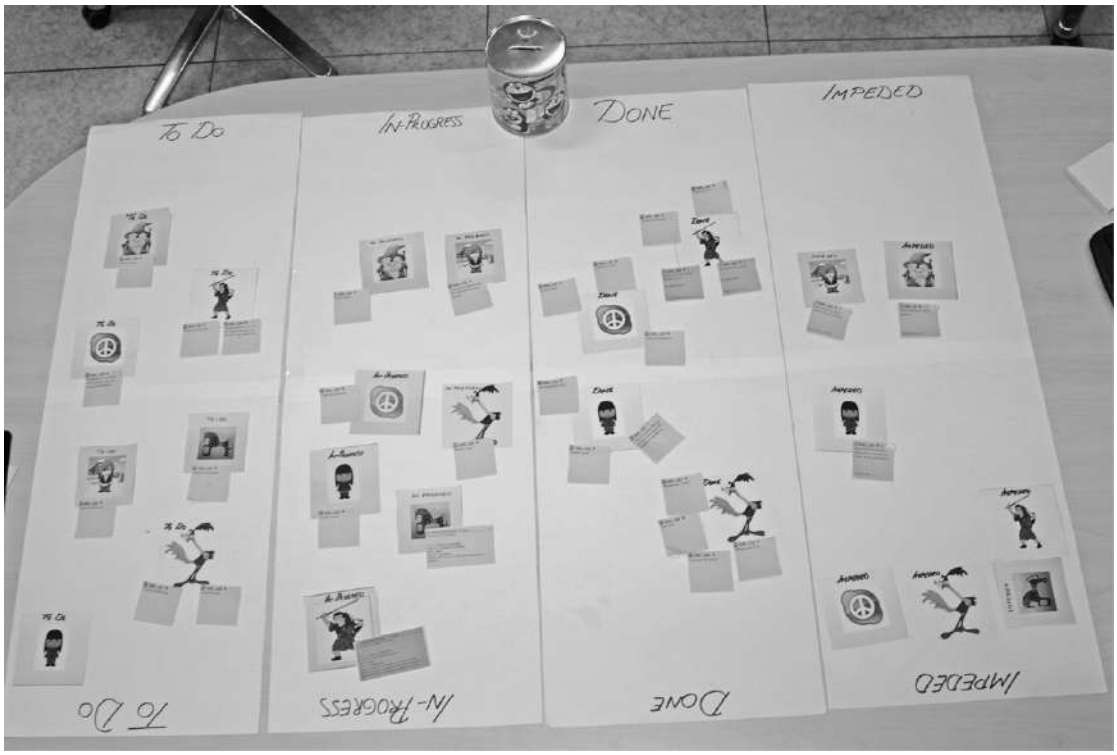


Figura 6.4. Un ejemplo de panel real.

Cuando se va a iniciar una tarea, se mueve del área “pendiente de empezar” al área “en realización”. Usando símbolos y distintos colores para letras o etiquetas, se puede enriquecer la información, añadiendo datos sobre los componentes a que se refieren, quién trabajan en ello, a qué *Release* corresponde, etc. Como se ve, es una herramienta muy simple, fácil de utilizar y útil.

Otra herramienta muy importante es el diagrama conocido como *burn-down chart* (la traducción “diagrama de quemado” parece poco afortunada y apenas se usa).

El *burn-down chart* representa la evolución del trabajo durante un *Sprint*. El eje X (horizontal) muestra el tiempo, expresado como días, desde el inicio hasta el final del *Sprint*. El eje Y (vertical) muestra la cantidad de trabajo que se debe realizar. Esa cantidad de trabajo puede medirse en puntos de historia o como la suma de entidades (tareas e historias de usuario) que se van a realizar a lo largo del *Sprint*. También puede aplicarse al conjunto del proyecto, aunque es algo menos habitual.

Una línea recta diagonal marca la evolución ideal del trabajo: se sitúa al principio en el número total de puntos o entidades que se van a resolver y llega hasta el final del periodo señalando cero, o que se ha resuelto completamente. La siguiente figura muestra un *burn-down chart* en una herramienta informática.

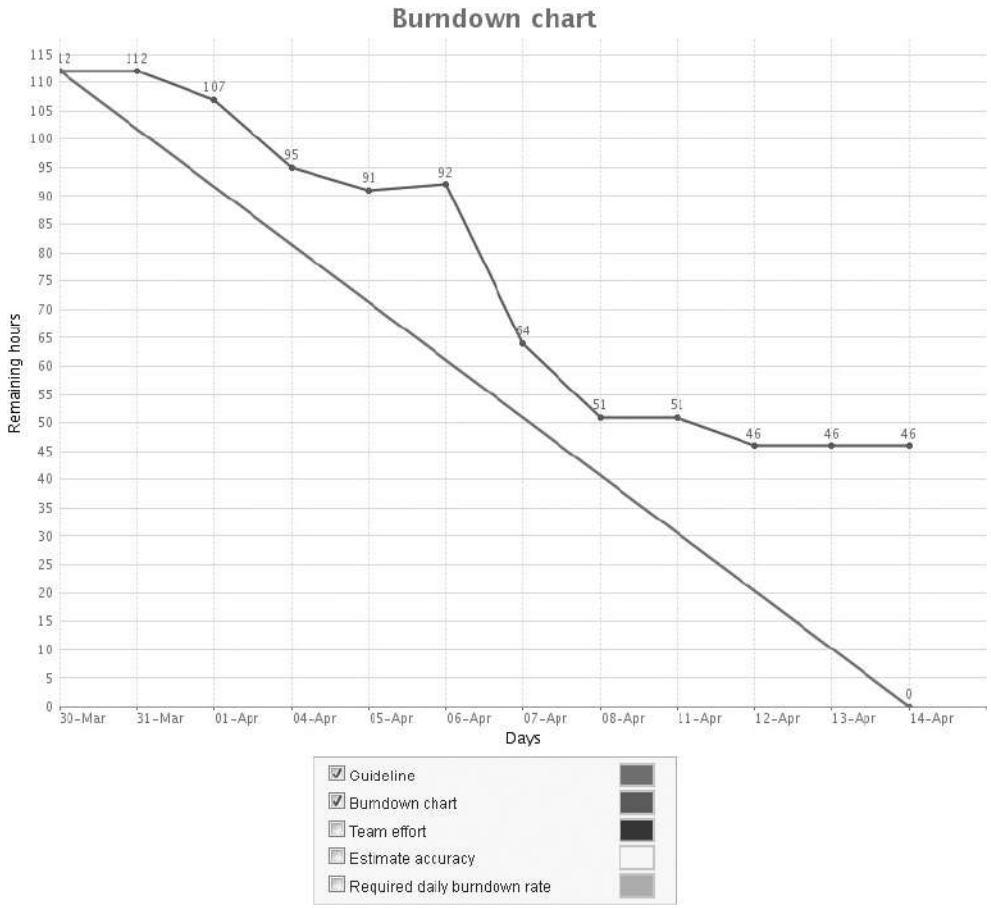


Figura 6.5. Burn-down chart.

Periódicamente (mejor día a día), el SM se encarga de actualizar la cantidad de trabajo realizado y pendiente. Si se utiliza una aplicación informática, esta actualización será automática. Puede ocurrir que en un día no se resuelva gran cosa (e incluso aumente, aunque no sea muy ortodoxo, cuando ha sido necesario añadir tareas que no se habían contemplado inicialmente al *Sprint Backlog*), mientras que en otros se avance mucho. La distancia entre la evolución real y la ideal o estimada va a ir señalando si el trabajo se está realizando a un ritmo adecuado, o si hay algo que corregir en el proceso. Se trata de una herramienta básica de consulta diaria.

Diagramas como el *burn-down chart* también pueden construirse usando paneles, papel y rotuladores, aunque en este caso resulta más costoso reflejar los cambios.

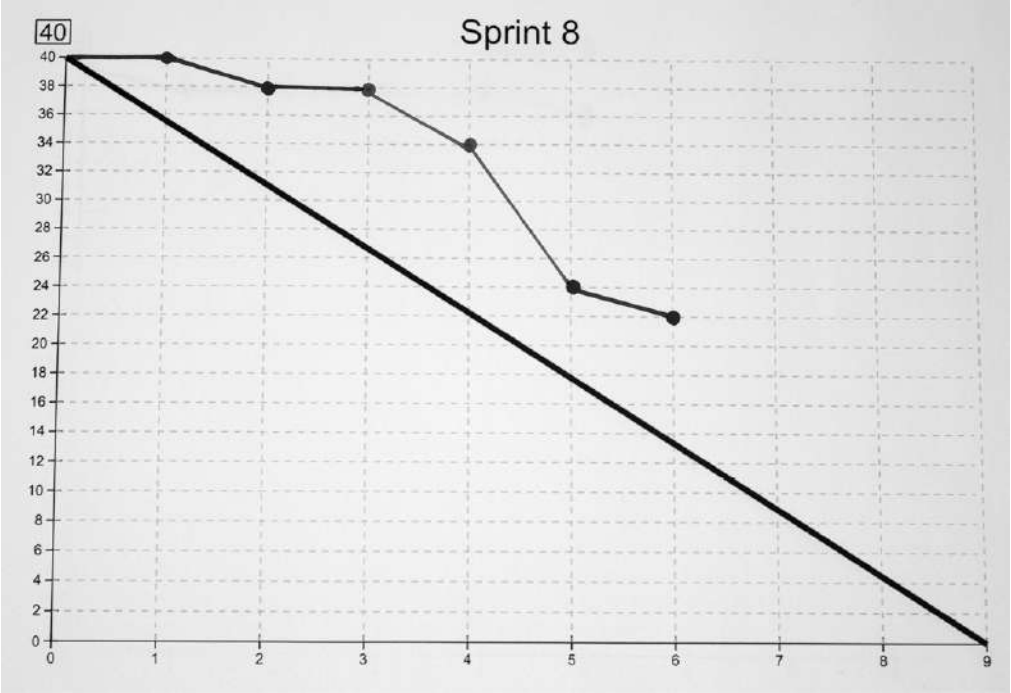


Figura 6.6. Un Burn-down chart mantenido manualmente.

El *burn-down chart* da información inmediata y de un vistazo de la evolución del *Sprint* y ayuda a preparar acciones correctoras en momentos muy tempranos. Si la línea de evolución se aleja rápidamente por la parte superior de la línea diagonal ideal, el diagrama está diciendo que el ritmo de resolución de historias es muy bajo y que quizá resulte conveniente pensar en reducir el alcance del *Sprint*. Si la línea de evolución se muestra, por el contrario, por debajo de la diagonal, el ritmo de resolución es superior al previsible y quizá se pueda concluir el trabajo antes de que termine el *Sprint*, por lo que habrá que pensar en extraer nuevas historias del *Backlog* o contar con algo de tiempo extra al final del *Sprint* para realizar otras tareas como refactorizar código u organizar repositorios.

Herramientas informáticas

Paneles y tableros son fáciles de usar y muy accesibles, pero a cambio requieren esfuerzo manual continuado y una cierta destreza (además tener inconvenientes como etiquetas que se caen, pérdida de versiones anteriores y demás). Por ello, existen herramientas informáticas que permiten reflejar el contenido y estado de un proyecto desarrollado con *Scrum*.

Estas herramientas son necesarias para equipos distribuidos, ya que no pueden tener acceso constante al panel físico, pero sí a una herramienta informática.

Las más básicas son hojas de cálculo en las que cada celda refleja una tarea, historia, épica o tema. Las columnas representan los estados u otra clasificación de la información relevante para el proyecto.

Hay varias plantillas que funcionan sobre Excel. También se pueden usar versiones más sencillas en servicios en red, como Google Docs o Trello. Todas estas herramientas operan de forma parecida. Son compartidas por todos los miembros de equipo que van actualizando el estado de sus actividades en ellas. De esa forma es posible tener una imagen actualizada al momento del *Backlog* de producto y del *Sprint*, y ver cómo evolucionan las gráficas *burn-down*.

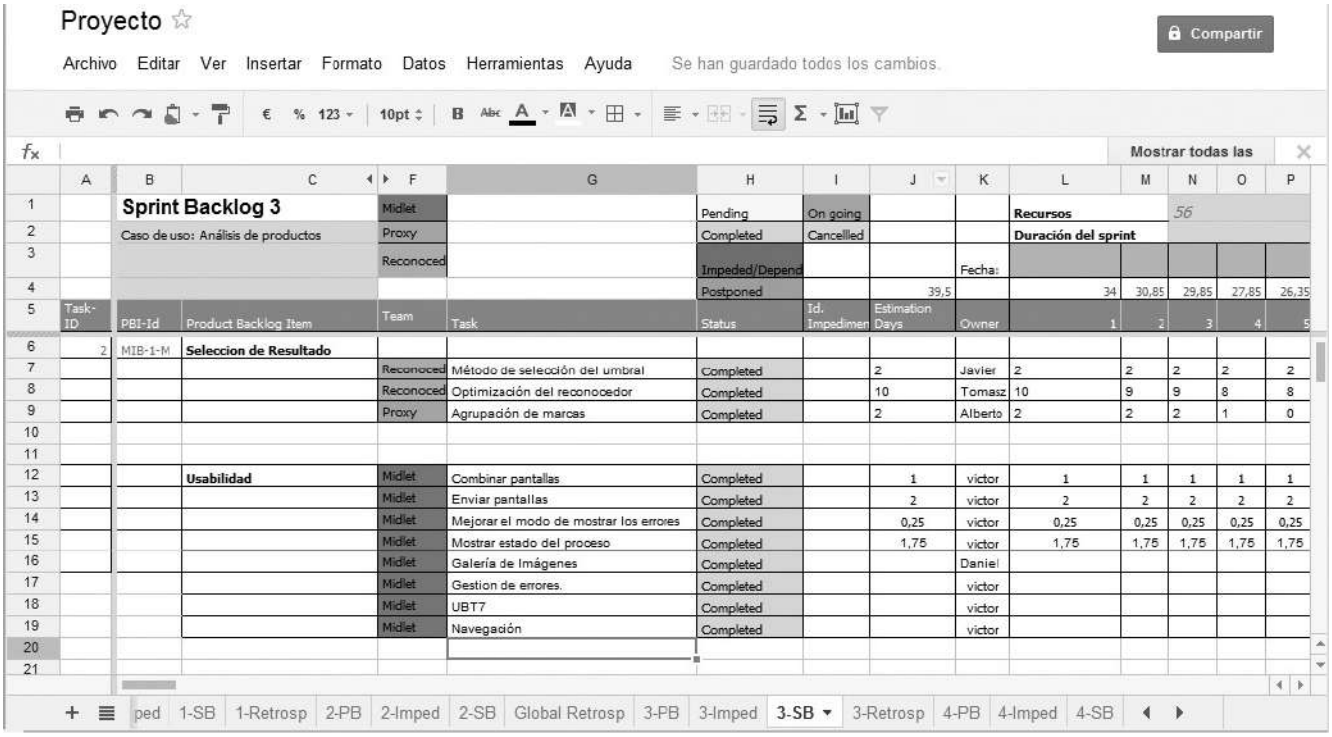


Figura 6.7. Hoja de cálculo para Scrum.

Una herramienta muy difundida es JIRA y está tan extendida que hay artículos que advierten sobre el riesgo de confundir el uso de JIRA con la aplicación de *Scrum*. Nació como herramienta para dar soporte a la gestión de tickets, pero la extensión o plugin GreenHopper[27] permite incorporar los elementos de *Scrum*.

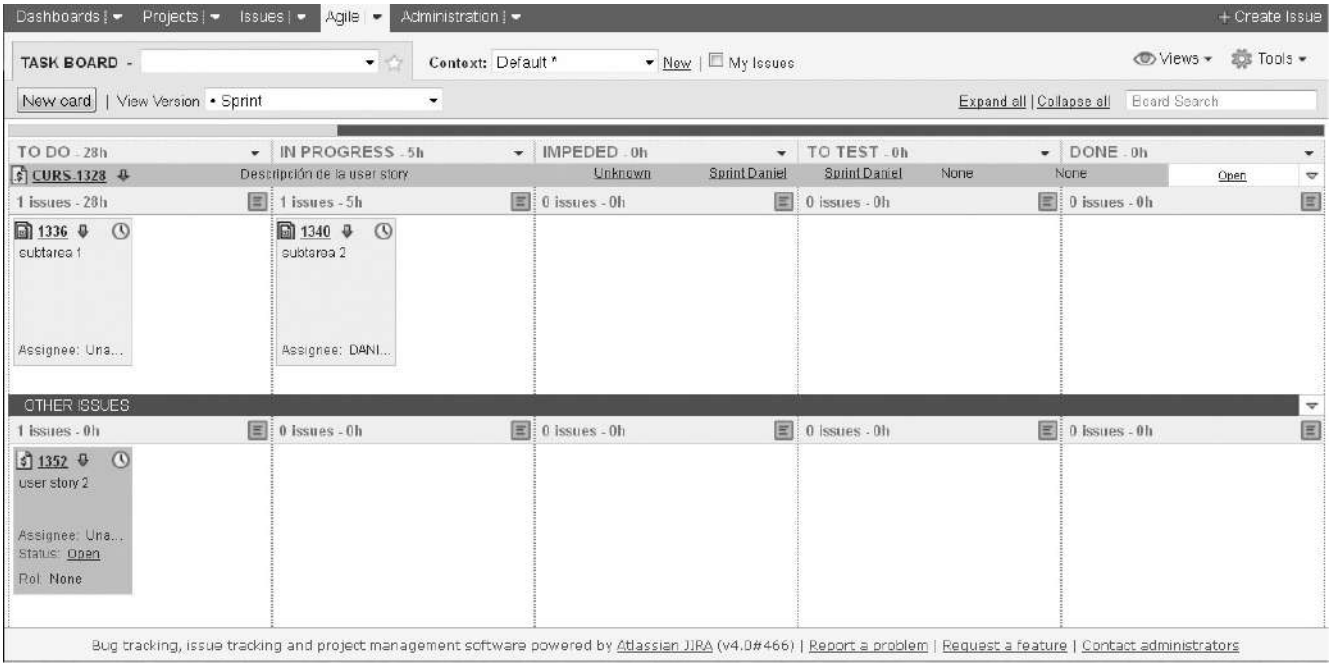


Figura 6.8. Panel en la herramienta JIRA.

Para cada entidad (historia, tarea, épica) hay una completa ficha y se permite reflejar estimaciones, agregar textos y documentación, establecer relaciones entre entidades y un largo etcétera de funciones adicionales. La información se dispone de forma parecida a un panel con las columnas que representan el estado de las historias y tareas.

También genera gráficas tipo *burn-down* con datos actualizados por *Sprint* y conjunto del proyecto.

JIRA está pensada para empresas con múltiples equipos trabajando en distintos proyectos. Permite un uso compartido de la aplicación, de forma que todos los miembros del equipo pueden conocer el estado actual y realizar cambios y modificaciones. Eso supone que la información está actualizada al instante y da una imagen precisa y real del proyecto.

Al tratarse de una herramienta orientada inicialmente a la gestión de tickets, cuando es necesaria la contribución de departamentos especiales en una empresa, la integración es muy simple.

Una prueba de la difusión de JIRA es la existencia de herramientas y extensiones que la complementan y ayudan a superar algunas limitaciones. Por ejemplo, JIRAclient[28] es un programa que facilita la edición masiva de entradas, algo complicado en JIRA y cuya interfaz es Web.

Además de JIRA, hay un buen número de herramientas para *Scrum*: pm*Scrum*, PlanBox, Pango *Scrum*, Agile Solutions (antes Rally)[29] y muchas otras. En cuanto a herramientas gratuitas, el catálogo es también bastante extenso: Kunagi[30], *Scrum*Do[31], *Scrum*y[32] o Banana *Scrum*[33].

De todas ellas, podemos destacar:

- Agilo for TRAC[34] (y su gemela Agilo for *Scrum*[35]), que en realidad es un plugin para *Scrum* de la herramienta de gestión de proyectos TRAC.Quizá lo más destacable es la capacidad de albergar información detallada sobre todos los aspectos del proyecto y la de enlazar los distintos elementos que definen el trabajo. Tiene perfiles específicos para los distintos roles *Scrum* y una amplia serie de informes. Como el medio de acceso es Web, se puede descargar e instalar localmente o acceder como servicio a la Web del fabricante.
- *Scrum*works[36] de Collabnet. Es una aplicación de escritorio pensada sobre todo para el uso de *Scrum* en entornos de desarrollo software con XP (véase el **capítulo 10** sobre el tema). También hay una versión Web más limitada. Cuenta con una completa herramienta de generación de informes que permite crear otros nuevos.
- *Scrum*Desk[37] es una herramienta directamente pensada para *Scrum* y que permite un uso gratuito para un número limitado de usuarios. Contiene la información de los distintos proyectos en servidores propios o compartidos y usa para acceder a ellos un cliente PC. Además de *Backlogs*, fichas y gráficas más convencionales, contiene gráficas exclusivas y algunas funcionalidades tan curiosas como una herramienta para hacer el *Planning Poker*, tal y como se puede ver en la siguiente figura:

Figura 6.9. La herramienta ScrumDesk.

- Mingle de ThoughtWorks[38]. Usa una interfaz que recuerda a un panel de tareas y maneja tarjetas para representar elementos. Permite trabajar con *Scrum* y XP, y además puede servir de herramienta para gestión de calidad. Cuenta con varias gráficas integradas y la posibilidad de crear otras nuevas.

En resumen

En este capítulo, se ha visto cómo se desarrolla el trabajo en un proyecto *Scrum* a lo largo de un *Sprint* o iteración. Cómo, para mantener el ritmo, evitar desviaciones y mantener el flujo de información se organizan reuniones diarias con todo el equipo (*Scrum* diario, *Daily meeting* o *Daily Scrum*). Uno de los resultados más importantes de esas reuniones es la pila o *Backlog* de impedimentos para el proyecto y la identificación de responsables para su resolución.

También se ha visto cómo se organiza un equipo y consejos sobre la forma de trabajar y comunicarse.

Por último, se ha dedicado un apartado para hablar de las herramientas utilizadas en *Scrum* y otras metodologías ágiles: desde los sencillos y efectivos paneles con etiquetas a sofisticadas aplicaciones informáticas.

En lo que respecta a este libro, el *Sprint* ha terminado y hemos encontrado varios obstáculos en el desarrollo del trabajo planificado. Varios de esos obstáculos han podido solventarse, pero otros han impedido completar el ambicioso programa que habíamos previsto.

Ahora que ha finalizado el *Sprint* llega el momento de revisar qué hemos hecho y cómo lo hemos hecho para ver dónde podemos mejorar de cara al futuro.

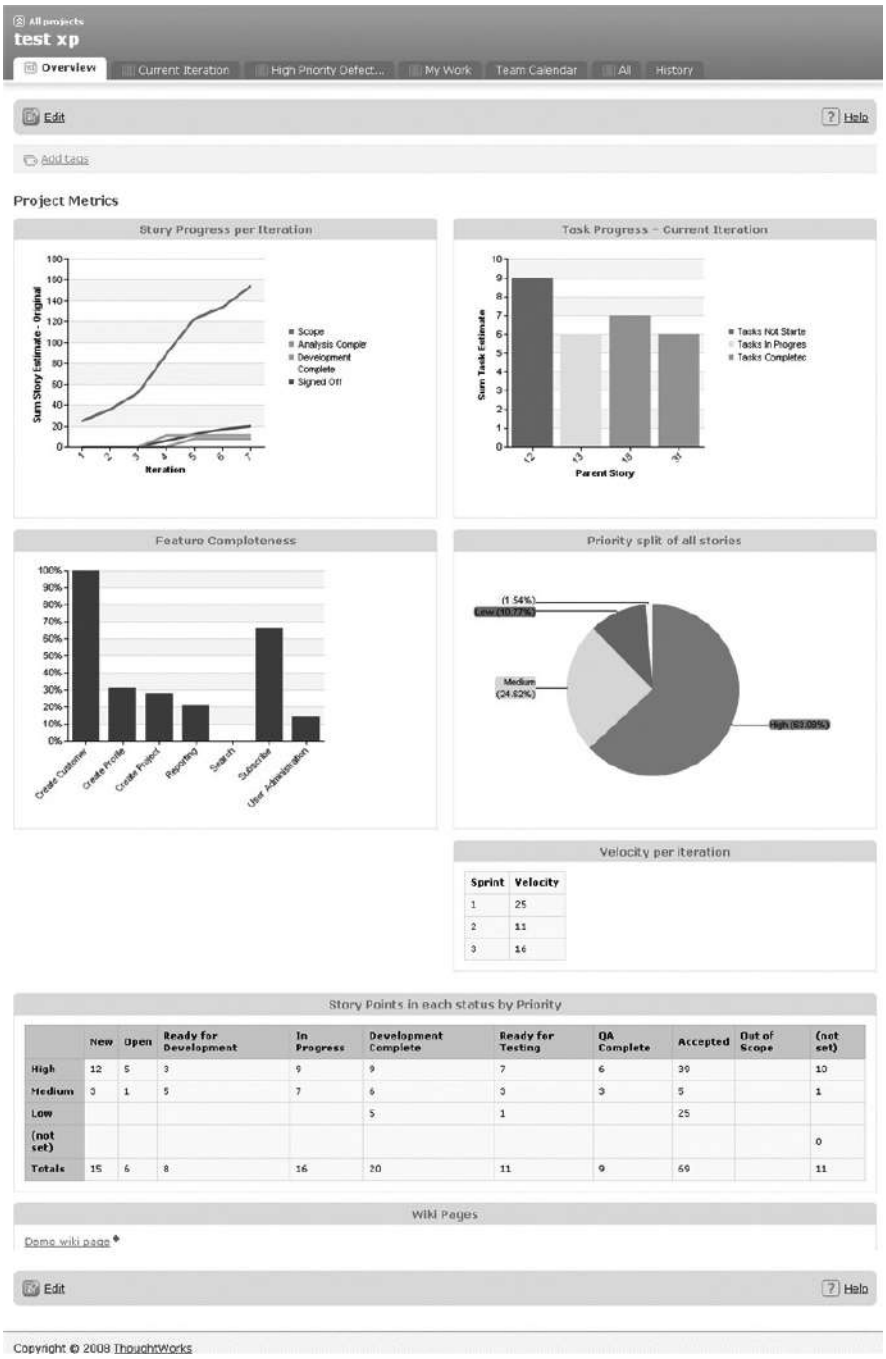


Figura 6.10. La herramienta Mingle.

---

25 Clientes, usuarios y, en general, todas las personas interesadas en el resultado del trabajo.

26 Hay una sencilla regla llamada las dos pizzas, que limita el tamaño del equipo al número de personas que puedan comer con dos pizzas.

27 http://www.atlassian.com/software/greenhopper/

28 http://almworks.com/jiraclient/

29 https://www.ca.com/us/products/agile-solutions.html/

30 http://kunagi.org/

31 http://www.scrumdo.com/

32 http://scrumy.com/

33 http://www.bananascrum.com/

34 http://www.agilofortrac.com/

35 http://www.agiloforscrum.com

36 http://www.collab.net/products/scrumworks/

37 http://www.scrumdesk.com/

38 https://www.thoughtworks.com/mingle/

7 ¿Vamos por buen camino? La Sprint Review

En este capítulo aprenderá:

- La definición de la *Sprint Review*.
- Objetivo de la *Sprint Review*.
- Problemas de la *Sprint Review*.

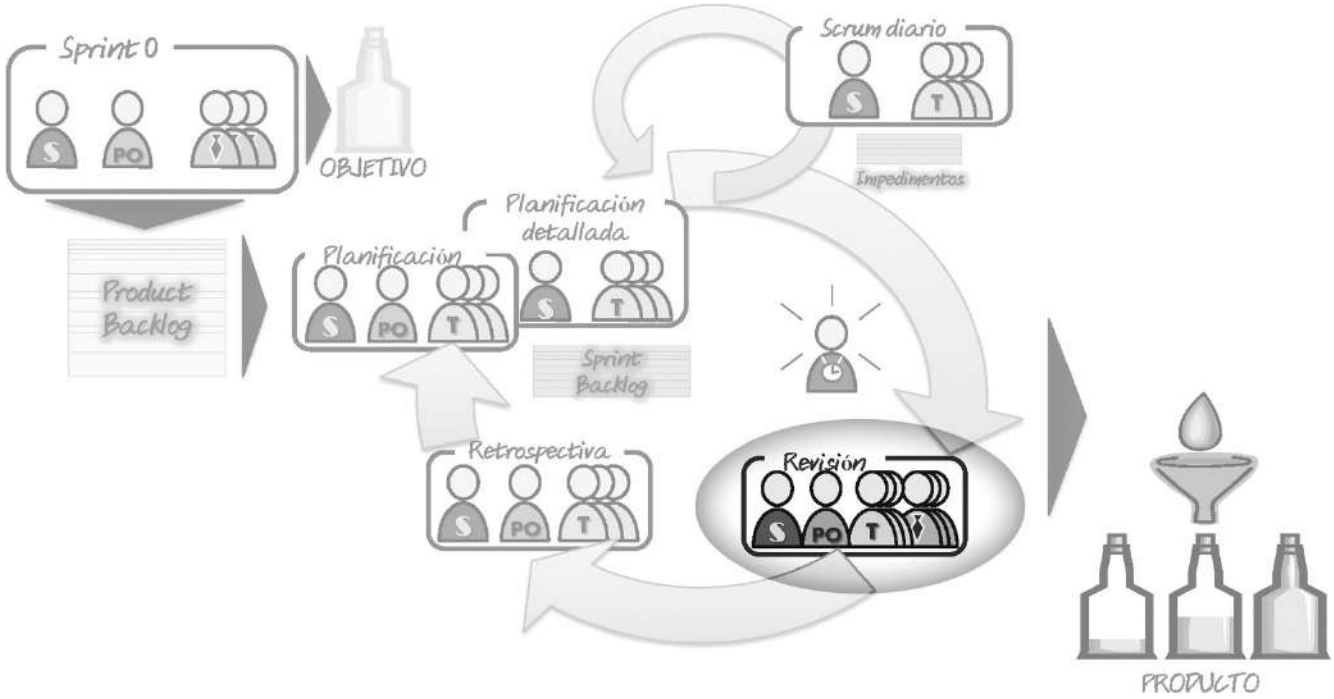


Figura 7.1. El desarrollo del Sprint en el ciclo Scrum.

Ahora que se ha cumplido el tiempo que nos hemos dado para nuestra iteración o *Sprint* llega el momento de verificar si el trabajo que hemos realizado se ajusta a nuestros planes y expectativas iniciales. En el caso de nuestro libro, nos propusimos un objetivo bastante ambicioso. A lo largo del *Sprint* han surgido y se han resuelto impedimentos, algunas tareas han supuesto menos esfuerzo del que preveíamos en un principio y otras se han complicado más de lo esperado inicialmente. Ahora ha llegado el momento de la revisión y veremos hasta qué punto hemos cumplido con nuestro compromiso y estimación.

Uno de los principios de *Scrum* es instrumentalizar el proceso de aprendizaje. En *Scrum* se elimina la suposición de que el cliente tiene claro lo que necesita o busca. Se considera que el equipo va a tener que construir con cierto grado de incertidumbre y, por lo tanto, se cuenta con la necesidad de aprendizaje y adaptación continua. Otra de las suposiciones que se elimina es que el equipo sabe cómo trabajar en común. Es necesario también un aprendizaje continuo en la forma de trabajar como equipo para encontrar la estabilidad y productividad. Para conseguir este objetivo, dentro del método, se propone la realización de una reunión denominada *Sprint Review* como resultado natural de cada iteración, como se explica en el capítulo 6.

La *Sprint Review* es el punto de comunión entre los responsables de un producto o proyecto y el equipo de desarrollo. Por medio de refuerzos positivos o negativos, los responsables del producto asimilan el cómo se está desarrollando su producto y el equipo que lo está desarrollando interioriza por qué y para qué se está desarrollando. Todo lo aprendido

debe incorporarse a las siguientes iteraciones de desarrollo o *Sprints* y así adaptar el proceso a las necesidades específicas del proyecto.

Nota:

Jean Piaget definió que la adaptación está formada por dos movimientos: el de asimilación y el de acomodación. Estos dos movimientos se retroalimentan y se pueden repetir cíclicamente para facilitar la adaptación. Scrum es un ejemplo de proceso de adaptación y aprendizaje cíclico.

Un rápido vistazo a la Sprint Review

La *Sprint Review,* también denominada *Customer Sprint Review*, es una reunión definida en *Scrum* que tiene lugar el último día del *Sprint.* El objetivo principal de esta reunión es la recogida de información o *feedback* sobre el estado del proyecto o producto en desarrollo. Para ello, en esta reunión, se realiza una puesta en común de todo el trabajo realizado durante el *Sprint.* Esta revisión sirve para poder repasar y discutir sobre los puntos clave de la novedad generada durante la última iteración. En otras palabras, analizar en el valor añadido al producto o proyecto.

Esta reunión tiene una duración que es dependiente de la propia duración del *Sprint* que concluye, ya que, a más duración del *Sprint,* más trabajo hay que mostrar. Una posible regla de ayuda es reservar de media a una hora por cada semana de *Sprint*. El aspecto más importante es que esta reunión debe estar acotada en tiempo antes de comenzarla y no se debe permitir que extienda indefinidamente. El concepto de limitar el tiempo en *Scrum* para evitar que las reuniones se alarguen innecesariamente se denomina *Time boxing* y es uno de sus valores básicos.

Es tan importante revisar el qué se ha hecho durante un *Sprint* como el cómo se ha hecho para maximizar la adaptación durante el siguiente *Sprint*. Considerar la revisión de una iteración solo como la revisión formal de lo que se ha hecho, sin tener en cuenta el cómo se ha hecho durante una Retrospectiva, es uno de los errores más comunes que se pueden encontrar en la aplicación de *Scrum*.

La finalidad fundamental de la *Review* es revisar el qué y se puede resumir en tres puntos:

- Recoger comentarios referentes al objetivo del *Sprint*.
- Aceptar las tareas que se seleccionaron al inicio del *Sprint* como trabajo operativo.
- Analizar si se necesitan mejoras en el trabajo realizado hasta la fecha o añadir nuevos ítems en el *Product Backlog* para cubrir necesidades emergentes en el producto o proyecto.

Preparación

La *Sprint Review* no tiene que convertirse en una reunión excesivamente formal. De hecho, se considera un encuentro informal, aunque tampoco se debe improvisar demasiado y hay que realizar ciertos preparativos.

Cuando un *Sprint* comienza, la logística de su *Sprint Review* debería estar prácticamente preparada: la fecha, hora y lugar deben estar acordados y deben ser conocidos por los asistentes a esta reunión. En algunos casos, si así lo decide el equipo, estos datos se hacen públicos para que los interesados en asistir a esta reunión tengan oportunidad de hacerlo.

Nota:

Es útil tener una plantilla para una Wiki o para un cartel y poder rellenarla y compartir con todo el equipo rápidamente. Con esta práctica, se disemina la información del proyecto en la organización en la que se está trabajando y se minimizan las interferencias en las reuniones.

Figura 7.2. Tiempo para la Review.

Al iniciar un *Sprint*, el *cómo* se va a demostrar que se ha completado cada ítem del *Backlog* incluido en la iteración debería estar claro. A la hora de discutir cada ítem y estimarlo en el *Sprint Planning*, se ha tenido que acordar esta forma de demostrar el elemento o el *how to demo*. Los *how to demo* son una forma de visualizar los criterios de aceptación que se definen para un elemento del *Backlog*, tal y como se ha descrito en el capítulo 4 sobre el *Product Backlog*.

La preparación de la *Review* no debería ocupar más de 1 o 2 horas. Esta preparación dependerá de la duración de la *Review* y en ningún caso debe ser una distracción para el equipo. A la revisión de final de *Sprint* se debe llegar sin ningún esfuerzo adicional o traumático. En esta preparación, se debería:

- Asegurar que todo el material necesario para la demostración estará disponible.
- Crear un guion coherente para la demostración del trabajo realizado según todas las demostraciones parciales de los elementos del *Backlog*.
- Si se va a hacer una demostración práctica, verificar que todo funciona correctamente.

Nota:

Preparar la Review no implica tener en mente simplemente la demostración que se va a realizar. Hay que tener en cuenta que la Review va más allá de una demostración o presentación y se deben analizar todas las implicaciones que tiene. Una buena recomendación es plantearla como un focus group, que es una técnica de recolección de datos para grupos, con el fin de obtener información acerca de la opinión de los usuarios sobre un tema en particular.

Cuando se prepara el guion de la presentación de resultados es muy importante enfatizar el objetivo del *Sprint* y su consecución. El objetivo de un *Sprint* nunca debe ser simplemente la suma de ítems del *Backlog* terminados. No hay que olvidar que el objetivo del *Sprint* es llegar a la meta marcada en la planificación del *Sprint* por encima de la culminación de más o menos elementos del *Backlog*.

Los invitados

De todas las reuniones definidas en *Scrum*, la *Sprint Review* es la que admite más invitados. Lógicamente, es necesario que asista el equipo de desarrollo del producto, el *Scrum Master* y el *Product Owner*, pero otros roles, como usuarios, gestores y miembros de otros proyectos, son igualmente bienvenidos. A todos estos roles fuera del equipo *Scrum* se les denomina *stakeholders*.

En general, cualquier persona interesada en el proyecto debería poder asistir. Como uno de los objetivos de esta reunión es recoger comentarios e información del producto para la siguiente iteración, parece lógico que se abra el abanico de invitados a todas las personas que puedan aportar información para el producto.

Obviamente, dependiendo del número de invitados, la logística y dinámica de la reunión pueden verse afectadas. Se deberá sopesar el valor que aporte la asistencia de invitados a la

reunión frente a las trabas que puedan plantear al desarrollo de esta.

Los peligros escondidos

Como en todos los artefactos y medios definidos en *Scrum*, en la *Sprint Review* pueden aparecer una serie de problemas que pueden afectar tanto a la propia reunión como al proceso global y, en ocasiones, pueden ser mucho más nocivos de lo que aparentan a simple vista.

En los siguientes apartados se presenta una relación de posibles problemas que se pueden encontrar al realizar este tipo de reuniones.

Nota:

Esta lista puede crecer en función de las peculiaridades de la organización en la que se implante Scrum, ya que muchos de los problemas son causados por elementos de la organización ajenos al equipo de trabajo.

La reunión móvil

Al tratarse de una reunión a la que pueden asistir muchas personas, algunas de ellas con un cierto nivel de responsabilidad en la organización, es habitual que existan problemas con las agendas de los asistentes. Se está tentado de modificar el día de la *Sprint Review* cuando hay problemas de asistencia y, por lo tanto, variar la duración del *Sprint* sobre la marcha. Este cambio tira por tierra el compromiso adquirido en la planificación del *Sprint* por parte del equipo, ya sea porque se reduce el tiempo para realizar el mismo trabajo o porque se amplia.

Nota:

El compromiso que adquieren los miembros del equipo cuando se realiza la planificación del Sprint es uno de los pilares de Scrum y el fundamento de muchos principios que facilitan el desarrollo de la iteración, por ejemplo, la autogestión del equipo.

Aunque se cuente con mayor tiempo, la incertidumbre de fechas a las que se somete al equipo hace que su compromiso pueda flaquear. Se puede abrir la puerta a plantearse por qué van a esforzarse para tener un trabajo en una fecha si posiblemente esta acabe cambiándose.

Es importante no cambiar, por lo tanto, las fechas del final de un *Sprint.* Es preferible la ausencia de algún invitado a la reunión de fin de *Sprint* que el cambio de fecha.

La mejor contramedida contra este problema es fijar con mucha antelación las fechas de

las *Review* y buscar el compromiso de todas las partes para respetarlas al máximo.

Figura 7.3. Reunión convocada con anterioridad suficiente para que todos los invitados a la Review puedan asistir.

El Sprint Review como una demostración o presentación de resultados al estilo metodología en cascada

Aunque frecuentemente los *Sprint Reviews* se ven como "la demo del fin de *Sprint*", su propósito va más allá de una simple demostración o presentación de resultados. Como se ha comentado anteriormente, estas reuniones se deberían enfocar como un *focus group* o reunión de captura de opiniones para recoger comentarios e información sobre el *Sprint* realizado, la funcionalidad obtenida y las adaptaciones necesarias que hay que aplicar en el *Backlog*. Por otro lado, debe tomarse como un mecanismo de aprendizaje sobre las implicaciones que las funcionalidades que se solicitan tienen sobre el producto. Esto quiere decir que la reunión sirve a los responsables de producto para relacionar el valor que se ha añadido al producto con el coste que ha implicado.

Tener una agenda de la *Review* preparada con distintas secciones, en las que la demo o presentación de resultados es una de ellas, es una buena práctica para desacoplar el concepto

demo del de *Review*.

Trabajar para la Sprint Review

Muchas veces, cuando el equipo de desarrollo de un producto o proyecto se inicia en la dinámica de los *Sprint*, se centra demasiado en la iteración en la que se está trabajando. Se crea un falso sentimiento de necesidad de trabajar para una demostración que satisfaga a los clientes o al *Product Owner*. Se pierde la visión del producto global que se está desarrollando e incluso del objetivo del propio *Sprint*. Esto hace que muchas veces se decidan tomar atajos o buscar soluciones cortoplacistas para que se pueda cumplir el objetivo de la demo.

Nota:

El Scrum Master tiene que ayudar al equipo a no perder el rumbo durante el Sprint, transmitiendo correctamente la visión del producto y el objetivo que el Product Owner persigue.

Se tiene que evitar confundir los principios YAGNI (*You Aren't Gonna Need It* o "No lo vas a necesitar"), que nos hablan de crear solo lo que se necesita en el momento actual, con desarrollar para la demostración en vez de para el producto. La labor del equipo trabajando en un producto o proyecto es crearlo de forma incremental y periódicamente se muestra el estado del desarrollo en la *Sprint Review*. No se trabaja para las demostraciones o presentaciones de resultados, un conjunto de funcionalidades para contentar a los actores implicados en el proyecto o producto y luego adaptar dicha funcionalidad al producto creado.

Nota:

Cuando un desarrollador del equipo plantea hacer algo con la justificación de mejorar la demostración, es necesario analizar si estamos añadiendo valor al producto o simple y exclusivamente a la demo.

Preparación excesiva de la Review

En línea con el anterior punto, cuando se trabaja solo para poder realizar una buena demostración, se dedica un tiempo excesivo para preparar la reunión. El tiempo efectivo del *Sprint* se ve reducido, empeorando la velocidad del equipo y, por lo tanto, los tiempos de desarrollo del producto.

Nota:

Como se ha mencionado antes, la Sprint Review debe ser un resultado natural de un Sprint y no se debería dedicar más de 1 o 2 horas a su preparación.

No se debe confundir el tiempo de preparación de una *Review* con el tiempo dedicado a garantizar la calidad del producto o proyecto. La garantía de la calidad forma parte la creación del producto o del desarrollo del proyecto y hay que dedicar todo el tiempo necesario.

La *Review* no es una presentación, no es una exposición, no es una conferencia, por lo que no es necesario preparar ningún material de apoyo como, por ejemplo, presentaciones en Power Point. Lo único que se tiene que presentar es el trabajo en una versión *enseñable*, tangible o ejecutable (cuando por ejemplo se esté hablando de software).

Foco en los ítems del Backlog y no en los objetivos

Al comienzo de un *Sprint*, se elige la cantidad de trabajo que un equipo puede asumir en la siguiente iteración. Teniendo en cuenta los ítems que van a entrar en el *Sprint,* se define una meta que servirá de mapa de ruta para guiar al equipo. Además, dará la flexibilidad necesaria para realizar las correcciones que se tengan que hacer dentro de la iteración, en caso de que surjan problemas para cumplir el compromiso inicial del equipo.

Esa meta y objetivo son los que ayudan al equipo a mantenerse unido en un objetivo común y evitar la mera ejecución de un listado de acciones a realizar. Es la guía que les ayuda a trabajar realmente como un equipo. Si no se presta atención en cubrir ese objetivo para el fin de *Sprint* y no se le da la relevancia suficiente en la *Sprint Review,* se perderá el hilo conductor que une los elementos del *Sprint Backlog* y dificultará la extracción de información relevante para adaptar la ejecución de posteriores *Sprints*.

Desviación del propósito de la reunión por los asistentes

El propósito y objetivos de la reunión deben estar claros para todos los asistentes. Se puede ser flexible a la hora de cubrir los objetivos de la reunión, pero se debe evitar que la reunión la monopolicen los *stakeholders*, gestores o usuarios. Su aportación tiene un espacio perfectamente delimitado en la reunión y, por eso, se debe reconducir a su cauce, siempre que se pueda, si se desvía de su propósito. Una buena práctica es comenzar haciendo un repaso del objetivo y planteamiento de la reunión para que los asistentes tengan claro cuál será la mecánica de esta.

Nota:

Si surge algún tema relevante para el producto, pero que no aplica a la reunión de Sprint Review, siempre se puede crear una reunión específica para tratar y resolver ese tema de forma dedicada.

Convertir la Review en una reunión de aprobación de requisitos

La *Sprint Review* no es una reunión de aprobación de los ítems del *Backlog*. Los ítems del *Backlog* deben llegar a esta reunión cerrados o no cerrados por el equipo, ya que se han definido unos criterios de aceptación que delimitaban su realización. Si han surgido dudas durante la realización del *Sprint,* se deberían haber resuelto con el *Product Owner* es ese momento. Si al revisar la realización de un ítem del *Backlog,* se comprueba que este no cubre las necesidades actuales del producto, se identificará esta nueva necesidad y se incorporará al *Backlog* como una mejora sobre un ítem del *Backlog* o incluso se creará un nuevo ítem.

Si un ítem del *Backlog* no cumple los criterios de aceptación, el equipo de calidad, en el caso de que lo hubiera, debería impedir que se diera por cerrado. Con esta premisa, en la reunión, no se acepta o rechaza un ítem del *Backlog*. Más bien se recoge información sobre este para tomar una decisión en el siguiente *Sprint*. Se debe evitar que, repetidas veces, un ítem del *Backlog* se traslade abierto al siguiente *Sprint* porque no se apruebe su realización. Si esto ocurre, los criterios de aceptación no se están fijando correctamente y se deben revisar.

Rechazo a las opiniones

Es importante evitar que la reunión se convierta en una discusión alrededor de alguna opinión proporcionada por uno de los asistentes a la reunión, decidiendo si se incluye o no en el *Backlog*. Para evitar entrar en ciclos de rechazo-venganza o desmotivar a los asistentes a que proporcionen sus opiniones, todo comentario debe ser bien recibido e incluido en el *Backlog*. Si no es relevante en el momento actual del proyecto, se le asignará una prioridad baja y se revisará en futuras reuniones de trabajo sobre el *Backlog*.

Duración excesiva de la Review

Aunque se respete la regla de una hora de reunión por cada semana de trabajo, muchas veces la duración de este tipo de reuniones crece incontroladamente. Es importante fijar y transmitir la duración de la reunión de forma anticipada a los asistentes para que se respeten

al máximo. El concepto de *TimeBoxing* o limitación del tiempo es muy importante en *Scrum*.

Para evitar que las demostraciones de los elementos del *Backlog* se alarguen demasiado, es conveniente seguir la definición de cómo hacer la demo que se haya definido en la preparación del *Sprint* y no intentar cubrir todos los casos de uso del ítem del *Backlog*. Se puede entregar el incremento del producto o la información generada para la *Sprint Review* a los asistentes y *Product Owner* después de la reunión para que experimenten o contrasten el resultado del *Sprint* de forma completa.

Para evitar que se abran discusiones interminables sobre el producto en sí, es una buena práctica que el *Product Owner* trabaje en la estrategia del producto o proyecto de forma constante. Una buena práctica es reunirse con los implicados en las decisiones del producto o proyecto durante el *Sprint* y cerrar allí estas discusiones para que no se traten solamente en la reunión *Review*. De esta forma, se puede enfocar la discusión en el avance del proyecto y no sobre la estrategia de este.

Nota:

Una de las reuniones que se deben mantener durante un Sprint es la denominada Refinamiento del Backlog, que ayuda a preparar el Backlog para nuevas iteraciones y genera discusiones sobre la estrategia del producto.

Un guion es la mejor ayuda

Tener un guion preparado y respetarlo en las *Review* ayuda a que todo el mundo interiorice la mecánica de la reunión de forma rápida, mejorando la realización de esta. El siguiente guion es una propuesta que resume las distintas actividades que podrían realizarse en la *Sprint Review* para sacarle el máximo partido:

- Establecer las normas y reglas de conducta en la reunión si hay nuevos asistentes que no hayan estado con anterioridad.
- Repaso de los objetivos del *Sprint.*
- Recapitulación del *Product Owner* identificando qué cosas se han terminado y cuáles no. Aquí el concepto “terminado” define lo que el equipo considera listo según los criterios de aceptación que se definieran para cada elemento del *Backlog*.
- Revisión por el equipo de sus estadísticas y métricas, repaso de los logros del producto, los problemas que han aparecido en la realización de los ítems del *Backlog* y cómo se han tomado medidas para resolverlos. Aquí se habla desde el punto de vista del producto. En el siguiente capítulo se hablará de cómo se analizan los resultados desde el punto de vista del equipo con la Retrospectiva.

- Demostración por equipo de los ítems considerados como terminados, respondiendo a todas las dudas que puedan surgir en referencia a ellos.
- Repaso por parte del *Product Owner* del *Product Backlog* tal y como estaba antes de la reunión y el plan de entregas preestablecido.

Nota:

Cada equipo, en función del ámbito en el que esté creado el producto o proyecto y según los parámetros de calidad que quieran aportar, define un conjunto de métricas que se pueden revisar en la Review para analizar su estado y evolución. Un ejemplo podrían ser los datos de cobertura de las pruebas en un desarrollo software, como indica la siguiente imagen.

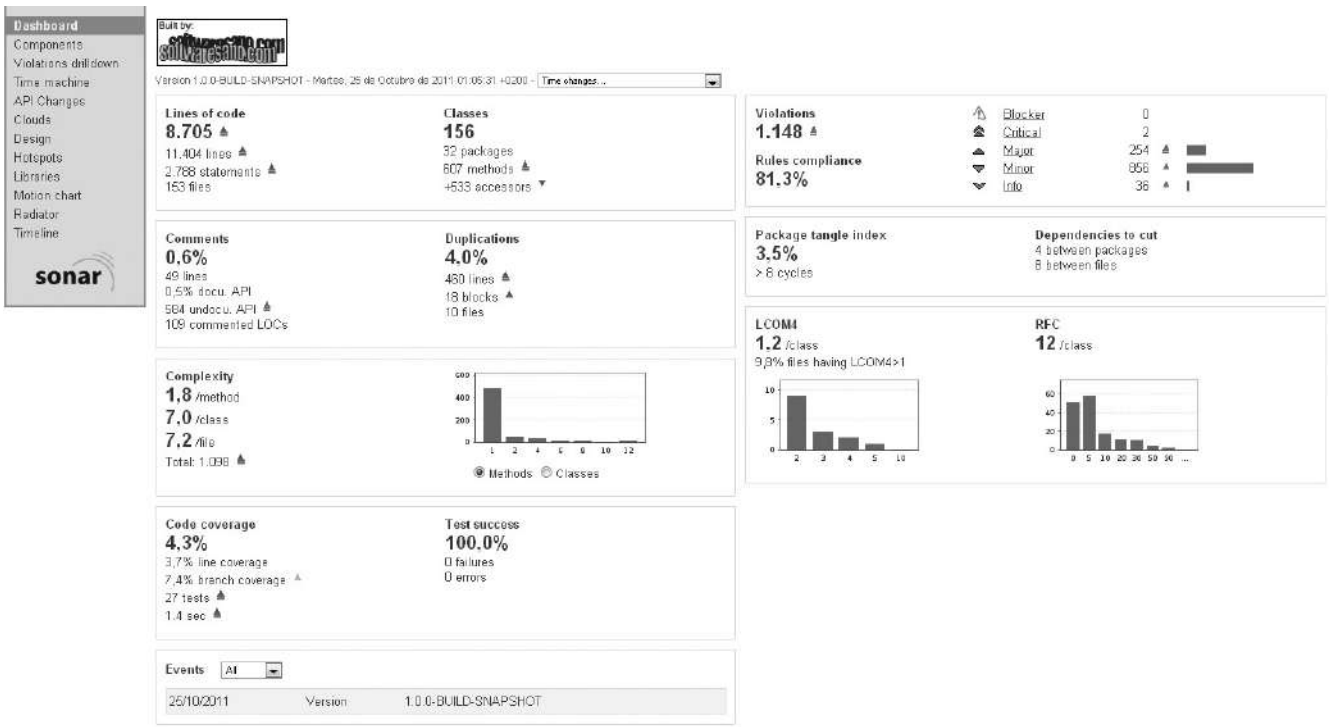


Figura 7.4. Métrica de cobertura en Sonar sobre Hudson.

- Comentarios de los *stakeholders* y *Product Owner* respecto a todas las novedades referentes al producto que puedan afectar al desarrollo de este.
- Recogida de todos los comentarios generados en la reunión, incorporándolos de forma apropiada al *Product Backlog* para tener el material necesario para las siguientes reuniones de estimación y planificación.

¿Qué se espera tener al final de una Review?

El resultado de esta reunión siempre debe ser el mismo, tiene que ser un *Product Backlog* adaptado a las nuevas necesidades del producto que se hayan detectado en la reunión de *Review*.

También se tendrá un plan de entregas actualizado con la nueva velocidad del equipo y el cambio de prioridades.

Si después de la reunión de *Review,* no ha habido ninguna modificación en el *Product Backlog*, puede considerar la reunión como un fracaso. Se deberán analizar detenidamente las causas por las que no se ha generado ninguna realimentación para el producto e intentar resolverlas durante la siguiente iteración.

La revisión que realizamos sobre la iteración de nuestro libro ha sido en general satisfactoria. Buena parte de las historias que habíamos incorporado al *Sprint Backlog* se han completado. Sin embargo, ha habido que rechazar una, ya que no se había completado, lo que supone que pasará al próximo *Sprint* únicamente con las tareas pendientes, y los puntos de historia que habíamos estimado no se podrán contabilizar. Lo mismo ocurre con otras dos que no se han podido iniciar: vuelven al *Product Backlog* para volver a ser priorizadas e incluidas en un próximo *Sprint*. En nuestra *Review*, también se ha detectado la necesidad de añadir un nuevo capítulo para completar el contenido de este libro. Y así lo hemos añadido a nuestro *Product Backlog*.

8 ¿Cómo lo hemos hecho? La Retrospectiva

En este capítulo aprenderá a:

- Descubrir por qué son necesarias las Retrospectivas.
- Identificar las diferentes etapas de una Retrospectiva.
- Aplicar diferentes técnicas para llevar a cabo una Retrospectiva.
- Mejorar las Retrospectivas.
- Decidir si necesita reservar un tiempo de descanso entre *Sprints*.

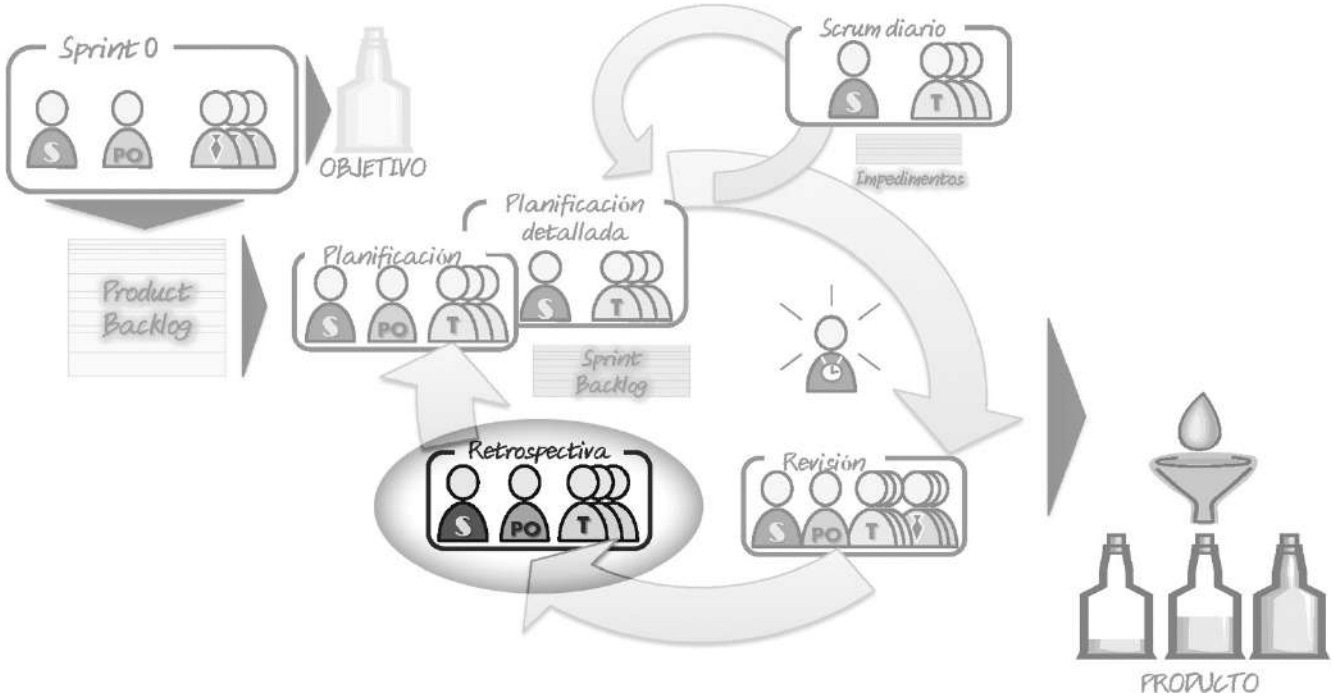


Figura 8.1. La Retrospectiva en el ciclo Scrum.

Hemos pasado ya un *Sprint* completo desde que lo iniciamos con la planificación, realizamos el trabajo revisando periódicamente su evolución en nuestros *Daily meetings* y verificamos si los resultados alcanzados eran los previstos en la revisión. Sin embargo, no lo podemos dar por cerrado sin analizar cómo lo hemos hecho. *Scrum* persigue la mejora continua y eso supone pararse a pensar si estamos haciendo las cosas de la mejor manera posible.

En el caso del proyecto para escribir nuestro libro, arrancamos un *Sprint* con una planificación bastante ambiciosa: parecía viable completar tres capítulos, lo que incluía revisión e imágenes. La historia de los anteriores *Sprints* nos hacía parecer confiados. El caso es que los resultados no fueron los esperados: solo uno de los capítulos pudo considerarse terminado, hubo problemas con el flujo de la información entre los autores y la primera experiencia con el banco de imágenes no fue la esperada. ¿Qué podemos hacer?

Durante una *Retrospective* o reunión de Retrospectiva, todo el equipo se reúne para revisar el pasado, sea reciente o no, y poder organizar la forma de trabajar más efectiva en el futuro. Con la *Review* analizamos el resultado de nuestro esfuerzo y la Retrospectiva nos ayuda a mejorar la forma en la que trabajamos. Es decir, buscamos la mejora constante tanto en lo que hacemos como en cómo lo estamos haciendo.

Al terminar cada iteración y antes de comenzar con la planificación de la siguiente es un gran momento para que el equipo analice en común cómo está trabajando.

El objetivo de una Retrospectiva es aprender de la experiencia para mejorar

constantemente la forma en la que se está construyendo el producto y así trabajar *Sprint* tras *Sprint* de manera más efectiva y agradable.

En las Retrospectivas se detecta todo aquello que no es útil al proyecto para eliminarlo o modificarlo. También sirven para potenciar y maximizar lo que es útil para mejorar el trabajo. Este mecanismo de búsqueda de la mejora constante aumenta la calidad de lo que se construye. Por otro lado, la Retrospectiva es una oportunidad extraordinaria para revisar los posibles riesgos del proyecto y para mejorar la comunicación y la relación entre las personas del equipo. En la Retrospectiva, el *Scrum Master* anima al equipo a revisar cómo fue el último *Sprint* teniendo en cuenta todos los aspectos: cada persona individualmente, las relaciones entre ellas, los procesos seguidos y las herramientas utilizadas. En el curso de la Retrospectiva, el equipo debe identificar y priorizar los aspectos que funcionan correctamente y aquellos que ayudarán a hacer las cosas mejor.

Una Retrospectiva eficaz debe concluir en acciones concretas de mejora que deberán implementarse en un corto plazo de tiempo.

En una Retrospectiva deben tratarse con el mismo interés tanto los aspectos operativos como los más personales, ya que ambos afectan directamente a la productividad del equipo.

Por qué necesitamos las Retrospectivas

Un cuento popular explica la historia de un leñador que fue a pedir trabajo en un aserradero. El sueldo era bueno y las condiciones laborales mejores aún; así que el leñador se propuso hacer un buen trabajo.

El primer día se presentó al capataz y este le dio un hacha y le asignó una zona del bosque. El hombre, entusiasmado, salió al bosque a talar. En un solo día cortó dieciocho árboles.

—Te felicito, sigue así —dijo el capataz.

Animado por estas palabras, el leñador se decidió a mejorar su propio trabajo al día siguiente.

A la mañana siguiente se levantó antes que nadie y se fue al bosque. A pesar de todo su empeño, no consiguió cortar más de quince árboles.

—Debo estar cansado —pensó. Y decidió acostarse con la puesta del sol.

Al amanecer se levantó decidido a batir su marca de dieciocho árboles. Sin embargo, ese día no llegó ni a la mitad. Al día siguiente fueron siete, luego cinco, y el último día estuvo toda la tarde tratando de talar su segundo árbol.

Inquieto por lo que diría el capataz, el leñador fue a contarle lo que le estaba pasando y a jurarle y perjurarle que se estaba esforzando hasta los límites del desfallecimiento. El capataz le preguntó:

—¿Cuándo afilaste tu hacha por última vez?

—¿Afilar? —respondió el leñador—. No he tenido tiempo para afilar. He estado demasiado ocupado talando árboles.

Lo más importante de las Retrospectivas es asegurarse de que tienen lugar. Parece una afirmación algo absurda pero no lo es en absoluto.

Todos los equipos coinciden, sin dudarlo, en lo útil que es tratar de mejorar constantemente la forma en que se trabaja y poner en común los problemas y preocupaciones para buscar entre todos posibles soluciones. Sin embargo, también es muy habitual que los equipos tengan tendencia a saltarse la Retrospectiva y a empezar directamente con el siguiente *Sprint* después de la *Review* y que necesiten que alguien les dé un empujoncito para que se realicen estas reuniones. Ese alguien puede ser el *Scrum Master*, un *coach* o cualquier miembro del equipo que haya descubierto la utilidad de las Retrospectivas con otros equipos.

Debemos evitar que nos pase como al leñador y que el "día a día" no nos deje revisar nuestra forma de trabajar.

Los equipos deben mejorar constantemente la productividad y la calidad del producto que están creando. Asimismo, es vital que revise con frecuencia qué es lo que no les deja disfrutar en su trabajo diario. Es importante tener en cuenta que, si un equipo no se reúne periódicamente para abordar estos temas y revisar cómo está trabajando, los problemas se repetirán, iteración tras iteración.

Las Retrospectivas son la clave para mejorar el trabajo en equipo a partir de reconocer cómo se están haciendo las cosas. Si algo no funciona del todo bien, revíselo y mejórelo.

Es obvio que una idea para mejorar nuestro trabajo puede surgir en cualquier momento y en cualquier lugar, pero las probabilidades de que esa idea sea aceptada aumentan notablemente si se plantea en la Retrospectiva y se permite a todo el equipo discutir y enriquecer esa idea. En definitiva, si la idea final es fruto de las aportaciones de todo el equipo y todos la sienten como suya el grado de implicación será mayor.

Por otro lado, es bueno recordar que una visión global de todo el equipo nos permite descubrir detalles que una persona sola no podría ver.

Las Retrospectivas nos ayudan enormemente a compartir diferentes puntos de vista y a descubrir que "mis enfoques y planteamientos" no tienen por qué ser los mismos que los que tiene otra persona del equipo. Esta puesta en común nos ayuda a ver las cosas desde diferentes ángulos y a no dar nada por supuesto. Cuanto más se dialogue dentro de un equipo, más se avanzará, siempre y cuando estos diálogos conduzcan a algo productivo, lógicamente.

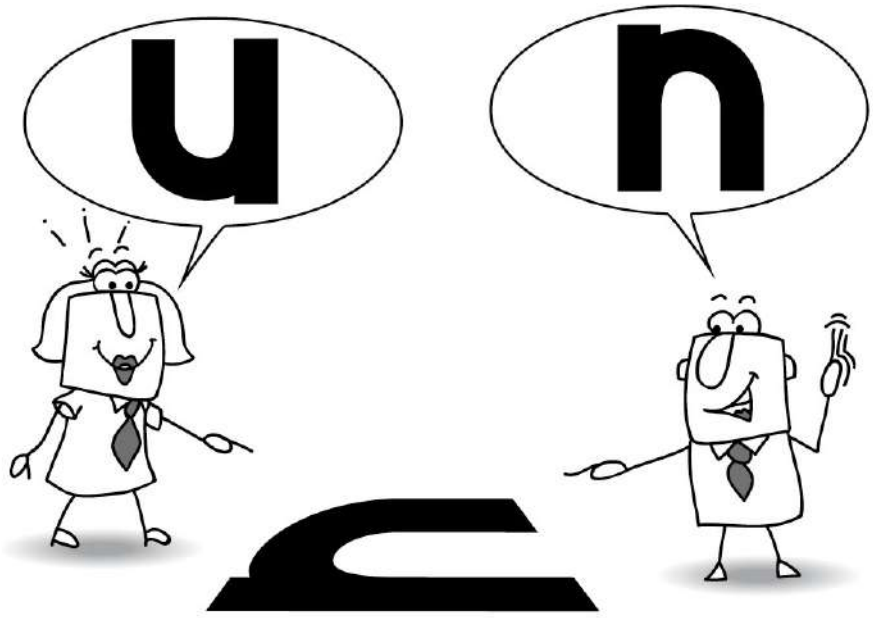


Figura 8.2. Diferentes puntos de vista.

Nota:

Es muy positivo ser conscientes de que las cosas pueden cambiar más de lo que pensamos inicialmente y no conformarnos con frases como "esto siempre se ha hecho así". Si no se intenta hacer nada, nada cambiará. ¡No empecemos una Retrospectiva sabiendo de antemano cómo va a acabar!

Se pueden realizar Retrospectivas en distintos momentos del proyecto: al final del *Sprint*, después de una entrega intermedia o *release* o al final del proyecto.

Las Retrospectivas son tremendamente útiles siempre, se usen o no métodos ágiles. Si no se utiliza *Scrum*, se pueden adaptar las Retrospectivas para revisar y adaptar la forma en que se está trabajando en el curso del proyecto y hacer ese análisis una vez al mes o cada vez que se cumpla un hito significativo.

Observación:

Sería injusto pensar que las Retrospectivas solo son necesarias y útiles para los equipos que tienen problemas. Una Retrospectiva ayuda a mejorar a todos los equipos y esto incluye a los que ya lo hacen bien y no tienen demasiados problemas. Todo puede ser mejorado.

Antes de empezar una Retrospectiva

Una vez que tenemos clara la utilidad de las Retrospectivas y que estamos decididos a llevarla a cabo, hay una serie de consideraciones que debemos tener en cuenta para que sea un éxito.

Algunas recomendaciones

La primera de ellas es que para abordar una Retrospectiva ante todo debemos ir con una mentalidad positiva. ¿Qué significa esto? Pues que, aunque haya problemas, seguro que algo se hizo bien y es crucial recordarlo. Hay que dedicar una parte de la Retrospectiva a comentar los éxitos y las buenas prácticas utilizadas, así como recordar cualquier aspecto positivo para el equipo que haya ocurrido a lo largo del *Sprint*. Si nos embarcamos en una Retrospectiva, nuestra intención real debe ser tratar de mejorar las cosas, pero, si comenzamos con espíritu destructivo, negativo o pesimista, francamente, poco conseguiremos. Por eso, viene muy bien dedicar la primera parte de la Retrospectiva a los aspectos positivos.

Pero, para que la Retrospectiva tenga sentido, es necesario hablar de los temas que molestan o inquietan al equipo por el motivo que sea. La práctica nos ha enseñado que es infinitamente más útil hablar de “los temas que hay que mejorar” que de “las cosas que no funcionan”. El primer enfoque da por hecho que es posible cambiar y hacer las cosas mejor y, sin embargo, con el segundo enfoque, es muy fácil caer en el derrotismo.

En definitiva, debemos ser positivos y evitar a toda costa ser tremendistas y derrotistas.

Si realmente queremos ser constructivos, al comenzar una Retrospectiva debemos partir de la premisa de que todo el mundo ha hecho el mejor trabajo posible teniendo en cuenta su capacidad, conocimientos y los recursos disponibles. Sin embargo, no debemos engañarnos: en ocasiones, dentro de los equipos puede haber personas que no tengan el perfil adecuado, sean poco productivas o directamente sean conflictivas. Obviamente, esto es un problema para la productividad del equipo y hay que solucionarlo. Este asunto debe tratarse con las personas adecuadas dentro de la organización o con el departamento de recursos humanos, pero desde luego no en el curso de una Retrospectiva. El objetivo último de la reunión debe ser siempre buscar la forma de mejorar cómo trabajamos como grupo y no resolver cuestiones que atañen solo a algunas personas.

Consejo:

Las Retrospectivas no deben ser utilizadas si nuestro objetivo es poner en evidencia públicamente que alguien no está haciendo bien su trabajo o solucionar un problema que venga arrastrándose desde hace tiempo entre varias personas. Estos temas deberán tratarse en otro momento.

Está claro que la Retrospectiva es una reunión clave y, por ello, merece ser preparada

cuidadosamente antes de comenzar.

Aunque trabajemos juntos cada día, antes de una Retrospectiva es conveniente fijarse en los detalles cotidianos del entorno de trabajo de los demás: puestos de trabajo, corchos, mesas, *Backlogs*... Toda esta información tan familiar, mirada con detenimiento, es una fuente muy valiosa de información.

El objetivo general de toda Retrospectiva es aprender, mejorar y adaptar. Pero debemos concretar un poco más y pensar en objetivos más específicos. Por ello, es importante tener claro por qué vamos a invertir un tiempo en esta reunión y qué beneficios pretendemos obtener a raíz de ella. La Retrospectiva debe ser una reunión que merezca la pena mantener.

Algunos ejemplos de estos objetivos más específicos podrían ser: mejorar la relación entre algunos miembros del equipo, aumentar la productividad, mejorar prácticas de programación, buscar la forma de optimizar otras reuniones de *Scrum* o descubrir el origen real de algunos problemas del equipo.

Estos objetivos no deben ser restrictivos y deben dejar la puerta abierta al diálogo y a temas nuevos. También es necesario decidir qué personas deben asistir a la Retrospectiva e invitarles con antelación y decidir qué datos se deben aportar para que la reunión sea efectiva.

En definitiva, saber a qué nos vamos a enfrentar y con qué propósito. Por ello, lo ideal es que esta información se comparta con todos los asistentes antes de comenzar.

Importante:

Asistir a una Retrospectiva es voluntario. Nadie está obligado a participar en estas reuniones, pero, desde luego, si alguien no quiere ir a esta reunión, el SM deberá investigar los motivos de esa decisión, ya que esto esconde un problema más profundo de comunicación dentro del equipo, de colaboración o de algún tipo de presión.

Una pregunta habitual es ¿cuánto debe durar una Retrospectiva? y la respuesta es “depende”.

Una recomendación habitual es que la duración sea de media hora por cada semana de *Sprint*, pero, si el equipo identifica mejoras y soluciones antes de agotar este tiempo, no tiene sentido prolongarla. Si concluye este tiempo solo con mejoras superficiales y no se ha alcanzado el objetivo de la Retrospectiva, habrá que prolongarla o incluso replantearla por completo, ya que el objetivo de mejora no se cumple. Cuando un equipo hace periódicamente Retrospectivas, aprenderá a optimizarlas y serán más eficaces. También la duración debe depender del tipo de Retrospectiva que estemos realizando. No debería durar lo mismo una Retrospectiva de fin de *Sprint* que la de un hito mayor como un fin de *release*.

El moderador de la Retrospectiva

Pocas personas, si hay alguna, disfrutan con reuniones en las que se pierde el tiempo en discusiones que no conducen a nada y en las que sus opiniones, ideas o sugerencias caen en el vacío más absoluto.

Las Retrospectivas tienen que ser reuniones prácticas y útiles. De no ser así, pierden su razón de ser.

El moderador de la Retrospectiva debe ayudar al grupo a analizar datos, a compartir ideas y a obtener conclusiones útiles, evitando en todo momento discusiones innecesarias y la pérdida de tiempo.

El rol de moderador pueden desempeñarlo indistintamente el *Scrum Master*, un *coach* o un miembro del equipo, aunque lo más habitual es que sea el *Scrum Master* la persona que modere estas reuniones.

Sea quien sea la persona que esté desempeñando este rol, las siguientes recomendaciones le ayudarán a obtener mejores resultados de la reunión:

- Un moderador debe ser paciente y bueno escuchando.
- Cuando escriba en las tarjetas o *post-it* lo que dice una persona, debe escribirlo tal cual lo dice y no hacer interpretaciones. Si hubiera que resumir una idea, este resumen lo deberá hacer la misma persona que la tuvo.
- Debe recordar que su misión es facilitar la dinámica de la reunión y no debe participar en discusiones. En el caso en que tuviera que unirse a la conversación como participante, deberá delegar el rol de moderador a otra persona (temporal o definitivamente).
- Fomentará que todo el mundo participe activamente invitando a hablar a los más tímidos o retraídos y moderando activamente a aquellos que tiendan a monopolizar la reunión.
- El moderador debe evitar que se produzcan conflictos entre personas durante la Retrospectiva.
- Debe fomentar el diálogo productivo.

Etapas de una Retrospectiva

Ya entendemos los objetivos y lo que se quiere y no conseguir con una Retrospectiva: estamos listos, pero ¿y ahora qué hacemos?

El primer paso es, ante todo, no olvidar que de forma muy resumida el objetivo final de una Retrospectiva es obtener de manera clara y concreta la siguiente información:

- Qué es lo que estamos haciendo bien (para celebrarlo y continuar trabajando de esta forma).

- Qué otras cosas tenemos que mejorar o incluso en ocasiones dejar de hacer.
- Qué vamos a hacer en la siguiente iteración teniendo en cuenta la información de los dos puntos anteriores.

Para conseguir esta información, es muy eficaz seguir unas etapas[39], de forma que podamos obtener los datos realmente útiles y de una manera ordenada. Estas etapas pretenden básicamente establecer las bases, recopilar datos, buscar el por qué de las cosas, establecer un plan de acción y cerrar la Retrospectiva. A continuación, se trata un poco más en detalle cada una de ellas.

Establecer las bases

El tiempo es oro y es muy interesante para los participantes en la Retrospectiva saber, antes de llevarla a cabo, cuánto tiempo van a emplear en ella y cómo se va a utilizar. Tal y como se comentó antes, es necesario invertir tiempo para realizar una Retrospectiva, pero cuidando siempre que este tiempo se emplee correctamente. En definitiva, tener la tranquilidad de que esta inversión va a ser útil.

El moderador será el encargado de recordar a los convocados la duración de la Retrospectiva y de comentar con el equipo qué dinámica se va a emplear en esta ocasión para facilitar que surjan útiles acciones de mejora.

A continuación, una buena pauta para romper el hielo y fomentar la participación de todos, es invitar a que cada miembro del equipo haga una brevísima introducción, no más de una frase corta y hasta una palabra podría servir, en la que se exponga al resto de los asistentes qué es lo que se espera obtener de la Retrospectiva. Esto nos ayuda a todos a conocer el estado de ánimo del resto del equipo y a compartir las expectativas.

Por último, hay una excelente recomendación útil para cualquier reunión y que consiste en marcar unos acuerdos de actuación durante la misma. ¿Qué es esto exactamente? Pues decidir entre todos cómo actuar durante la reunión. Por ejemplo, acordar si se responde a llamadas del teléfono móvil o si usamos exclusivamente los ordenadores que sean imprescindibles para el desarrollo de la reunión. También es un excelente momento para decidir cómo gestionar el turno de palabra, las interrupciones y, sobre todo, recordar que debe primar el respeto hacia las opiniones de los demás.

Recopilación de datos

Para evitar divagar demasiado o especular sobre las posibles mejoras que hay que implementar en el siguiente *Sprint*, o más lejos en el futuro, es necesario crear entre todo el equipo una foto detallada y compartida de la situación actual. Recordemos algo que ya hemos comentado antes y es que una visión compartida por todo el equipo nos permite

descubrir detalles que una persona sola no podría ver.

Esta foto debe contemplar tanto datos objetivos como subjetivos, teniendo muy presente que los aspectos emocionales son tan importantes muchas veces como los aspectos objetivos o más técnicos.

Se puede comenzar poniendo sobre la mesa datos relevantes, concretos y objetivos que resuman el último periodo o *Sprint*. Para ello, escribiremos en *post-it* o tarjetas los datos que surjan a raíz del análisis de las métricas o de los acontecimientos ocurridos durante la iteración y los iremos pegando progresivamente en un lugar visible para todo el equipo.

Como métricas, se puede revisar la velocidad del equipo estimada frente a la realmente conseguida, el número de requisitos completados frente a los estimados, el gráfico del progreso del trabajo del equipo (*burndown chart*), la cantidad de defectos encontrados y corregidos, etc.

Como eventos, se deben recordar decisiones transmitidas en otras reuniones, incorporaciones, bajas o cambios en el equipo, hitos cumplidos, cambio en las máquinas de trabajo, cambio en la tecnología utilizada, celebraciones, percances y cualquier otro evento relevante que haya podido influir en el *Sprint*.

Nota:

Las métricas de un proyecto son una manera de documentar el proceso de mejora. El análisis sucesivo de las mismas permite hacer un seguimiento de la evolución de dicho proceso.

Otro tema que es necesario revisar son los resultados de la Retrospectiva anterior y comprobar el estado de las acciones de mejora que surgieron para verificar si los temas pendientes se han solucionado y actualizar su estado con el equipo. Si quedara algo pendiente, se podrían tomar decisiones de nuevo en esta Retrospectiva. Esta recopilación de datos estará incompleta si no se añaden los datos adicionales que vayan indicando espontáneamente todos los participantes y que reflejen los asuntos que nos han hecho sentir bien o, por el contrario, que han generado rechazo o malestar.

Recuerde:

Tan importante como detectar los problemas que tiene un equipo en su día a día es hacer un reconocimiento de las buenas prácticas del equipo para potenciarlas o mantenerlas.

Más adelante, en el apartado “Algunas prácticas para Retrospectivas”, veremos con detalle algunas prácticas para realizar esta recopilación de datos de manera útil y eficaz.

Buscar el porqué de las cosas

Ya tenemos todos los datos de lo que ha ocurrido y ahora es el momento de pensar en el porqué de las cosas.

Se trata de no quedarse en un análisis superficial de los éxitos o dificultades y mirar en profundidad qué ha pasado y qué es lo que lo ha ocasionado y, sobre esto, actuar, para que las acciones que se tomen sean realmente efectivas.

Es frecuente tomar rápidamente decisiones sobre lo que hay que hacer para solucionar un problema. En ocasiones, las decisiones pueden ser útiles y funcionar, pero puede darse el caso de que simplemente estemos "maquillando" un problema más profundo y que no se solucione así definitivamente.

Para ayudarnos a encontrar las causas, hay numerosas técnicas. A continuación, se explica cómo llevar a cabo dos de ellas: los cinco porqués y el diagrama causa-efecto.

- **Los cinco porqués:** Se agrupan los participantes por parejas o en pequeños grupos. Una persona pregunta a otra por qué ha ocurrido un problema o un evento y, cuando responda, volverá a preguntar por qué y así hasta cuatro o cinco veces. De esta forma se puede llegar a comprender la causa real y origen que causó un problema o inconveniente.
  Un ejemplo sencillo de esta práctica es el siguiente:

  P: ¿Por qué salió disgustado el *Product Owner* de la *Review*?
  R: Porque fue un desastre.
  P: ¿Por qué fue un desastre?
  R: Porque empezamos tarde y nerviosos.
  P: ¿Por qué empezasteis tarde y nerviosos?
  R: Porque no funcionaba el proyector de la sala con el que teníamos que hacer la demo.
  P: ¿Por qué no lo comprobasteis antes?
  R: Porque no pensamos que iba a fallar. Siempre funciona. Pero esto no nos vuelve a pasar. Para la próxima *Review* preparamos seguro el proyector con tiempo.

- **Diagrama causa-efecto o Ishikawa:** Este diagrama también se conoce como el diagrama de espina de pescado.
  Se dibuja una línea que representa el problema a modo de la espina dorsal del pez. Las causas que originan el problema desembocan en esta espina dorsal o en otras espinas y así va creciendo el diagrama hasta encontrar las causas raíz.
  Las espinas iniciales pueden ser las que respondan a las cinco preguntas que rodean a un acontecimiento: ¿qué ha pasado?, ¿dónde ha ocurrido?, ¿quién estaba implicado y afectado?, ¿por qué ha ocurrido? y, por último, ¿cuándo y en qué circunstancias ha

pasado? Pero, por supuesto, esto debe adaptarse en cada situación para encontrar las causas que mejor puedan ayudar a describir el problema que se esté tratando.

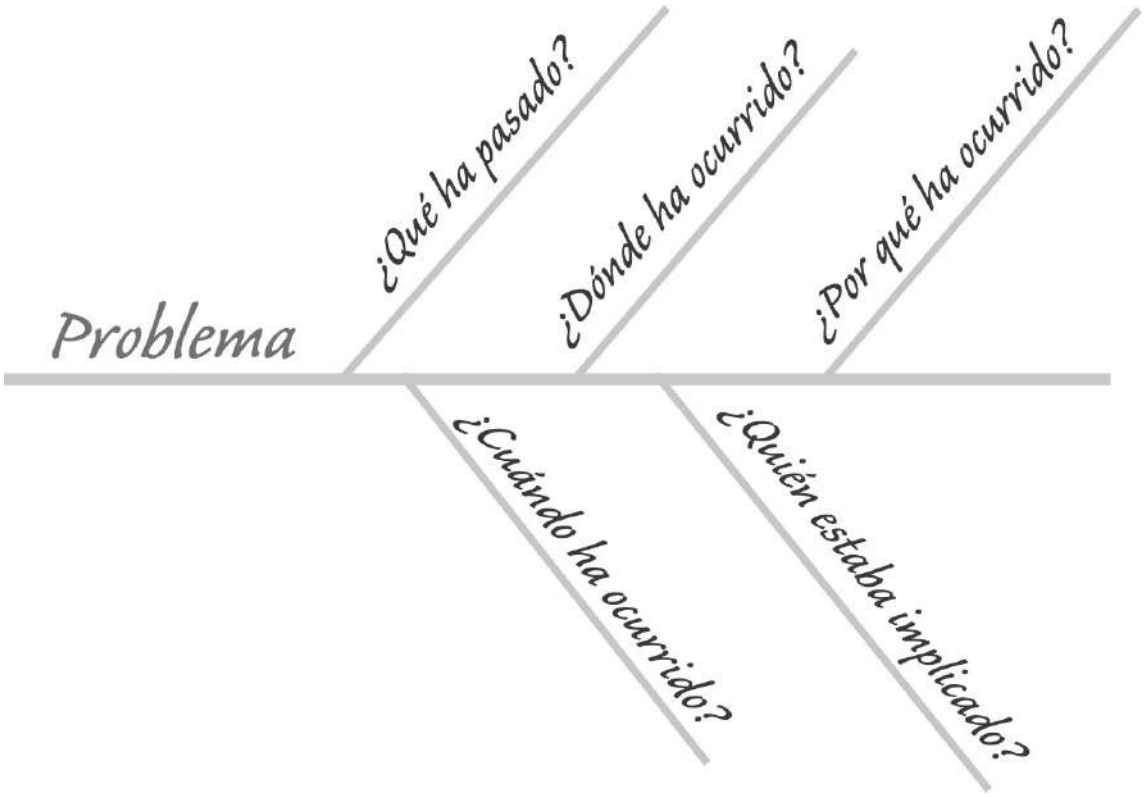


Figura 8.3. Diagrama causa-efecto.

Plan de acción

Ya tenemos identificados los problemas y las causas que los producen y, ahora, ¿qué hacemos?

De nada nos sirve lamentarnos si no vamos a hacer nada al respecto.

Lo lógico será pensar ya en las acciones que nos permitan solucionar, o al menos paliar, esos problemas. Es decir, decidir las acciones de mejora para nuestro proceso.

Es de nuevo el equipo el más adecuado para pensar y proponer soluciones a los problemas detectados por ellos mismos. Una forma eficaz de hacerlo es repasar cada uno de los problemas detectados y utilizar una “lluvia de ideas” para buscar entre todos la mejor solución.

Está claro que no siempre se pueden solucionar en el curso de un *Sprint* todos los problemas detectados, por lo que habrá que decidir qué problemas van a ser abordados y qué acciones de mejora se van a implementar.

Un primer filtro será tratar de poner solución a los problemas más dolorosos para el equipo, es decir, los más urgentes. Para hacer esta primera clasificación, podemos utilizar un sistema de votos y, para ello, podemos por ejemplo conceder 3 (o los que quiera: 5, 10...)

votos a cada asistente y él los asignará a los temas que necesitan ser abordados con más urgencia según su criterio. Podrá incluso, si lo considera oportuno, asignar sus tres puntos a un único asunto. Una vez finalizada la votación, se suman los puntos de cada entrada y ya tenemos una primera priorización de los problemas.

Otro criterio que se debe tener en cuenta es el esfuerzo que supone implementar una acción de mejora, su coste o el tiempo necesario para llevarla a cabo.

Y, por último, no debemos olvidar que es crucial tener siempre presente la satisfacción del cliente a la hora de decidir qué acciones tomar.

Para que todo este esfuerzo no quede solo en buenas intenciones, y una vez que ya está claro lo que tenemos que hacer y en qué orden hacerlo, es el momento de concretar. Es el momento de hablar de quién va a hacer las cosas y cuándo. Las acciones de mejora se pueden introducir en el *Sprint Backlog* (o en el *Product Backlog* si no son muy prioritarias) y podrán empezar a ser asignadas por los miembros del equipo.

Importante:

Hay problemas que se resuelven solo con ser detectados, por ejemplo, si en un equipo falta comunicación, si las reuniones diarias son largas y dispersas, si no están bien trabajados los criterios de aceptación de un requisito o si no se actualiza el Sprint Backlog a diario. Todos estos temas que, por el hecho de exponerlos, se consiguen mejorar casi sin tomar medidas adicionales. La comunicación ayuda a identificarlos y solventarlos.

Además, no hay que perder de vista una verdad que a veces olvidamos: como en todos los aspectos de la vida, hay problemas que tienen solución y otros que no la tienen.

En el primer caso, habrá que tratar de encontrar esa solución para aplicarla lo antes posible. Sin embargo, en el caso de los problemas que no tienen solución, debemos pensar en las acciones que nos ayudarán a disminuir, mitigar, el impacto del problema pero siendo conscientes de que este impedimento seguirá existiendo.

Nota:

Un equipo debe ser capaz de decidir soluciones reales que puedan implementar ellos mismos sin necesidad de permisos de sus superiores. Si los cambios los elige el equipo y no han sido impuestos, las personas se implicarán más en que se lleven a cabo.

Conclusiones finales con el equipo

Antes de la despedida, conviene hacer un brevísimo resumen de lo ocurrido durante la Retrospectiva que acaba de terminar y las conclusiones a las que han llegado entre todos.

También es útil dedicar unos minutos a comentar cómo ha ido la Retrospectiva, hacer una breve retrospectiva de la Retrospectiva para hacer estas reuniones cada vez más eficaces. No debemos olvidar nunca que la Retrospectiva es una reunión clave de *Scrum* y merece ser documentada para poder hacer un seguimiento posterior de las conclusiones tomadas y de si se están realizando las acciones comprometidas. También es necesario documentarlas para poder comprobar con datos si algún tema es recurrente y desde cuándo se ha identificado.

Por otro lado, en este documento también quedarán reflejados problemas o cuestiones detectadas por el equipo y para los que explícitamente se decidió no hacer nada, ya que no se consideró prioritario en su momento.

Tal y como ya hemos comentado, es una muy buena costumbre iniciar una Retrospectiva revisando las conclusiones de la Retrospectiva anterior.

Poco a poco, paso a paso, iteración tras iteración, iremos solucionando los principales impedimentos detectados y construiremos nuestro producto cada vez más eficazmente.

Algunas prácticas para Retrospectivas

Ya sabemos la teoría para cada una de las etapas que debe cubrir una Retrospectiva, pero ¿cómo ponemos esto en práctica?, ¿por dónde empiezo? En este apartado, se explican varias formas de cómo llevar a cabo una Retrospectiva desde un punto de vista puramente práctico. Se facilitan también algunas pautas para la recopilación de datos y para encontrar las acciones de mejora que correspondan. Las técnicas que vamos a comentar son: “Bien, Mejorable, Mejoras”, “Línea de tiempo”, “Estrella de mar”, “Triste, enfadado, contento”.

Pues vamos allá y decida, para cada equipo y para cada situación, cuál de las siguientes técnicas le puede ayudar más en su proyecto. Si lo prefiere, haga una combinación de varias de ellas.

Bien, Mejorable, Mejoras

Empecemos dibujando tres columnas de forma que en cada una de ellas podamos ir añadiendo tarjetas o *post-it*.

La primera columna representa lo que ha ido bien durante el *Sprint*, la segunda columna la utilizaremos para exponer lo que no ha ido tan bien y la última la reservamos para trabajar en las mejoras.

Una vez hecha la revisión de los datos objetivos ocurridos durante el anterior *Sprint*, por ejemplo, comparación de la velocidad estimada con la real, comentarios de la *Review*, revisión de las acciones de mejora comprometidas en el última Retrospectiva, etc., debemos empezar a recopilar más datos entre todos.

De forma muy resumida, se podría decir que estamos buscando respuestas concretas a las siguientes preguntas: ¿qué hemos hecho bien?, ¿qué podemos mejorar? y ¿qué mejoras vamos a introducir?

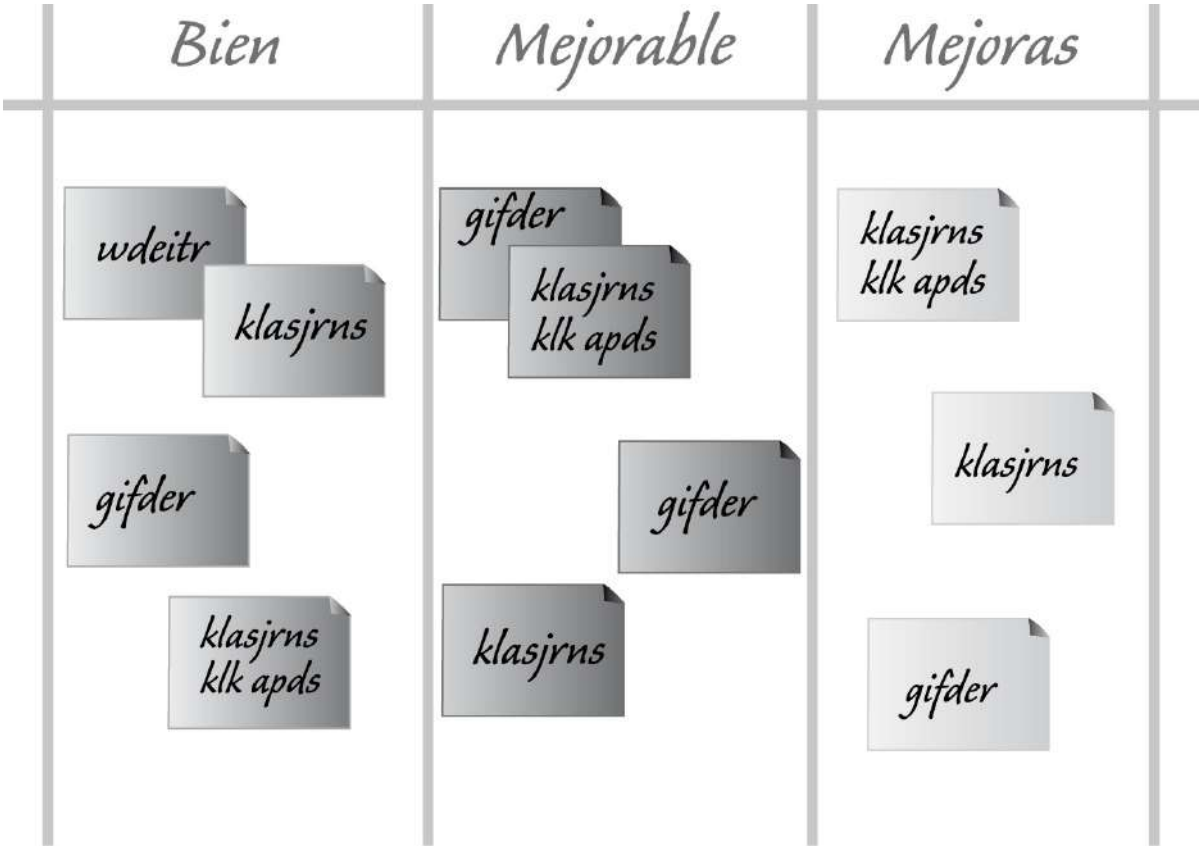


Figura 8.4. Tablón de Retrospectiva.

Ahora hay que empezar a rellenar estas columnas y lo haremos de la siguiente manera:

- Durante 5 minutos cada participante, de forma individual, pensará en todo aquello positivo que haya ocurrido durante el desarrollo del *Sprint*. Es el momento de recordar los buenos momentos, las alegrías, la toma de decisiones correctas, los éxitos, las buenas prácticas adoptadas e incluso felicitar a quien corresponda.
  Todos estos datos se escribirán en tarjetas o *post-it*, teniendo en cuenta que debe ir cada entrada en una tarjeta diferente para permitir la posterior agrupación por temas similares. Si escribimos un listado de cosas inconexas en una única tarjeta, la agrupación resultará imposible.
  Una vez transcurrido el tiempo dedicado a pensar, es el momento de compartir el resultado. Cada uno de los asistentes explicará al resto sus conclusiones de esta fase y aclarará las dudas o matices que le pregunten al respecto. En definitiva, es una puesta en común de los datos recolectados por cada uno. La forma de participar puede ser de manera aleatoria o siguiendo un orden establecido, pero siempre respetando el turno de

palabra y sin interrupciones innecesarias.

A medida que se exponen las ideas, el moderador o el mismo participante las van colocando en la primera columna.

El moderador debe evitar discursos largos. Hay que tratar de ser operativos, prácticos y evitar la dispersión. Asimismo, debe animar a todos los asistentes a participar activamente. Ya tenemos un listado de los temas que han ido bien durante el *Sprint* y, entre todos, debemos tratar de agruparlos por temas relacionados.

- La segunda columna la rellenaremos de forma similar a cómo se hizo la anterior, pero ahora se escribirán en las tarjetas o *post-it* los problemas que se hayan detectado durante el *Sprint*, aquellas cosas e incluso actitudes que impiden al equipo trabajar a gusto cada día, cualquier aspecto que sepamos que puede ser mejorado, etc. En definitiva, ahora es el momento de compartir con el equipo todo aquello que consideremos que debemos cambiar o mejorar en la forma en la que estamos trabajando.

  La manera de exponerlo y agruparlo será también igual a la de la anterior columna.

  Cuando los temas dolorosos para el equipo ya estén puestos en común, debe dedicarse algo de tiempo a comentar cada uno para, entre todos, buscar el origen o raíz de dicho inconveniente.

  Una vez tenemos agrupados los aspectos que hay que mejorar o cambiar, debemos priorizarlos para detectar entre todos en qué orden deben ser abordados en el siguiente *Sprint* y, para ello, podemos utilizar el sistema de votos por puntos, el que el equipo decida o el que el moderador sugiera.

  Es muy importante conservar aquí un tono positivo, o al menos evitar el negativo. En lugar de “aspectos negativos”, hablemos de “mejorables”. La idea de “mejora” es positiva, ayuda a levantar el ánimo del equipo y evitar buscar culpables de “los problemas” o de “lo malo” que nos está pasando.
- Ha llegado el momento de ser prácticos y hacer que la Retrospectiva sea útil. Hay que tratar de solucionar todo aquello que esté en nuestra mano y escalar o derivar los temas cuya solución no dependa directamente de nosotros.

  El moderador leerá el tema más prioritario de la columna central, es decir, el que necesita ser mejorado y que ha sido clasificado por el equipo como el más urgente y todos sugerirán posibles soluciones en forma de lluvia de ideas. El moderador las anotará y colocará en la columna de acciones de mejora aquellas acciones concretas propuestas por el equipo que ayudarán a solucionar, o al menos a disminuir el impedimento, problema o inconveniente detectado. En el caso de que se esté introduciendo alguna nueva práctica, la forma de implementarla se incluirá en esta columna. Así, una por una se va leyendo y trabajando cada aspecto susceptible de cambio o mejora de la columna central y se van aportando posibles soluciones en la columna de la derecha.

  Nuestro plan de acción estaría incompleto si no concretáramos aun más cómo llevarlo

a cabo. Para ello, podemos proceder de la misma manera que propone *Scrum* para la asignación del resto de tareas, es decir, el equipo se autoasignará las acciones concretas que se van a llevar a cabo durante el *Sprint* (o a más largo plazo si procede) y sugerirá una fecha prevista para su finalización o, al menos, una fecha en la que hacer una actualización del estado de la acción al resto del equipo.

En definitiva, si las acciones no acaban la Retrospectiva con dueño y fecha de realización, corren el riesgo de que todo esto acabe como un propósito de buenas intenciones.

Si el número de propuestas es muy grande, se puede seleccionar, por ejemplo, votando una cantidad que creamos posible realizar en el plazo del *Sprint*.

Recuerde documentar todo el proceso seguido durante la Retrospectiva de forma que quede registro de cómo se ha llegado a las conclusiones y porqué. Por ejemplo, tome fotografías de la pizarra en que está trabajando y amplíe la información gráfica con los comentarios que añadan valor.

Truco:

Es muy efectivo si cada participante se levanta para presentar y explicar sus notas al resto del equipo y es él mismo quien las coloca en la columna que corresponda. Es una forma sencilla de implicar a todos los asistentes en la Retrospectiva y evitar que se pierda el foco o la concentración en la reunión.

Línea de tiempo

Otra forma de conducir una Retrospectiva es basándonos en la secuencia temporal de los hechos. La forma de proceder es la siguiente:

Dibuje una larga línea horizontal en un lugar visible y accesible para todo el equipo.

¿Qué significa esta línea? Pues representa nada más y nada menos que la guía de nuestro *Sprint*.

Al inicio de la línea escriba la fecha en la que comenzó el *Sprint* y el final de línea representará el momento actual. El objetivo es ayudarnos a recordar y a poner en común todos los eventos significativos que han ocurrido desde que empezó el *Sprint*, o el periodo que se esté analizando, hasta el momento actual.

- En la parte superior de la línea, iremos proponiendo, entre todos, las cosas positivas que han ocurrido: las buenas prácticas, éxitos conseguidos, decisiones acertadas o cualquier cosa que nos recuerde un buen momento del *Sprint* y lo situaremos en el momento temporal en el que tuvo lugar.

En la parte inferior, representaremos la cruz de la misma moneda, es decir, recordaremos todo aquello que nos molestó, perturbó o disgustó durante el *Sprint*.

Las entradas pueden, así suele ser, aparecer simultáneamente arriba y abajo, es decir, se irá completando la línea de tiempo sin un orden determinado.

Lo importante de esta práctica es representar lo que ha ocurrido durante todo el *Sprint* y tener una visión compartida por el equipo.

Como en cualquier Retrospectiva, el moderador debe fomentar la participación de todos los asistentes de forma activa.

- El siguiente paso será hacer una selección de los temas que necesitan una solución más urgente y que aparecen en la parte inferior de la línea. Para ello, mediante un sistema de votos se hace una primera criba eligiendo los tres o cuatro temas más urgentes (por ejemplo, asignando un número de votos a cada participante, que reparten como consideren oportuno entre los ítems).
- Continuaremos haciendo el ejercicio de buscar las causas a estos problemas, así que uno por uno se analizarán en profundidad los temas seleccionados en el punto anterior y se anotarán las causas que los producen.
- Una vez que tenemos identificadas las causas que producen nuestros tres o cuatro problemas con más impacto, es el momento de pensar en acciones concretas para solucionarlos o, al menos, aliviarlos.

  De forma similar a la anterior, se pensará en las acciones de mejora asociadas a cada problema o impedimento y se le asignarán dueño y fecha de implementación.

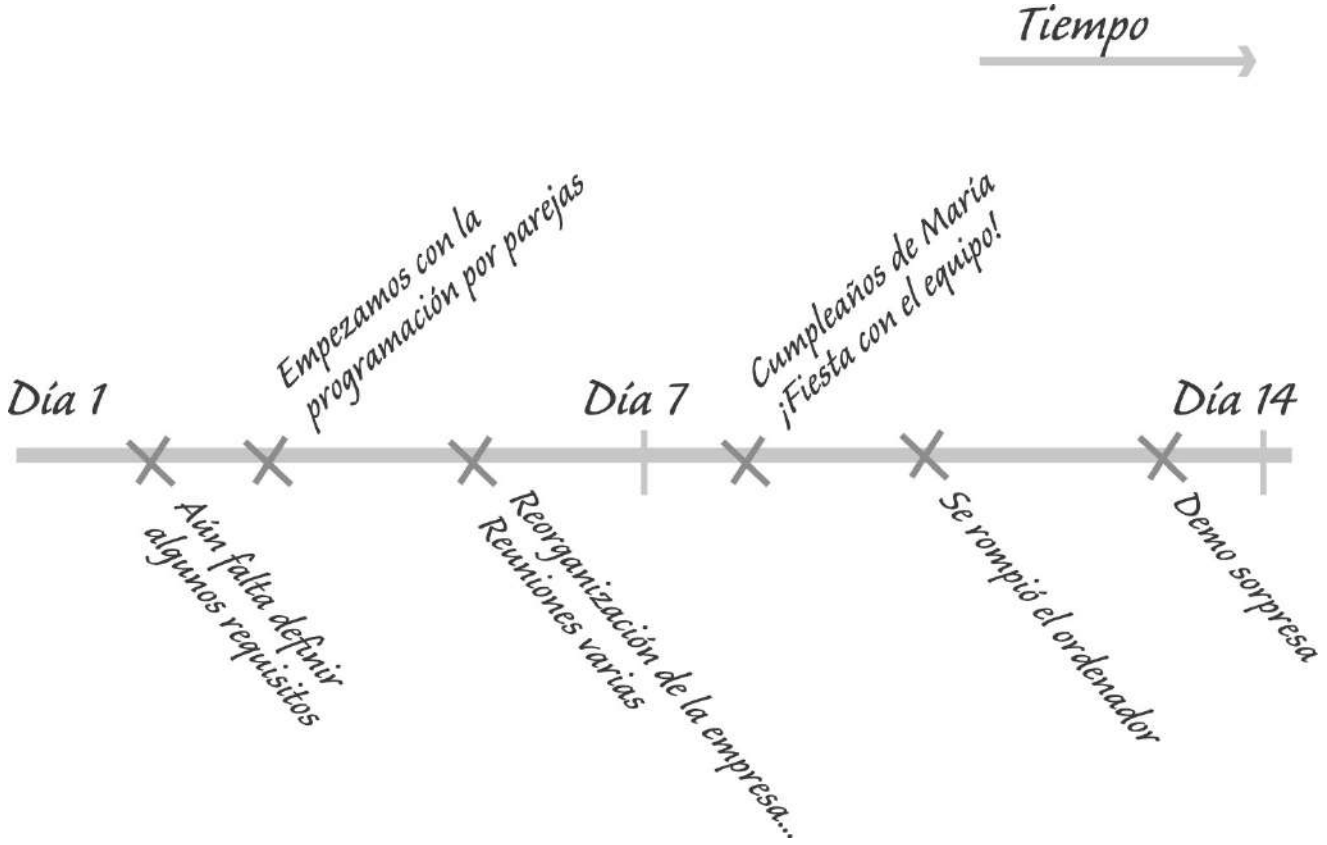


Figura 8.5. Retrospectiva con línea de tiempo.

Nota:

Esta técnica es muy útil para Retrospectivas que abarquen amplios espacios de tiempo, por ayudar a refrescar la memoria de los participantes, y no quedarse solo con los acontecimientos más recientes. Por ejemplo, puede serle útil para release o en el final de un proyecto.

Estrella de mar

No todas las cosas son blancas o negras. Hay momentos en los que meter una escala de grises puede sernos tremendamente útil para mejorar poco a poco, teniendo en cuenta todos los aspectos del proyecto: los roles, técnicas, prácticas, tecnologías, etc. Por este motivo, vamos a exponer otra técnica para hacer Retrospectivas que nos permitirá matizar algo más lo que fue bien durante el *Sprint* y lo que no fue tan bien y qué debemos hacer pensando en el siguiente *Sprint*.

Empiece su Retrospectiva dibujando un diagrama como el que se muestra en la siguiente figura, en el que vendrá representado un espacio para trabajar cada una de las siguientes categorías:

Figura 8. 6. Retrospectiva de estrella de mar.

- **Seguir haciendo:** En este apartado, debemos identificar todo aquello que echaríamos de menos si no estuviera o dejáramos de hacerlo. También es el lugar donde anotar aquellas prácticas que consideramos que debemos mantener porque aportan valor a nuestro trabajo. Un ejemplo puede ser invitar a miembros de otros equipos a las *Reviews* y que opinen sobre nuestro trabajo. Si lo estamos haciendo y nos va bien, ¿por qué dejar de hacerlo? Pero debemos cuidar el que se mantenga así.
- **Empezar a hacer:** Este es el lugar para plantear todo lo nuevo que queramos hacer, si queremos empezar a probar cosas nuevas para cambiar algo que no funciona o simplemente propuestas de cambio para darle un nuevo aire a la forma de trabajar que ya tenemos. En este caso, el ejemplo podría ser algo del tipo "¿Qué tal si subimos a la wiki del proyecto las actas de las reuniones para no tener que buscarlas cada vez que las necesitamos?".
- **Menos de:** ¿Qué hacer si hay algo que no aporta demasiado valor, pero sería una pena dejar de hacer radicalmente? Puede ocurrir que una práctica sea menos útil en las circunstancias actuales, pero lo fue en un pasado y podría volver a tener su valor. Por ejemplo, en un proyecto cuyo equipo estaba distribuido geográficamente, al principio del proyecto se tomó la decisión de reunirse presencialmente al menos tres veces al mes para discutir temas técnicos, ya que la distancia era un obstáculo para la comunicación. Pasado un tiempo, el equipo está consolidado y las reuniones por videoconferencia son muy eficaces. Eso sí, las reuniones presenciales para el *Planning* y Retrospectiva se mantienen, ya que el beneficio sigue siendo muy alto.
- **Más de:** Este apartado es el complementario del anterior. Aquí el planteamiento será "¿por qué no incrementamos esto que tan bien nos funciona aquí?". Es el momento de pensar en cómo perfeccionar las prácticas, técnicas, etc. que el equipo quiera intensificar porque no se les esté sacando todo el partido o no se estén aprovechando del todo. Por ejemplo, utilizar también la wiki para anotar cualquier asunto relacionado con el proyecto y no solo las actas de las reuniones como se venía haciendo hasta ahora.
- **Dejar de hacer:** Este último apartado nos servirá para limpiar nuestro día a día de todo aquello que, aunque tal vez en un pasado tuvo valor, ya no lo tiene. Lógicamente, aquí deberán reflejarse todas aquellas prácticas que se siguen realizando por rutina o costumbre, pero también es el lugar para proponer cambios radicales en el proyecto. Sería la otra cara de la moneda del apartado "empezar a hacer". Un ejemplo podría ser proponer el dejar de enviar un correo electrónico cada vez que queremos informar al resto del equipo de un error detectado en el código, ya que estos errores se reportan desde hace tiempo en la herramienta de seguimiento del proyecto.

Este tipo de Retrospectivas proporciona una información muy visual del estado del proyecto y es muy ilustrativo comparar cómo van evolucionando las estrellas obtenidas en las Retrospectivas a lo largo de un proyecto.

Triste, Enfadado, Contento

Esta técnica nos ayuda a buscar, desde un punto de vista emocional, las mejoras que debemos introducir en nuestro día a día.

Dibujaremos tres columnas para representar en la primera de ellas todo lo que nos produjo alegría, satisfacción, y que nos motivó durante el *Sprint*.

En la columna central, representaremos todo aquello que nos enfadó o frustró durante el *Sprint* y la última columna contemplará los aspectos que nos entristecieron y desmotivaron de alguna manera.

La forma de trabajar para completar las tres columnas puede ser similar a la descrita para la técnica “Bien, Mejorable, Mejoras”, explicada en el apartado correspondiente. Recuerde que inicialmente se reserva un espacio de tiempo para la reflexión individual y posteriormente se hace una puesta en común de las conclusiones individuales para llegar a acuerdos.

Serán los aspectos detectados en las columnas de enfado y tristeza los que el equipo deberá analizar para encontrar las causas que los produjeron y buscar la mejor solución.

Consejo:

En las Retrospectivas es mejor no ser demasiado ambicioso y concentrarse en unas pocas mejoras que sea posible implementar durante el Sprint.

Cómo mejorar las Retrospectivas

La excusa de falta de tiempo que algunos equipos utilizan para dejar de hacer Retrospectivas no tiene mucho sentido, ya que, tal y como hemos visto, los beneficios que nos brinda una Retrospectiva eficaz son altísimos. Aumentar la calidad del producto, mejorar las relaciones entre los miembros del equipo, detectar posibles riesgos en el proyecto y mejorar los procesos poco a poco bien merecen el empleo de unas horas al mes.

¿Cuánto tiempo se emplea charlando del proyecto durante el café?, ¿cuántas conversaciones cruzadas dentro de un equipo preguntándose unos a otros por qué se hace tal o cual cosa? o ¿cuántas charlas durante la comida opinando sobre mejoras que se quedan ahí, en el plato, y que no transcienden más? Obviamente, nadie duda del enorme valor que tiene que los equipos hablen, comenten, charlen. Faltaría más. Pero precisamente porque, para un buen equipo, cualquier momento es bueno para pensar en cómo mejorar la forma en que está trabajando, no arañemos tiempo precisamente del momento reservado en *Scrum* para aprender y mejorar de forma organizada, optimizada y productiva. En definitiva, no debemos quejarnos o lamentarnos por los pasillos de que algo no funciona del todo bien en mi

proyecto y luego o no hacer Retrospectivas o, si se realizan, no comentar estos temas en el momento y en el foro adecuado. No olvidemos que es responsabilidad de todo el equipo trabajar en la mejora continua.

No caigamos ni en el derrotismo de pensar que “total, ¿para qué?” ni en la soberbia de pensar que ya lo hacemos todo perfecto. Los mejores equipos son los que son capaces de mejorar constantemente.

Ahora bien, ¿qué pasa cuando realmente las Retrospectivas dejan de sernos útiles? ¿Qué podemos hacer cuando un equipo lleva 5 *Sprints* llegando a las mismas conclusiones en sus Retrospectivas y ha entrado en un bucle? ¿Y si el equipo tiene la sensación de que ya no tiene prácticamente nada que pueda mejorar?

La opción fácil es decidir dejar de hacer Retrospectivas. Pero esa no es la respuesta correcta.

Lo que debemos hacer es inyectar nueva energía a nuestras reuniones. Buscar la forma de llegar a otras conclusiones, de romper ese bucle en el que el equipo ha entrado. En definitiva, buscar la manera de reinventar las Retrospectivas con nuestro equipo para que vuelvan a resultarnos imprescindibles y útiles.

Sí, bien, pero ¿cómo lo podemos hacer? A continuación, proponemos algunas pautas para darle un nuevo aire a sus Retrospectivas:

- Salga de su entorno habitual. Es una forma de romper con la rutina. Cambie la sala donde se reúnen habitualmente para trabajar y realizar sus Retrospectivas, planificaciones y reuniones por otra sala que sea poco frecuentada por el equipo.
  A quién no le ha ocurrido alguna vez ir conduciendo al trabajo o a casa sin ser conscientes de por qué calles hemos pasado. No pensamos por dónde tenemos que ir, ya que lo hacemos casi como autómatas. Conocemos el camino de memoria y lo tenemos perfectamente interiorizado. Dónde girar. Dónde aparcar. Cambiar la sala e incluso el edificio para realizar nuestra Retrospectiva sería como cambiar la ruta para ir a casa. Descubriremos infinidad de detalles nuevos y, desde luego, prestaremos mucha más atención. Parece algo sin importancia, pero a muchos equipos les ayuda salir del lugar habitual de trabajo para cambiar de perspectiva a la hora de dirigirse al resto de su equipo. El cambio de sala suele ser útil para realizar una Retrospectiva en un momento especial del proyecto como pueda ser el cumplimiento de un hito o cuando sea necesario analizar un problema relevante o bien si se quisiera replantear la forma de trabajo de forma sustancial.
- Incluya dinámicas y actividades. El convertir una Retrospectiva en algo lúdico tiene varios beneficios. El inmediato es que dejan de ser aburridas en el caso de que así fuera. Eso sí, debemos no olvidar que no solo se trata de pasarlo bien. Se trata de buscar una forma amena pero productiva de encontrar la manera de mejorar en nuestro trabajo. El objetivo final de incluir dinámicas es tratar de que el equipo se relaje y que la gente hable con más soltura fomentando la participación de todos. A los tímidos les

costará menos hablar y los acaparadores tendrán que respetar su turno.

- Realice una Retrospectiva alternativa. ¿Qué hacemos si hay una persona que revienta una Retrospectiva constantemente? o ¿qué hacemos si hay alguien que incomoda al equipo de tal forma que si esa persona está presente no se habla con claridad y franqueza? En definitiva, hay que pensar qué hacemos si hay alguien cuya presencia no es deseada en la Retrospectiva. Una opción es realizar una Retrospectiva sin esa persona en concreto para poder hablar relajados y que el equipo no pierda la oportunidad de seguir mejorando. Eso sí, es necesario solucionar este problema en el foro correcto. El primer sitio debe ser en la Retrospectiva alternativa (sin la persona que incomoda), ya que uno de los temas que deberá tratarse en esta Retrospectiva es el por qué no se puede hablar cómodamente en su presencia. Con esta información, el *Scrum Master* debe trabajar en solucionar este problema, ya que muy probablemente no afecte solo a las Retrospectivas, sino a más aspectos del proyecto. ¡Ah!, y, obviamente, habrá que tratar el tema con la persona afectada.
- El anonimato. Puede darse el caso de que al equipo no le apetezca transmitir sus opiniones en público o decir abiertamente lo que piensa. Una forma para no perder esta información es trabajar en pequeños grupos y compartir las conclusiones de forma anónima. De esta manera, será la opinión del grupo la que se exponga y no la de una persona en concreto. Otra opción es que las propuestas, ideas y sugerencias se planteen de forma anónima.

  Todo esto no es más que solucionar de forma muy superficial un problema grave que tiene el equipo y el *Scrum Master* deberá trabajar en profundidad este tema con ellos ya que así se pone en evidencia o bien miedo a expresarse con franqueza; o bien una ausencia gravísima de confianza dentro del equipo.
- Trabaje en paralelo los temas. Si el grupo es muy numeroso, discutir y comentar entre todos cada tema podría desembocar en debates interminables y poco productivos. Si la Retrospectiva la están realizando muchas personas, es útil abordar temas por subgrupos y de esta forma no todos tienen que hablar de todo y así optimizará la reunión. De otra manera, podría ser interminable. Forme subgrupos y que ellos voluntariamente elijan los temas que quieran tratar y analizar. Una vez finalizado el tiempo reservado para el análisis, debe hacerse una puesta en común de todo el equipo para exponer las conclusiones.
- Cambie al moderador. Tal vez sea útil que no sea siempre el *Scrum Master* o el *coach* la persona que facilite la Retrospectiva. Una buena práctica es que los mismos miembros del equipo actúen de forma cíclica como facilitadores y, cuando sea su turno, propongan nuevas prácticas o dinámicas. Esta práctica puede revitalizar enormemente las Retrospectivas, ya que cada una de ellas tendrá un aire nuevo.
- Varíe la estructura de su Retrospectiva. Trate de aplicar las diferentes prácticas para realizar las Retrospectivas de forma que se rompa la rutina y surjan preguntas y enfoques nuevos. Una primera Retrospectiva podrá hacerse pensando qué hicimos

bien, no tan bien y qué podemos mejorar. Pero la siguiente Retrospectiva podemos empezarla dibujando la línea de tiempo que nos sirva de guía para hablar sobre lo que nos ha pasado y cómo nos hemos sentido a medida que avanzábamos en el *Sprint*. Utilice las técnicas propuestas en el apartado "Algunas prácticas para Retrospectivas", una combinación de ellas o cualquier otra que le resulte útil. O aplique algunas de las técnicas alternativas de Retromat[40], una colección pública de herramientas para dinamizar y variar la organización de retrospectivas.

- Cambie el objetivo. Revise las acciones de mejora acordadas en las últimas Retrospectivas y analice cuál está siendo el foco del equipo y amplíelo o modifíquelo. ¿Qué significa esto?, pues si últimamente el equipo está centrado únicamente en mejorar las prácticas de *Scrum*, tal vez sea el momento de analizar las prácticas de programación que se están siguiendo o el trabajo en equipo o la comunicación con otras áreas de la empresa, por ejemplo.
- Traiga invitados. Invite ocasionalmente (después del cumplimiento de hitos o momentos clave del proyecto) a personas que aumenten la perspectiva del equipo y les haga recordar que no trabajan solos y que su labor está relacionada con otros grupos a los que les afectan sus resultados y decisiones. Puede ser muy útil invitar a alguien que trabaje en la misma área de la empresa para que opine sobre el impacto de nuestro trabajo en la organización o animar a participar en las Retrospectivas a grupos transversales al del equipo, como, por ejemplo, alguien de marketing o mantenimiento, y que haga entender al equipo otros puntos de vista del producto en el que se está trabajando.
- Pida ayuda. Si nada de lo anterior ha funcionado y sus Retrospectivas siguen sin ser todo lo eficaces que desearía, pida ayuda. Busque una persona que le ayude a reenfocar las dinámicas y a liderar las reuniones. Es posible que un equipo no consiga romper con sus rutinas y que le cueste dar un giro a su forma de trabajar, pero tal vez simplemente contar con una ayuda externa es suficiente. Busque alguien que aporte un nuevo enfoque a los problemas habituales, nuevas ideas y, sobre todo, alguien que no esté condicionado ni por resultados anteriores del equipo ni por el día a día del trabajo en el proyecto.

Las Retrospectivas son el mecanismo para analizar y mejorar constantemente en nuestro trabajo. Si dejan de ser útiles, debemos revisar cómo las estamos realizando, buscar alternativas y adaptarlas a nuestras necesidades para que recuperen su utilidad.

Fin de la iteración y comienzo de la siguiente. ¿Periodo de descanso?

Bueno, ya hemos hecho una iteración completa, con su planificación, elaboración, *Review* y Retrospectiva... ¿Empezamos inmediatamente la siguiente iteración?

Hay autores que recomiendan parar uno o dos días entre iteraciones para realizar todas las tareas que, durante el curso de *Sprint*, no se pueden realizar por falta de tiempo o por no tener una alta prioridad en ese momento. Tareas como, por ejemplo, investigar nuevas herramientas, trabajar y enriquecer el *Product Backlog*, formarse y aprender alguna técnica, leer artículos, libros o *blogs* relacionados con el trabajo, etc. Este "descanso" entre *Sprint*s tiene sus pros y contras. Hay quienes prefieren no hacer paradas significativas entre iteraciones ya que se puede perder el ritmo de trabajo y la sincronización que el equipo hubiera adquirido. Sin embargo, los defensores de este periodo de descanso o *Improvement Period* argumentan que un equipo que siempre está "esprintando" verá disminuir su ritmo de trabajo antes o después.

En cualquier caso, es muy recomendable separar la Review y Retrospectiva de un *Sprint*, de la siguiente planificación, aunque sea unas horas. El *Product Owner* necesitará algo de tiempo para poder ajustar las prioridades en función del resultado de la *Review* anterior. El equipo debería analizar las conclusiones del final de *Sprint* antes de comenzar con el siguiente.

Hay equipos que ajustan sus *Sprint*s para que acaben en viernes y comiencen en lunes. El dejar un fin de semana entre *Sprint* y *Sprint* es una práctica que permite marcar una clara frontera temporal y da margen para asimilar la información obtenida tanto en la *Review* como en la Retrospectiva.

En resumen

En este capítulo, hemos visto por qué y cómo se realizan las reuniones de Retrospectiva al final de una iteración o *Sprint*.

Para muchos autores, la Retrospectiva es la reunión más importante y es la seña de identidad de *Scrum*. Su objetivo es favorecer la mejora continua de la forma en que se trabaja, encontrar soluciones a los problemas y preocupaciones que encuentra el equipo en su trabajo. Esa mejora continua se manifiesta, sobre todo, en incrementar la productividad del equipo y la calidad del producto. Por el contrario, un equipo que no mejore, encuentre problemas y no dedique un tiempo a exponerlos y encontrar solución está condenado a repetirlos.

Además, las Retrospectivas ayudan a compartir diferentes puntos de vista y a descubrir nuevos enfoques y planteamientos.

Como en otros momentos del ciclo de trabajo de *Scrum*, en la Retrospectiva, el *Scrum Master* actúa como facilitador animando al equipo a revisar cómo se desarrolló el último *Sprint*. En el curso de la Retrospectiva, el equipo debe identificar y priorizar los aspectos que

funcionan correctamente y aquellos otros susceptibles de mejorar para hacer las cosas mejor.

Hemos visto también varias técnicas para ayudar a aflorar esos aspectos positivos y mejorables de la forma de trabajar, aunque nada de esto tiene utilidad si no se acompaña de un compromiso de todos por poner los medios para mejorar los problemas encontrados y aplicar las soluciones sugeridas.

---

39 *Agile Retrospectives: Making Good Teams Great*, Esther Derby and Diana Larsen.

40 https://plans-for-retrospectives.com/es/

9 Scrum en acción: "Tu" Scrum

En este capítulo aprenderá:

- Qué es un *Scrum But*.
- Cómo detectar si se está haciendo un *Scrum But*.
- Ejemplos típicos de *Scrum Buts*.

Scrum es adaptación y aprendizaje. Es un viaje en el que siempre se puede aprender algo nuevo. *Scrum* marca las reglas del juego y, dentro de estas, va intrínseco un proceso de mejora continua. Se dice que, actualmente, hay empresas que están adoptando *Scrum* y otras que están adaptando *Scrum*. Como no es posible definir una metodología generalista que se pueda aplicar al 100% a todos los entornos, empresas y proyectos, *Scrum* no lo intenta. *Scrum* se suele definir más como un marco de trabajo que como una metodología. Es una herramienta que se puede usar para incrementar y mejorar la productividad, la calidad y la predictibilidad. *Scrum* no dice expresamente lo que hay que hacer, pero lo que sí permite es adaptar las prácticas al entorno en el que se aplique. Por esta razón, si se realiza una encuesta a todos los proyectos y equipos aplicando *Scrum*, se podrá comprobar cómo, en un alto número de ocasiones, su aplicación ha sido asimilada por cada organización y "mejorada".

¿Se puede cambiar Scrum?

Caveat emptor se traduce como "cuidado o mantenido por el comprador". Es una doctrina de la ley que indica que la responsabilidad de una compra recae sobre el comprador. Este se informará sobre lo adquirido evitando reclamaciones al vendedor. Una vez que lo ha adquirido, puede hacer lo que quiera con ello asumiendo las consecuencias. Con *Scrum* aparece una situación similar, el concepto está a disposición de la gente para que lo use y lo adapte si quiere, pero siendo consciente de que esas modificaciones pueden acabar convirtiendo a *Scrum* en otra cosa. Está en la decisión de cada practicante de *Scrum*: ser un *Scrum Mass* (término coloquial que se usa para definir a la inmensa mayoría de gente que aplica *Scrum* con modificaciones) o un purista de *Scrum*, aferrado a los principios de la metodología, intentando que no se pierda ni el más mínimo potencial de aplicar todas las prácticas al 100%.

Se suele usar la afirmación de que *Scrum* es como el sexo en los adolescentes. Todos dicen que lo practican, pero solo el 10% lo hace, y, de estos que lo practican, casi todos lo hacen mal.

Desviándose del camino al Nirvana. Los Scrum Buts

Modificar *Scrum* es posible, pero se tiene que tener especial cuidado en añadir sobre *Scrum* lo que necesitamos y no variar la esencia de este. *Scrum* define una serie de artefactos, reglas y acciones que se usan de manera conjunta para obtener todos los beneficios de la metodología y, sobre todo, para generar predictibilidad. Esta predictibilidad nos puede ayudar

a adelantarnos a problemas que pueden aparecer en un proyecto, inspeccionando sus indicios. Si modificamos los cimientos de *Scrum* al hacer estas modificaciones, se romperá este ecosistema generando resultados no esperados. *Scrum* no es ni una bala de plata ni es el camino al Nirvana. Es un conjunto de prácticas que bien aplicadas producen un beneficio. No aplicarlas bien no implica tener un perjuicio, pero sí no obtener el mismo beneficio.

Figura 9.1. Camino al Scrum-Nirvana.

Existen distintas opiniones sobre el tema de la adaptación a *Scrum*, tantas como gente aplicándolo. Aquí reside una de las bellezas de *Scrum*, da lugar a interpretaciones y aplicaciones diversas. En la mayoría de los casos, todas estas opiniones tienen una justificación y son válidas dentro de sus contextos. Existen corrientes más puristas que definen que, en el momento en el que se modifica algo de *Scrum*, ya no se le puede denominar como tal. Al hacer estas modificaciones se crea otra metodología que comparte artefactos o reglas con *Scrum*, pero no es *Scrum*.

Hay quien opina que estas modificaciones forman parte de *Scrum* y generan lo que se conoce como los **Scrum Buts** (literalmente "*Scrum* pero..."). *Scrum But* es un término que fue creado por Eric Gunnerson en 2006 y normalmente se asocia a malas prácticas que se adoptan para justificar por qué no se está siguiendo alguna de las reglas de *Scrum*. Tienen

una estructura que normalmente se suele expresar como: *"Usamos Scrum pero + (práctica de Scrum no seguida) + (excusa lógica) + (adaptación de la práctica*)*"*.

Ejemplos de *Scrum Buts* son:

- "*Nosotros usamos Scrum, pero no hacemos reuniones diarias cortas porque el equipo está formado por personas de diferentes países con distintos idiomas y hemos decidido hacerlas más largas para que la gente pueda expresarse con tranquilidad en un idioma que no es el suyo"*.
- "*Usamos Scrum, pero no elegimos nosotros las tareas porque no tenemos demasiado conocimiento del proyecto y un especialista nos las asigna por nosotros"*.

Como se puede ver en los dos ejemplos anteriores, hay escalas en los impactos que los *Scrum Buts* pueden producir. Algunos apuntan a la base de *Scrum*, como puede ser la autogestión en el segundo ejemplo, y pueden ser realmente nocivos. Otros hacen ligeras modificaciones que pueden estar justificadas y no tienen por qué degenerar en una mala práctica de *Scrum* como el primer ejemplo. Se puede decir entonces que existen dos tipos de aproximaciones para la adaptación de *Scrum* a un entorno. Por un lado, están las modificaciones positivas, que añaden valor a la base de la metodología, respetando las bases de *Scrum*. A este tipo de *Scrum Buts* en algunos ámbitos se les conoce como **Scrum Ands** (literalmente "*Scrum* y..."), ya que se suele poder expresar en una frase con las siguiente estructura: *"Yo hago Scrum y, además, usamos un sistema de videopresencia para que los compañeros del equipo, que no están algunos días por teletrabajo, puedan trabajar como si estuvieran presencialmente"*. En este caso, se tiene el añadido del videopresencia para enriquecer o extender la práctica de tener al equipo sentado cerca.

Por otro lado, tenemos las modificaciones negativas. Estas modificaciones son atajos o soluciones que toman los equipos cuando hay un problema que les imposibilita cubrir alguna de las prácticas y normalmente afectan a la ejecución de *Scrum*. Estos ejemplos son la versión de los *Scrum Buts* más conocida y por lo que siempre se asocia una visión negativa a este concepto. Normalmente, a estos *Scrum Buts* se les denomina también como *Scrum Bad* o mal *Scrum*. Lo más importante que hay que destacar cuando aparece un *Scrum But* es que hay un problema que, en vez de solucionarlo de raíz, se está tratando de ocultar con la modificación de *Scrum*.

Scrum expone una disfuncionalidad que está desencadenando el problema. Es esa disfuncionalidad la que hay que reparar. A veces, se habla de que *Scrum* es el espejo en el que las organizaciones se miran. Cuando ven algo que no les gusta, suelen culpar a *Scrum* del problema, pero lo que realmente están viendo es el reflejo de su propia organización. A este efecto se le conoce como disparar al mensajero de *Scrum*.

Si existe un problema, además de encontrar una solución temporal mediante un *Scrum But*, se debería apuntar a ese problema en la organización e intentar resolverlo.

Nota:

Es importante evaluar la repercusión que el Scrum But puede generar. Esto se hace generando un Scrum aargh!, que se define con los siguientes datos: el Scrum But, el principio de Scrum que se está saltando y la consecuencia (beneficio potencial de Scrum que se pierde, cuantificación de ese valor perdido).

Ken Schwaber habla de un nuevo artefacto para *Scrum* relacionado con los *Scrum Buts*. A este artefacto lo denomina el ETC o el *Enterprise Transition Team*, que es un equipo *Scrum*, con su *Product Owner*, *Scrum Master* y equipo organizado para resolver todas las disfuncionalidades en la organización que se detecten con la aplicación de *Scrum*. Cuando se detecte un problema que se resuelve temporalmente con un *Scrum But*, se debería crear un elemento en el *Backlog* de este equipo que se priorizará y trabajará para resolverlo; convirtiendo ese *Scrum But* de nuevo en *Scrum*. Es interesante ver cómo *Scrum* puede convertirse en un agente de cambio de las organizaciones en las que se implanta.

Como se mencionaba anteriormente, la práctica de los *Scrum Buts* está muy extendida, Ken Schwaber la visualiza por medio de la siguiente gráfica.

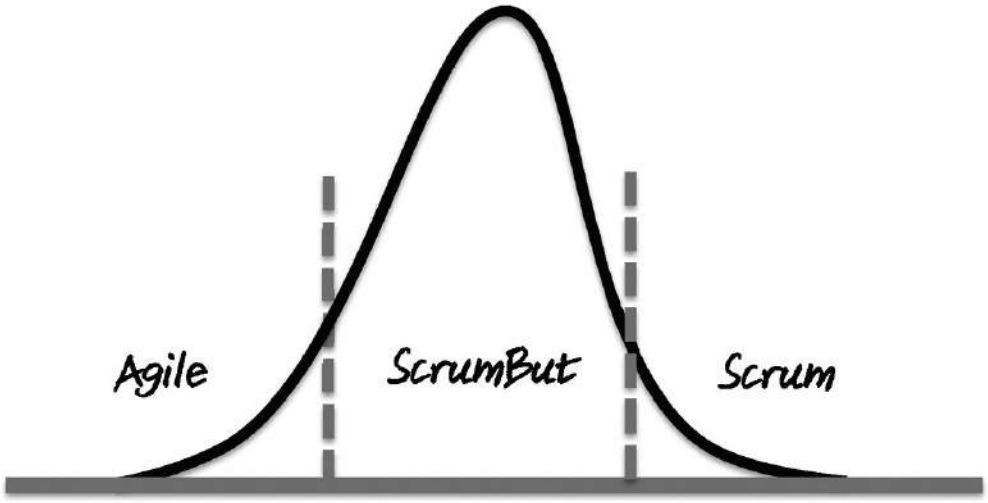


Figura 9.2. Distribución de la adopción en Scrum.

La imagen muestra una gráfica gaussiana en la que la inmensa mayoría de los practicantes de *Scrum* están realmente ejecutando algún tipo de *Scrum But*. El resto de los practicantes fuera de la media están realizando un *Scrum* de manera correcta o se quedan en la aplicación simplemente de los valores y principios ágiles, sin entrar en *Scrum*.

Estos datos son muy significativos e indican que muchos equipos aplican parcialmente *Scrum*. Si se analizan las razones de esta práctica parcial, se podrá comprobar que no se trata, en muchos casos, de problemas que eviten aplicar todas las prácticas de *Scrum*. Muchos equipos solo eligen las prácticas que les resultan más cómodas o atractivas, es lo que se

conoce como *Cherry-picking practices*: *"Nosotros hacemos Scrum, pero solo las prácticas que nos han parecido más interesantes"*. Es muy importante ejecutar el conjunto completo de las prácticas en *Scrum*, ya que, aunque cada una de ellas tiene un valor intrínseco por sí misma, la realimentación que tienen con las otras prácticas las hace aumentar su valor. Kent Beck usa el principio de Pareto que se explicaba en capítulo 3 para establecer que: *"Si sigues el 80% de los procesos, obtienes solo un 20% del beneficio total que se podría obtener"*, como ejemplo de la no aplicación de todas las prácticas de XP.

¿Cómo saber si se es un Scrum But?

En la anterior sección, se ha hablado de los *Scrum Buts* y de su problemática. Cuando se está haciendo un *Scrum But*, lo más importante es ser consciente de que se ha dejado de seguir una práctica de *Scrum* y de que, al menos, se debería tener una razón por la que saltarse un principio de *Scrum*. Gracias a Bas Vode y a Jeff Sutherland y su test, conocido como el *Scrum But* test, es más sencillo saber cuándo no se está aplicando *Scrum* de forma ortodoxa. Este test, que está publicado *on-line*, sirve para dos propósitos. Por un lado, da información sobre cómo se está aplicando *Scrum* en función de un cuestionario de nueve preguntas. Por otro lado, sirve para recoger información estadística de cómo se está aplicando *Scrum* en el mundo. El test se puede encontrar en varios sitios de Internet para su cumplimentación[41]. El test intenta cubrir un amplio espectro de la aplicación de *Scrum* realizando una pregunta sobre las categorías más propensas a modificaciones en *Scrum*:

- Iteraciones.
- Pruebas.
- Metodología ágil.
- *Product Owner*.
- *Product Backlog*.
- Estimaciones.
- *Burndown*.
- Interrupciones.
- El equipo.

Una vez cumplimentado el test, se genera una graduación sobre nueve según los pesos de las respuestas de cada pregunta.

Nota:

Es interesante realizar este test varias veces para observar cómo se va mejorando en las prácticas de Scrum con el tiempo, como parte del proceso de aprendizaje continuo.

Top 10 Scrum Buts

Se pueden encontrar infinidad de ejemplos de *Scrum Buts* en foros, libros y conferencias. En esta lista se han recopilado unos cuantos de ellos que resultan interesantes como ejemplos didácticos. Es importante recordar que *Scrum* nos muestra problemas que, aunque se minimicen u oculten con un *Scrum But*, deberían ser atacados y resueltos lo antes posible. Cuando un *Scrum But* aparece, independientemente de tolerarlo o no, es importante ser conscientes de la causa y, para ello, se suele usar la técnica de los cinco porqués. Esta técnica explica que, si se consiguen resolver cinco porqués de una consecuencia, se suele llegar a la raíz de esta. Por ejemplo, partamos de un *Scrum But* como pueda ser:

> *"Nosotros usamos Scrum, pero no hacemos reuniones de fin de Sprint con el Product Owner porque no puede asistir a las reuniones y un miembro del equipo hace su papel"*.

Si se realizan los cinco porqués, obtendremos lo siguiente:

- ¿Por qué no puede asistir el PO a las reuniones?: Porque le coinciden con otra reunión.
- ¿Por qué le coinciden con otra reunión?: Porque es *Product Owner* de otro producto también.
- ¿Por qué es *Product Owner* de otro producto y de este?: Porque este producto se quedó sin *Product Owner* y le asignaron a él este producto.
- ¿Por qué le asignaron el producto?: Porque no hay *Product Owners* suficientes en la organización.
- ¿Por qué no hay *Product Owners* suficientes en la organización?: Porque la organización no está focalizada en pocos productos y asume más productos o proyectos de los que puede ejecutar.

Como se puede comprobar en este ejemplo, *Scrum* destapa una ineficiencia de la organización que, por medio de un *Scrum But*, se intenta tapar. La solución no es el *Scrum But*, la solución debería apuntar más al problema que está a los cinco porqués de distancia.

A continuación, se van a presentar otros *Scrum Buts* que pueden usarse como indicadores de problemas en las organizaciones adoptando y adaptando *Scrum*. En sí, estas modificaciones pueden ser más o menos inofensivas, pero tienen que ayudar a visualizar qué tipo modificaciones intentan hacer los equipos para sobrellevar sus problemas.

> *"Nosotros usamos Scrum, pero no elegimos nuestras tareas, ya que no tenemos suficiente experiencia y un gestor nos dice lo que tenemos que hacer"*.

Este *Scrum But* está relacionado con lo que se conoce como los *smells* de *Scrum*, que son los síntomas de que *Scrum* no está funcionando correctamente. Cuando se llega a la situación de este *Scrum But*, se está modificando el principio de autogestión de los equipos, cayendo en la microgestión. Es muy importante crear un marco de aprendizaje para el equipo. Si el equipo no se considera maduro para tomar decisiones y no se le deja intentarlo, nunca alcanzará esta madurez. Sería interesante probar uno o dos ciclos permitiendo al equipo decidir, dándole soporte en la elección de tareas y corrigiendo las disfuncionalidades que puedan aparecer. Es importante hacerse las preguntas necesarias para destapar el problema causante de esta situación. Podría ser que el equipo de creación de un producto estuviese completamente integrado por personas con poca experiencia debido a que, para conseguir el proyecto de ejecución del producto, se rebajaron los costes para resultar competitivo.

"Nosotros usamos Scrum, pero no hacemos mejoras o corrección de problemas, ya que tenemos demasiadas funcionalidades que implementar y lo dejamos para el equipo de pruebas".

Figura 9.3. Microgestión.

La "marcha de la muerte" o *death march* es un término que se asocia a un proyecto en el

que se está intentando alcanzar un objetivo que está abocado al fracaso, pero se sigue trabajando en él, aunque se sea consciente del final esperado. En muchos proyectos, tener una cantidad de funcionalidad desorbitada para los recursos hace que el equipo se vea forzado a ritmos insostenibles de trabajo que afectan a la calidad del proyecto y a la motivación del equipo. Este *Scrum But* es insostenible en el tiempo.

Es importante analizar las causas de este *Scrum But*. Puede darse por una cantidad muy grande de funcionalidad y una presión elevada para ejecutarla. En esta situación, el equipo no marca el ritmo de trabajo. No decide qué puede entrar en un *Sprint* en función de *Sprints* anteriores y se suele asumir más trabajo del que se puede comprometer. El resultado es que nunca se cumplen los compromisos de las iteraciones de trabajo.

Podría darse el caso también de usar el *ScrumBut* como una excusa del equipo que no quiere trabajar en mejorar la calidad del producto. Si es así, se debería trabajar en concienciar al equipo en las bondades de dedicar tiempo a la mejora de la calidad del producto.

> *"Nosotros usamos Scrum, pero no hacemos Retrospectivas, ya que no solemos tener tiempo y comentamos nuestros problemas tomando café"*.

Las Retrospectivas son un elemento básico de *Scrum* para adaptarse y mejorar el proceso; son extremadamente potentes, pero solo tienen sentido cuando el equipo está convencido de sus beneficios. Si piensan que no tienen valor, es muy importante analizar por qué no les aporta ese beneficio e intentar corregirlo. Por ejemplo, en algunos casos existen conflictos entre los miembros del equipo que hacen que estos no se expresen libremente en la Retrospectiva y hacen que se sientan incómodos. La resolución de estos conflictos debe tratarse de forma eficaz para recuperar un clima de cooperación y *feedback* dentro del equipo.

> *"Nosotros usamos Scrum, pero no hacemos las reuniones diarias todos los días, ya que no las necesitamos porque estamos todo el tiempo hablando y las celebramos una vez por semana"*.

Hablar muchas veces no es lo mismo que comunicarse. Las reuniones diarias aportan muchos beneficios de comunicación y sincronización. Aunque los equipos estén constantemente hablando entre ellos, no suelen ser conversaciones entre todos los miembros, con lo que cada persona del equipo suele tener una visión parcial del estado de desarrollo del proyecto en cada momento. Las reuniones diarias suelen ayudar a generar la foto global del estado del proyecto para todos los miembros del equipo. Cuando los equipos intentan reducir la frecuencia de las reuniones diarias, suele ser porque no les aporta ningún beneficio o incluso les causa un perjuicio. La ejecución de las reuniones diarias debería ser revisada. Muchas veces, la duración excesiva, desviaciones del objetivo de la reunión o el uso de estas reuniones para la ejecución de un control sobre los miembros del equipo suele hacer que los equipos intenten evitarlas.

"Nosotros usamos Scrum, pero no hacemos el diseño, la implementación y las pruebas de cada historia de usuario en el mismo Sprint, porque no nos da tiempo a llevar a cabo todas las tareas en el mismo Sprint y las hacemos en Sprints consecutivos".

Cuando se llega a este *Scrum But*, el problema suele estar en dos sitios.

Puede estar en el *Product Backlog* y en su mantenimiento. En muchos casos, el *Product Backlog* no está trabajado lo suficiente y se queda en una capa de épicas que difícilmente pueden ser abordadas de forma completa en una iteración. La solución que toma el equipo es trabajar esa épica en varios *Sprints*, pero cae en un modelo de mini-cascada, haciendo las fases del desarrollo de una funcionalidad de manera consecutiva, en vez de realizarlas de forma conjunta en un mismo *Sprint*.

La segunda razón puede ser que el equipo no esté estimando de manera correcta las historias de usuario, centrándose solo en el concepto de la implementación de estas. Se debe valorar el esfuerzo completo que implica el desarrollo de una historia de usuario: desde que se empieza a pensar en ella hasta que se certifica que ha sido correctamente implementada siguiendo sus criterios de aceptación.

"Nosotros usamos Scrum, pero no tenemos un Scrum Master y Product Owner separados porque no los necesitamos y son la misma persona".

Scrum Master y *Product Owner* son roles y no personas. Por esta razón, físicamente nada impide que puedan ser una misma persona, pero no es una buena adaptación de *Scrum*, como ya se vio en el capítulo 5. Por un lado, son roles que deberían consumir mucho tiempo. Si los desempeña la misma persona, puede dar por seguro que las tareas de alguno de los dos roles se quedarán sin hacer. Por otro lado, son dos roles que son incompatibles porque inevitablemente sus atribuciones o principios van a entrar en conflicto constantemente. El *Product Owner* vela por el producto que se quiere desarrollar mientras que el *Scrum Master* vela por el proceso y el equipo. En muchas situaciones, las necesidades del producto pueden empujar hacia un camino que vaya en contra del equipo o del proceso. En la situación en la que están todas las competencias sobre la misma persona, normalmente la mitad *Product Owner* suele ser la que gana llevándose el equipo la peor parte. Tras este *Scrum But*, se suele esconder una organización que apuesta por *Scrum* a medias y no facilita los recursos necesarios a un equipo para desempeñar los roles necesarios. Es este el problema que debería tratar de resolverse. En caso de no poder encontrar una solución, suele ser más lógico compartir el rol de *Scrum Master* entre los miembros del equipo. Muchos equipos lo rotan entre *Sprints* para que no recaiga siempre sobre la misma persona.

Figura 9.4. Conflicto de criterios.

"Nosotros usamos Scrum, pero no hacemos iteraciones cortas porque nuestro proyecto está muy claro y las hacemos cada 3 meses".

Si un proyecto está tan claro que en 3 meses no va a cambiar en nada, posiblemente lo que no esté claro es por qué se está usando *Scrum*. Es interesante usar *Scrum* cuando puede aportar un valor dado ante una situación específica de realización de un proyecto o creación de un producto. Si la situación no es aplicable a la utilización de *Scrum*, quizás sea interesante utilizar una metodología que se adapte más a la naturaleza de la tarea que se está llevando a cabo. Evitar utilizar plazos cortos de interacción con el cliente puede ocultar adicionalmente otras disfuncionalidades que podrían analizarse. Debería revisarse la implicación del cliente en el proceso, ya que quizás no tenga interés en la ejecución real del proyecto. Otro posible problema podría ser la imposibilidad de generar un resultado tangible en un corto periodo de tiempo, lo cual fuerce el alargamiento de los procesos de realimentación.

"Nosotros usamos Scrum, pero no hacemos las reuniones diarias todos los días porque el Scrum Master no puede estar todos los días y solo las hacemos cuando está él".

Peligro, este *ScrumBut* nos está hablando de un posible problema subyacente. Independientemente de hacer o no las reuniones todos los días, es de especial atención que no se hagan porque el *Scrum Master* no esté. Esta situación se puede estar dando porque el equipo está reportando al *ScrumMaster* en vez que comunicarse entre ellos los avances del proyecto. Debería analizarse la reunión diaria cuando se realiza con el equipo y *Scrum Master* para poder mitigar este problema. No son reuniones de reporte, son reuniones de sincronización del equipo.

Finalmente, tenemos algo que no es un *Scrum But* como tal, pero, por estar tan extendido, se merece una mención especial en este capítulo.

"Nosotros no lo llamamos Scrum, pero ya estábamos haciendo todas estas cosas".

Scrum es sentido común, ofrece un conjunto de prácticas razonables que no tienen un valor novedoso en sí mismas, pero sí en la agrupación de estas para ofrecer el máximo beneficio. Por esta razón, cuando se empieza a conocer *Scrum*, al principio resulta todo muy evidente y la gente suele asociarlo con tareas que ya está haciendo. Es como cuando te presentas a un examen, no has estudiado nada y te resulta muy sencillo por el desconocimiento. Lógicamente, suspendes. Cuando te presentas y has estudiado, el examen te parece más complejo, pero acabas aprobando. Con *Scrum* ocurre lo mismo. Cuanto más se practica, más se aprende sobre él, por lo que se suele dar un consejo: aplica y aprende antes de adaptar. Experimenta con *Scrum* tal y como es antes de decidirte a cambiarlo para saber y entender realmente qué es lo que está fallando.

¿Es Scrum el final del túnel?

Scrum no es el final, no es el método perfecto, es un punto al que se ha llegado después de años de metodologías encorsetadas. Es la consecuencia y el principio de algo nuevo. *Scrum* ha supuesto un cambio hacia un nuevo modelo de gestión, pero no es el límite más allá del cual no se puede continuar. Como se verá en posteriores capítulos, *Scrum* se puede extender o incluso hay otras propuestas que aportan un nuevo valor a los principios ágiles. Lo importante es saber siempre por qué se han realizado las cosas para iniciar una propuesta de cambio que nos permita mejorar.

---

41 URL de ejemplo para ejecutar este test: .

Segunda parte
Aplicación avanzada de los métodos ágiles

10 XP. Una aplicación de los métodos ágiles al desarrollo software

En este capítulo aprenderá:

- El origen de XP.
- El ciclo de vida de XP.
- Dónde es posible aplicar XP.
- Los valores, principios y prácticas de XP.
- A combinar *Scrum* con XP.

Las prácticas, valores y principios de la programación extrema, más conocida como XP (*eXtreme Programming*), no fueron ideados todos a la vez. De hecho, muchas de las prácticas que propone este método de programación no son en absoluto nuevas. Lo novedoso de XP reside en su propuesta de aplicar las prácticas de forma simultánea y que se realimenten entre ellas. Algunas de ellas son técnicas que se han aplicado con éxito anteriormente y han demostrado resultar tremendamente valiosas. XP sigue evolucionando a lo largo de los años; poco a poco, se han ido incorporando prácticas nuevas a su catálogo.

XP empezó a gestarse como método en los años 80 con Kent Beck y Ward Cunningham cuando trabajaban en un proyecto utilizando *Smalltalk*. *Smalltalk* es un lenguaje de programación orientado fundamentalmente al público en general y no solo para informáticos especializados. La filosofía de este lenguaje se basa en elementos como la programación en parejas, refactorización, adaptación al cambio, integración frecuente, desarrollo iterativo, pruebas constantes y, sobre todo, una continua comunicación y relación con el cliente.

Hay algunos autores que incluso afirman[42] que XP es la cultura de trabajo de *Smalltalk* ampliada a otros entornos de programación.

Poco a poco, se fueron incorporando más prácticas entre las que cabe destacar el desarrollo dirigido por pruebas, en inglés *Test Driven Development* (TDD).

Kent Beck publicó el libro *Extreme Programming Explained*[43] en 1999, en el que incorporaba a las prácticas puramente de programación numerosos conceptos de *Scrum* para fortalecer así la convivencia del desarrollo de software con algunos conceptos de gestión de equipos. Al final de este capítulo, en el apartado “Combinando *Scrum* con XP”, se comentan los aspectos que comparte *Scrum* con XP y el porqué es posible y valiosa su convivencia.

XP ha continuado evolucionando a lo largo de los años y, en 2004, Kent Beck publicó la segunda edición[44] del libro anterior para reflejar esta constante evolución.

Ciclo de vida de XP

En numerosas ocasiones, se atribuye el fracaso de los proyectos a la poca definición de requisitos o a un mal diseño de la arquitectura de un sistema al inicio del proyecto. En una primera aproximación, es difícil comprender cómo XP propone trabajar eliminando las fases de definición exhaustiva de requisitos, diseño y arquitectura, así como los documentos asociados a estas fases antes de comenzar el desarrollo. El método XP en absoluto propone eliminar estas actividades, sino más bien potenciarlas. La forma de trabajar es justo la opuesta, definiendo los requisitos, arquitectura y diseño cada día, y no en un periodo de tiempo determinado y acotado. XP apuesta por simultanear todas las fases y llevarlas a cabo en paralelo de forma que se vaya adaptando el producto y el sistema a las necesidades a medida que vayan surgiendo.

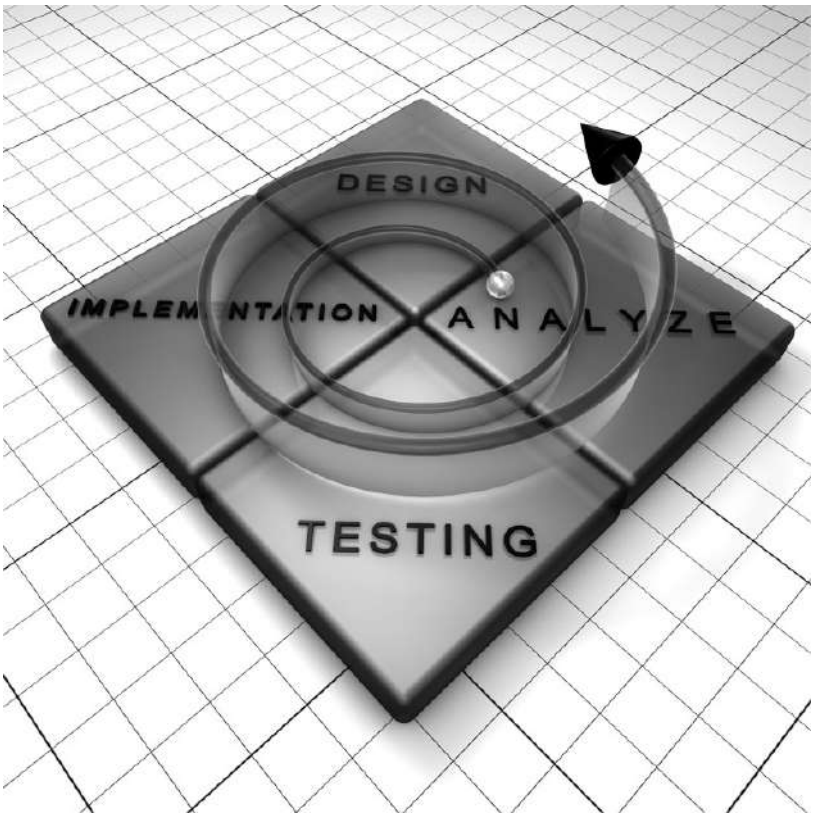


Figura 10.1. Fases simultáneas.

Cada semana se realiza un ciclo completo o iteración en la que se aborda parte de cada una de las fases tradicionales de un desarrollo software, es decir, se hace un poco de todo. La manera de conseguir que esto funcione es trabajar con las llamadas “historias de usuario”, que, en definitiva, son requisitos que dan valor al cliente. Cada semana se planifica en qué historias trabajar y se llevan a cabo de forma completa con su análisis, diseño, pruebas y desarrollo. Al final de la semana, se podrá mostrar al cliente el resultado de la funcionalidad obtenida de forma que pueda opinar sobre ella.

Nota:

XP apuesta por realizar todas las fases simultáneamente de forma que la planificación, análisis, diseño, desarrollo, pruebas y despliegue de un producto se produzca frecuentemente y se genere funcionalidad completa en cada iteración.

A continuación, se detalla algo más sobre cómo llevar a cabo cada una de estas etapas de planificación, análisis, arquitectura, diseño, codificación y pruebas aplicando XP:

- **Planificación:** En cualquier equipo que vaya a desarrollar un proyecto, debe haber personas responsables de tomar las decisiones de negocio y que tengan clara cuál es la visión del producto, el plan de entregas, establezcan las necesidades que debe cubrir el

sistema y gestionen los riesgos. El resto del equipo hará sugerencias y estimaciones para matizar este plan inicial de entregas. Las historias de usuario deberán estar priorizadas para reflejar el orden en que se debe construir el producto, desde el punto de vista de negocio y contemplando también la secuencia lógica desde un punto de vista más técnico.

Naturalmente, en las etapas iniciales del proyecto, es necesario dedicar un mayor esfuerzo a la planificación. Sin embargo, esto no significa que no se vuelva a planificar a lo largo del desarrollo. En función de las nuevas necesidades que vayan surgiendo, los clientes deben revisar y mejorar el plan de entregas.

También será necesario adaptar el plan inicial en el caso de que surjan imprevistos o dependencias externas que dificulten mantener dicho plan.

Las historias más prioritarias serán las que se implementen en primer lugar. A su vez, el equipo, al comienzo de cada semana, planificará de forma detallada la manera de abordar la siguiente iteración. Además, cada día el equipo organizará el trabajo para esa jornada.

En definitiva, se planifica al inicio del proyecto, al inicio de cada iteración y, aún más en detalle, todos los días.

- **Análisis:** Para que el análisis se mantenga actualizado durante todo el proyecto, los clientes deben estar en comunicación constante y cercana con las personas que están construyendo el producto. Cuando un desarrollador tenga dudas sobre cómo implementar un requisito, debe poder preguntar al cliente. De igual manera, se debe trabajar en estrecha colaboración con los responsables tanto de pruebas como de diseño gráfico, de manera que no quepa ambigüedad en la definición de los requisitos.
- **Diseño y codificación:** XP propone trabajar de manera que tanto el diseño como la arquitectura se creen de forma incremental. De este modo, se mejora el diseño y la arquitectura poco a poco y de forma constante.

Nota:

XP propone técnicas avanzadas para acercarse constantemente a la excelencia técnica. Una de estas prácticas más conocidas es Test Driven Development (TDD). TDD permite realizar de manera simultánea el diseño, las pruebas, la arquitectura y la codificación. Esta técnica de desarrollar ayuda a los programadores a escribir el código que hace exactamente lo que se espera que haga, ni más ni menos.

Por otro lado, los desarrolladores deben usar un sistema de control de versiones para la gestión de la configuración y mantener actualizado su entorno de desarrollo y otras prácticas de desarrollo como son usar estándares de código, prácticas de integración, etc.

- **Pruebas:** Uno de los pilares sobre los que se fundamenta XP son las pruebas. Las pruebas deben realizarse a todos los niveles y todos los implicados en un proyecto contribuyen a su realización. Los desarrolladores construyen el código a la vez que lo prueban, ya que utilizando TDD se están realizando pruebas completas del código. Por otro lado, los usuarios realizan pruebas de aceptación. En ocasiones, una parte del equipo está dedicado a realizar pruebas más amplias y completas, pero, si no fuera así, el mismo equipo de desarrollo ejecutaría dichas pruebas.

Prácticas como la programación en parejas, que consiste en que dos personas escriban el código en una única máquina, la revisión frecuente del código, la integración continua y la automatización de pruebas contribuyen al aseguramiento permanente de la calidad de lo que se está construyendo.

Nota:

Un equipo que aplique correctamente las prácticas de XP generará un número muy limitado de fallos o errores (bugs) en el producto que desarrolle.

- **Despliegue:** La forma de construir el producto con XP hace posible que, al finalizar cada semana, el software obtenido pueda ser puesto en producción, ya que la funcionalidad comprometida está asegurada. Esto no significa que se realicen entregas al cliente final con esta frecuencia. Las entregas se realizarán siguiendo el plan de entregas establecido previamente con el cliente.

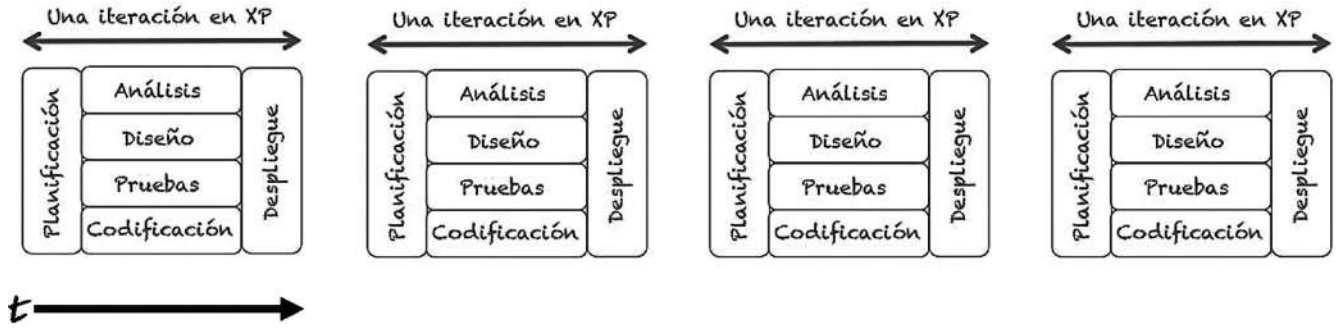


Figura 10.2. Ciclo de vida con XP.

Dónde es posible aplicar XP

En ocasiones, ocurre que las condiciones del entorno de trabajo, la organización o el tipo de proyecto que hay que construir no facilitan aplicar las prácticas que XP propone.

La aplicación de XP da por hecho que existe una predisposición por parte de las personas implicadas en un proyecto y sin la cual es imposible que se adopten sus prácticas con éxito.

Cada persona debe sentirse como parte de un equipo y debe participar de forma voluntaria. No se debe obligar a nadie a hacer nada en lo que no crea. Un equipo no es únicamente un conjunto de personas que trabajan juntas, sino de un grupo con intereses y objetivos compartidos.

Cuando un equipo trabaja, adopta los valores, principios y prácticas de XP, se da por supuesto que todos los implicados están dispuestos a realizar los cambios necesarios para alcanzar los objetivos del proyecto. Es decir, en el momento en el que se detecte que, si se modifica algún procedimiento, actuación o práctica, se podrá mejorar la productividad en el trabajo, se tendrá una actitud activa para realizar dichos cambios.

Recuerde:

La aplicación de las prácticas de XP depende más de las personas implicadas que del tipo de proyecto.

Tal y como comentan Shore y Warden en el libro *The Art of Agile Development*[45], existen una serie de premisas que deben cumplirse para poder trabajar de esta manera y son las siguientes:

- **Apoyo de los responsables en la empresa:** Es necesario que el equipo comparta el lugar de trabajo de forma que exista un entorno que facilite la programación por parejas y la comunicación frecuente. Los responsables de las pruebas, clientes y responsables del producto se deben sentir como una parte del equipo completo.
  Esta disposición del entorno de trabajo es muy difícil de llevar a cabo sin el apoyo directo de los responsables de la empresa en la que se trabaja.
  Es posible que, mientras el equipo está aprendiendo a aplicar estas prácticas, tal vez su productividad disminuya algo. Si los responsables de la empresa confían en los desarrolladores y tienen paciencia, los beneficios obtenidos serán altos, ya que, tal y como se ha comentado, aplicando estas prácticas el producto obtenido cubrirá ampliamente las expectativas del cliente, el número de defectos disminuirá notablemente y la productividad final aumentará.
  Por último, es necesario que el equipo tenga capacidad de tomar las decisiones técnicas necesarias sobre el proceso de desarrollo y todo lo relacionado con el mismo. Un ejemplo de ello sería poder realizar de manera autónoma el control de versiones.
- **Compromiso del equipo:** Tan importante como el apoyo de los responsables es el compromiso de todo el equipo para aplicar las prácticas de XP. Si el equipo al completo no confía en la potencia de XP, la recomendación es que no debe aplicarse. No tiene sentido forzar a una persona a aplicar unas prácticas en las que no se confía. En ocasiones, se puede tratar de animar a esta persona a trabajar así durante un tiempo y revisar esta decisión pasados unas semanas o meses. Si no funciona, directamente lo

mejor es dejar de aplicarlo.

- **Cercanía del equipo:** Tal y como se ha comentado anteriormente, el equipo debe trabajar en el mismo espacio. Es verdad que hay técnicas que hacen posible una buena comunicación entre personas que no están trabajando en el mismo centro, pero, indudablemente, la comunicación se ve resentida y el equipo perderá potencia en su trabajo. Muchas de las prácticas que XP recomienda llevan implícito el que el equipo trabaje junto.
- **Cliente cercano:** La necesidad de decidir al inicio de cada iteración qué requisitos han de implementarse hace imprescindible el trabajar de forma cercana con el cliente. También es importante poder consultarle cualquier duda o cuestión relacionada con la funcionalidad que se esté desarrollando.

  En el caso en que el cliente no pudiera tener ese grado de disponibilidad, puede delegarse esta responsabilidad a otra persona con capacidad para tomar este tipo de decisiones. En ocasiones, a este "embajador" del cliente se le denomina *Proxy* y hace posible que el equipo continúe su trabajo sin que se vea afectado por este asunto. Esta figura sería la equivalente a la del *Product Owner* en *Scrum*.
- **Tamaño correcto del equipo:** La recomendación habitual es que el equipo completo esté compuesto por no más de 12 personas. Esto no significa que si un equipo es mayor o menor no puedan aplicarse las prácticas de XP, sino que, simplemente, será más complicado hacerlo. En el caso de equipos muy grandes, es necesario buscar la manera de escalar la aplicación de XP, por ejemplo, dividiendo el equipo en grupos por funcionalidad.
- **Aplicar todas las prácticas:** XP está concebido de manera que se ha tratado de eliminar todo aquello que es superfluo. Con esta premisa, debe tenerse gran cautela a la hora de decidir dejar de aplicar alguna de sus prácticas y el impacto que tendría en el resultado final. Por ejemplo, si se decide trabajar con equipos separados, deberá valorarse el impacto de no poder hacer revisiones de código de forma frecuente, ni compartir el código y el conocimiento de manera sencilla, etc.

Recomendación:

En el caso de encontrarse con alguna barrera que le impidiera aplicar XP, y realmente deseara hacerlo, trate de eliminar dichos impedimentos, en vez de tratar de convivir con ellos.

Valores, principios y prácticas de XP

Tal y como se comentó en el capítulo "Métodos ágiles", las prácticas de XP representan el

día a día del trabajo de un desarrollador. Sin embargo, carecen de sentido si no se sigue el conjunto de ideales a los que llama valores. Las prácticas son la evidencia de los valores, la concreción de una filosofía de trabajo. Por ejemplo, ¿qué sentido tendría hablar de programación en parejas si tenemos problemas de comunicación dentro del equipo o si no respetamos el trabajo de los demás?

El nexo de unión entre los valores (el pensamiento) y las prácticas (el día a día) son los principios, ya que dan las pautas para la aplicación de una filosofía de trabajo a la aplicación directa en el trabajo diario.

Es importante recordar que seguir las prácticas no hace maestro a un programador. Aplicar una práctica en concreto desde luego aportará valor al proyecto, pero es la aplicación de las diferentes prácticas en distintos entornos lo que hará posible que se detecten los posibles riesgos de un proyecto y evitarlos a tiempo.

Veamos, a continuación, con algo más de detalle cada uno de los valores, principios y prácticas que propone Kent Beck[46] en su revisión de XP.

Figura 10.3. eXtremme Programming.

Los valores

Los 5 valores que guían el desarrollo en XP son la comunicación, simplicidad, *feedback*, valentía y el respeto, pero, obviamente, estos no son los únicos valores que deben aplicarse. En cada empresa y situación personal habrá que decidir qué otros valores adicionales incorporar, como, por ejemplo, asegurar la calidad de vida o el ser predecibles.

- **Comunicación:** La base del éxito del trabajo de un equipo de desarrollo es la comunicación, sin lugar a dudas. La comunicación entre los desarrolladores y, a su vez, con el cliente es la mejor forma de evitar la mayoría de los problemas y errores.

Si se comentan los problemas o impedimentos de forma constante entre los miembros del equipo, se podrá contar con el conocimiento de alguien que tal vez haya lidiado con un problema similar en otras ocasiones o, entre todos, pensar una solución y no hacerlo de forma individual.

Hay muchas formas de comunicarse, además de hablar. Por ejemplo, un código simple y bien escrito es una forma eficaz de comunicación entre los desarrolladores, ya que cualquiera de ellos podrá entender el código desarrollado por otro miembro de su equipo.

- **Simplicidad:** Las cosas deben hacerse de la forma más simple posible. Cuidado, es importante no confundir simple con simplista y para esto hace falta experiencia. No es fácil construir un sistema sencillo y que cumpla exactamente los requisitos establecidos.

  La forma de simplificar un sistema muchas veces se consigue con un método de trabajo de ensayo-error. Construir de forma simple disminuye la cantidad de código que hay que escribir y aumenta la calidad. Se debe simplificar el diseño, documentar el código de forma muy simple pero que aporte valor y utilidad y refactorizar el código, eliminando redundancias y las partes inservibles para mantener la sencillez del mismo.
- **Feedback:** El *feedback* debe entenderse como una combinación de comentarios, reacciones, sugerencias, opiniones, etc. Es necesario poder conocer siempre cómo de cerca está el producto que se está construyendo de lo que realmente se necesita.

  Una de las herramientas fundamentales para obtener comentarios y sugerencias del cliente es la estrecha comunicación con él. De esta forma se favorece que el equipo conozca con mucha frecuencia su opinión sobre el trabajo que se está realizando. Asimismo, se evita tener que realizar costosas modificaciones y ayuda los desarrolladores a centrarse en lo que realmente el cliente necesita. Teniendo esta comunicación, se eliminan malentendidos que pueden generar que en ocasiones haya que tirar una gran cantidad de trabajo que en realidad no era necesario realizar.

  La otra gran herramienta para obtener información sobre el sistema son los test automáticos que se crean a la vez que el producto, ya que ayudan a revisar el estado del código y facilitan detectar, de forma casi inmediata, posibles fallos producidos por cambios.

  El *feedback* está muy relacionado con la comunicación y la simplicidad. Sin comunicación, es imposible obtenerlo y, por otro lado, cuanto más simple se construya un sistema más fácil será opinar sobre él.
- **Coraje:** Hasta ahora, no se ha dicho nada que no sea de sentido común y es necesario ser valientes para aplicar estos valores a nuestro trabajo. Necesitamos valentía para comunicarnos clara y eficazmente con el cliente y con nuestros compañeros y afrontar sin miedo los cambios en el producto. Necesitamos ser valientes para construir de forma simple solo y exactamente lo que se necesita en este momento y tener valor para saber parar sin caer en la tentación de crear algo decidido por nuestra cuenta y de

manera individual. Necesitamos esa valentía para recibir y dar *feedback* de forma productiva y, por supuesto, es necesario ser valiente para aplicar algunas de las prácticas que XP propone, ya que, en ocasiones por desconocimiento, puede haber quien dude de su utilidad.

Cuidado:

La valentía aislada puede ser peligrosa y mal entendida, por este motivo debe ir de la mano de los otros valores de XP y tener siempre presente las consecuencias que una acción puede tener.

- **Respeto:** Los cuatro valores anteriores llevan implícito el respeto. Ningún método puede funcionar si no se trabaja con respeto mutuo, valorando el trabajo de los demás y sus aportaciones. Es necesario tener siempre muy presente cómo puede afectar nuestro trabajo a las personas que trabajan con nosotros y a su organización. De igual manera, hay que tener en cuenta a todas las personas que puedan interactuar con el producto que se está generando.

Los principios

Llevar al trabajo del día a día los valores anteriores no es algo trivial, ya que se trata de conceptos demasiado abstractos. Es necesario concretar algo más para ponerlos en práctica. Para ello, XP propone algunos principios útiles para un mejor desarrollo. Al igual que ocurre en el caso de las prácticas, los principios aquí mencionados no son los únicos y dependerá de cada sistema qué principios extra añadir. Los principios aplicables a todo desarrollo software son:

- **Humanidad:** El software lo desarrollan personas y es importante recordarlo y tener presente que los factores humanos son la clave para crear un software de calidad. Hablamos de factores como, por ejemplo, la necesidad de una persona de sentir que el trabajo que realiza es útil y sentir que sus compañeros entienden sus problemas y necesidades. Con humanidad también se quiere reflejar la necesidad que tienen las personas de poder aprender constantemente, de crecer profesional y personalmente, así como la necesidad de una estabilidad laboral.
- **La economía:** El producto que se cree debe ser rentable tanto a corto como a medio plazo, es decir, debe producir beneficios. Una aplicación directa de este principio es trabajar primero en los requisitos más prioritarios de forma que el beneficio obtenido sea el máximo cuanto antes.

Figura 10.4. XP propone construir de forma que se obtenga el máximo beneficio cuanto antes.

- **El beneficio mutuo:** Este principio se considera uno de los principales de XP y también uno de los más difíciles de poner en práctica. El beneficio mutuo propone pensar siempre en el beneficio de todas las partes implicadas en un desarrollo, es decir, pensar en los desarrolladores, clientes y cualquier otro actor o *stakeholder* afectado por el producto. Cuando existe un problema, y se actúa a la "desesperada", es fácil que alguna parte salga perjudicada de las acciones tomadas. Ejemplos de cómo aplicar este principio en el código es refactorizar, de manera que, si alguna persona en el futuro tiene que utilizar el código que otra escribió, le resultará más sencillo hacerlo. Elegir bien los nombres utilizados en el código y usar metáforas dentro del mismo ayudará también notablemente a las personas que hereden el código.
- **La autosemejanza:** Buscar soluciones similares en diferentes contextos. Los patrones de la forma de trabajo deben repetirse adaptándolos a la situación en la que estemos. Para una iteración, se pensarán las historias de usuario y se asociarán sus criterios de aceptación para cumplirlas. Pues, de forma análoga, al escribir código se escribirá el código y sus pruebas unitarias correspondientes para comprobar que funciona correctamente. La autosemejanza se refiere también a tener presente qué se hizo en el pasado en una situación similar a la actual y adaptarlo donde sea posible.
- **La mejora continua:** La perfección no existe, pero hay que tratar de acercarse a ella cada día pensando constantemente qué es lo que se puede hacer mejor mañana. Para

ello, es necesario el *feedback* del cliente, el análisis del resultado de las pruebas y la experiencia del propio equipo.

- **La diversidad:** Si en un equipo todos saben de lo mismo, piensan parecido y actúan de forma similar, podrán trabajar muy cómodamente, pero perderán efectividad. Tener dos enfoques diferentes sobre cómo hacer un diseño debe verse como una oportunidad de aprender y mejorar y en absoluto como un problema. El que en un equipo haya personas que piensen diferente puede ser una fuente de conflictos, pero, si se gestiona bien, puede ser una fuente de conocimiento y de intercambio de perspectivas y enfoques.
- **La reflexión:** Un equipo habitualmente piensa en cómo está trabajando y por qué está haciéndolo de esa manera para tratar de mejorar siempre. Pero, cuidado, debemos evitar caer en el análisis-parálisis, ya que no se debe analizar en exceso el cómo hacer las cosas y no empezar a producir nunca o hacerlo demasiado tarde. La reflexión en profundidad sobre cómo se trabaja debe realizarse después de hacer un trabajo y antes de comenzar otro, pero siempre la reflexión debe ir acompañada de la acción.

Figura 10.5. La reflexión sobre la forma en que se trabaja es necesaria para buscar mejoras.

- **El flujo:** Lejos de trabajar de forma secuencial e ir cerrando una fase del desarrollo antes de comenzar otra, XP propone abordar simultáneamente todas las actividades del

desarrollo (análisis, diseño, pruebas y desarrollo). Todas las prácticas de XP están basadas en que este flujo en el trabajo existe. Para mejorar constantemente lo que se está desarrollando, es muy eficaz ir creando con frecuencia pequeños incrementos de funcionalidad valiosa y, para ello, es necesario simultanear las fases del desarrollo.

- **Oportunidad:** Donde hay un problema, hay una oportunidad para mejorar. Precisamente, todas las prácticas de XP son eficaces para tratar de poner solución a los problemas tradicionales del desarrollo de software.
- **Redundancia:** Si existe algún aspecto crítico, se deben buscar varias soluciones de forma que, si una de ellas falla, podrá funcionar la otra alternativa buscada. En ocasiones, los defectos se encontrarán varias veces por diferentes caminos, pero se tiene la tranquilidad de saber que se controlan más los posibles fallos. Es importante y muy necesario controlar que la redundancia sea realmente útil y que siempre sea valiosa para el sistema y revisar que no se estén utilizando prácticas que podrían ser eliminadas sin causar mayor impacto en el producto.
- **Los fallos:** El cometer fallos o errores es algo tremendamente útil si se aprende de ellos. Sin perder el sentido común, es más productivo intentar hacer algo y fallar en el intento y, a continuación, volver a intentarlo, que tratar de buscar la forma perfecta de construir algo y retrasarse demasiado en la ejecución.
- **La calidad:** Trabajar con un nivel alto de calidad aumenta la eficiencia y la productividad y esto no se trata simplemente de un factor económico. Trabajar con calidad afecta tanto a los beneficios del producto como a los creadores del mismo, ya que, entre otras cosas, se sienten orgullosos de su creación. Por otro lado, la experiencia demuestra que los proyectos no acaban antes si se acepta que su calidad sea más baja de lo debido.

  Sin embargo, la calidad no debe convertirse en algo obsesivo y no se debe utilizar esta excusa y no actuar. En ocasiones, es necesario pasar a la acción construyendo siempre de la mejor manera posible para ese momento y en las circunstancias que lo rodean. Una muestra de calidad es también ir poco a poco construyendo grandes cambios.
- **Pequeños pasos:** Trabajar con iteraciones cortas en absoluto significa ir despacio. Se pueden realizar numerosas y productivas pequeñas iteraciones. El beneficio de trabajar así es que, si se da un paso equivocado, se podrá rectificar rápidamente, ya que se revisa el trabajo realizado con frecuencia y el impacto de la equivocación será pequeño.
- **Aceptar la responsabilidad:** Para que un equipo sea eficaz, las personas que lo componen deben ser responsables. Es poco útil decir a cada persona lo que debe hacer en cada momento, ya que las posibilidades de no acertar con el trabajo que se le asigna son altas. Probablemente, pueda hacer más o tal vez menos. Será cada persona la que, responsablemente, desarrolle la mayor cantidad de trabajo de la mejor manera posible.

Las prácticas

Las prácticas son simplemente la manera en que los equipos trabajan en su día a día. La adopción de unas u otras prácticas variará en función de la situación en la que se apliquen.

Kent Beck clasifica las prácticas en dos bloques: las prácticas primarias y las prácticas corolario. La adopción de las prácticas primarias puede hacerse de forma independiente y su aplicación aporta valor de manera inmediata al producto. Para aplicar las prácticas corolario, es necesario tener cierta experiencia en la aplicación de las primarias. El beneficio de utilizar varias prácticas en paralelo es enorme, por lo que se recomienda ir añadiendo las diferentes prácticas al desarrollo tan pronto como sea posible.

Prácticas primarias

Estas prácticas ayudan directamente a mejorar el desarrollo de software. Elija, en función de su entorno, cuáles debe empezar a incorporar en su trabajo.

Las prácticas primarias son las siguientes:

- **Historias:** Los requisitos del sistema deben escribirse en lenguaje de cliente y deben estar redactados de forma que reflejen pequeñas unidades de funcionalidad visible para el cliente. Un ejemplo de historia puede ser: “Como usuario de una página Web de música, quiero poder consultar los discos de mi cantante favorito para poder comprar el que me falte”.

  Una historia debe ir acompañada de la estimación del esfuerzo que supondrá implementarla y es muy útil si va acompañada de una descripción corta de la misma o incluso, cuando sea posible, un diseño gráfico de la funcionalidad que se espera.
- **Ciclos semanales:** XP propone trabajar en iteraciones de una semana de forma que al inicio el cliente elija las historias que haya que desarrollar en la semana siguiente y el equipo las divida en las tareas técnicas necesarias para llevarla a cabo. Al finalizar cada iteración, se revisará el trabajo realizado durante esa semana para detectar posibles desviaciones respecto al objetivo global del proyecto y se planteará el objetivo de la siguiente iteración.

  Los equipos aprenderán también a mejorar sus planificaciones de manera que organizar el trabajo para la semana siguiente les ocupe cada vez menos hasta limitarlo al orden de una hora.
- **Revisiones y planificaciones trimestrales:** Manejando una escala de tiempo mayor, se propone revisar trimestralmente el estado del equipo, del proyecto y los progresos, a la vez que se analizará si se mantienen los objetivos más generales establecidos para el proyecto.

  Así mismo, es el momento de plantear con qué nuevos temas se va a continuar

trabajando y analizar posibles impedimentos, pero desde una perspectiva más amplia que la que ofrecen las iteraciones semanales.

Un trimestre también es un periodo de tiempo razonable para revisar o definir nuevos hitos con suministradores, clientes, etc.

- **Holgura:** Cuando se realicen los planes para las iteraciones, deje cierta holgura de forma que, si surgen imprevistos, siempre existan tareas que puedan dejar de realizarse sin ocasionar demasiado impacto. Se trata, sobre todo, de no hacer promesas que no se puedan cumplir. De igual manera, al trabajar en un ambiente de responsabilidad y compromiso, si diera tiempo a realizar alguna tarea no comprometida en la planificación inicial, debería llevarse a cabo.
- **Sentarse juntos:** La mejor forma de potenciar la comunicación es ubicar al equipo de desarrollo en la misma sala y en un espacio abierto. Eso sí, es importante también que exista algún lugar reservado para que las personas puedan mantener una conversación privada, por ejemplo. Esto no significa que, si el equipo no trabaja en el mismo sitio, no se pueda aplicar XP, pero habrá que prestar especial atención para que la comunicación dentro del equipo no se vea afectada.
- **El equipo completo:** El equipo debe estar compuesto por personas que tengan todos los perfiles necesarios para poder abordar el desarrollo con éxito y todos ellos deben sentirse miembros de un mismo equipo con espíritu colaborador. Por otro lado, se consideran miembros de este equipo todas aquellas personas que tengan algo que ver con el producto, ya sean clientes, usuarios, etc.

  Lo que se considera en XP “equipo completo” lógicamente es algo que va cambiando con el tiempo, ya que, dependiendo del momento, será necesario incorporar diferentes perfiles o hacerlo de forma parcial o temporal.

Figura 10.6. Equipo trabajando junto y de forma colaborativa.

- **Puesto de trabajo con información:** En el puesto de trabajo debe poder consultarse, ya sea en pósteres, paneles u otras herramientas, el estado del proyecto y las tareas que hay que realizar.

  Un equipo debe eliminar de su lugar de trabajo toda aquella información que no sea relevante para el proyecto y mantenerlo actualizado.
- **Energía en el trabajo o ritmo sostenible:** Se debe trabajar de forma que las personas sean productivas y estén centradas en su trabajo. Elimine los "ladrones de tiempo" de su jornada laboral. Por ejemplo, planifique cuándo responder al correo o al teléfono para evitar interrupciones continuas que impiden la concentración.

  Se debe encontrar un ritmo de trabajo que se pueda mantener en el tiempo y, en consecuencia, evitar que trabajar horas extras sea algo habitual y permitir que cada miembro del equipo pueda disfrutar también de su vida privada.

  • **Programación en parejas:** El código lo escriben dos personas en una única máquina. Esta práctica suele tener cierto rechazo cuando se plantea por primera vez, ya que, mal entendida, se interpreta como la pérdida directa de un recurso. Sin embargo, aquellos que la practican descubren rápidamente los enormes beneficios que supone para el proyecto. Por ejemplo, programar por parejas ayuda a alinear al equipo hacia el objetivo de la iteración y aumenta la calidad del código que se escribe. Programando por parejas se plantearán preguntas del tipo: "¿Por qué haces esto así?", "prueba a hacerlo de esta otra forma" o "¿De verdad hay que hacer esta funcionalidad?, nadie nos lo ha pedido".

  Uno de los mayores beneficios que aporta la programación en parejas es que el código y el conocimiento es compartido de manera muy eficaz entre varios componentes del equipo, con todo el beneficio que esto supone. Por otro lado, esta práctica ayuda a aclarar las ideas, ya que, mientras se explica la forma en que se trabaja y qué problema se tiene, muchas veces se va descubriendo sobre la marcha cómo resolverlo.

Figura 10.7. Trabajo en pareja.

Consejo:

Lo ideal es que las parejas cambien con mucha frecuencia, por ejemplo, una vez al día. Hay programadores que optan por hacer revisiones de código como alternativa a la programación por parejas y el beneficio es similar.

Recomendación:

Programar en parejas supone un gran esfuerzo y no debe hacerse durante toda la jornada.

Figura 10.8. Pareja de programadores.

- **Diseño incremental o evolutivo:** La propuesta de XP es ir diseñando de forma incremental al tiempo que se va construyendo el software, de manera que el diseño se enriquezca también constantemente con las aportaciones y comentarios del cliente y crezca y evolucione progresivamente. Un equipo debe refactorizar su código con frecuencia para que el diseño mejore constantemente.
- **Pruebas antes de programar:** Es necesario escribir pruebas para comprobar que el código funcionará correctamente. Si las pruebas se escriben antes de actualizar y añadir código nuevo al sistema, se evitará codificar en modo "*Cowboy Coding*", es decir, dejándose llevar y de forma rápida. Si se escribe la prueba antes del desarrollo, se obliga al desarrollador a centrarse en el problema y a confirmar que el diseño es el correcto. En los casos en los que el diseño no es el correcto, será muy difícil escribir estas pruebas y se pondrá en evidencia un problema al que hay que poner solución. Por otro lado, si construye de esta forma el código, es decir, si se asegura que funciona todo lo que se desarrolla, se conseguirá que los compañeros de equipo tengan gran confianza en el trabajo de los demás.

  El desarrollo dirigido por pruebas o *Test Driven Development* (TDD) es una técnica de desarrollo cuyo fundamento es escribir la prueba unitaria que debe cumplir el código y,

a continuación, escribir el código de manera que pase dicha prueba. Cuando la prueba se pasa con éxito, se debe refactorizar el código para simplificarlo y mejorarlo. Este proceso se repetirá tantas veces como sea necesario.

El efecto de programar así es que todo el código habrá pasado pruebas unitarias de forma automática y cada vez que se realice una modificación en el código se asegura que no se ha "roto" nada.

- **Integración continua:** La integración del software con todos los cambios realizados debería realizarse cada dos horas aproximadamente para evitar los problemas de integrar a última hora y con prisas. Cada vez que se modifique código y se tenga una nueva funcionalidad, debe recompilarse y probarse. De esta manera, se evitarán frases frecuentes del tipo "¡pero si en mi máquina sí que funcionaba!".

  Montar un entorno de integración continua implica un esfuerzo que se ve recompensado con creces a lo largo de todo el proyecto.
- **Construcción en diez minutos:** El sistema debe poder compilarse y probarse en diez minutos de forma automática para poder hacer este proceso con frecuencia y así recibir comentarios y sugerencias sin que impacte en el ritmo del desarrollo. Por otro lado, si la construcción de un ejecutable se realiza de forma tan rápida cada vez que un miembro del equipo modifica código, y el proceso falla, se podrá rectificar de inmediato, de manera que el resto del equipo no se verá afectado por dichos cambios.

Prácticas corolario

Tal y como hemos comentado antes, la aplicación de estas prácticas corolario debe realizarse cuando las prácticas primarias ya están incorporadas dentro de la rutina de trabajo de un equipo, ya que, mal utilizadas, pueden desencadenar graves problemas. Por ejemplo, veremos que una práctica corolario propone, en determinadas circunstancias, reducir el número de miembros de un equipo. Aplicar esta práctica puede ser peligroso si no se respeta el ritmo sostenible del trabajo, la programación por parejas o conocimiento compartido.

Las prácticas corolario son las siguientes:

- **Participación real de los clientes:** Toda persona afectada por el producto se considera parte del equipo y debe poder participar activamente en la planificación y en la elaboración de los requisitos. En ocasiones, es necesario que una única persona canalice todas las necesidades para transmitir una única petición al equipo de desarrollo.

Figura 10.9. Participación del cliente.

- **Despliegue incremental:** Cuando se trabaje sobre un producto que ya existe, empiece por modificar solo alguna de sus funcionalidades y, poco a poco, vaya modificando el resto. ¡Evite hacer todos los cambios de una sola vez!
- **Negociar el alcance del contrato:** Un contrato de desarrollo contempla habitualmente el precio, los plazos y la calidad esperada; sin embargo, el alcance debe ser negociado a lo largo del desarrollo. En ocasiones, es más útil negociar varios contratos de duración más corta y fijar de esta forma el alcance para cada uno de ellos.
- **Pagar por funcionalidad:** Habitualmente los clientes pagan por cada entrega de producto y lógicamente querrán que cada entrega contenga el máximo número de requisitos. Si se cambia el enfoque y se paga por funcionalidad realizada, se sabrá con precisión hacia dónde dirigir el desarrollo.
- **Continuidad de los equipos:** Un equipo de desarrollo debe mantenerse, aunque se cambie de proyecto. El conocimiento que comparten los miembros de un equipo es una información muy valiosa y es algo no se debería perder.
- **Reducir los equipos:** Un equipo será cada vez más productivo. Manteniendo la carga de trabajo, se podrá enviar a alguno de sus miembros a formar un nuevo equipo. En el caso en el que un equipo sea muy numeroso, se podrá dividir en varios equipos menos numerosos y centrados en diferentes funcionalidades del sistema.
- **Análisis de las causas:** Cuando aparezca un problema, elimínelo y, sobre todo, ataque a las causas que lo originaron. De esta forma, se soluciona el problema detectado y se evitará que se produzca de nuevo.

- **Código y pruebas:** Son los elementos clave del proyecto que no se pueden dejar de mantener y mejorar nunca. La documentación se podrá generar a raíz de estos dos elementos en el momento en que sea necesario.
- **Propiedad colectiva del código:** Cualquier miembro del equipo debe ser capaz de modificar cualquier parte del código. El código es de todos y no de alguien en particular. La programación en parejas y las revisiones de código apoyan notablemente a compartir el código. El adoptar esta práctica tiene la enorme ventaja de que, si se detecta un error, cualquier miembro del equipo que esté disponible podrá corregirlo. También, si en un momento dado un desarrollador está de vacaciones o enfermo, otro compañero podrá continuar con el trabajo sin tener que modificar los compromisos para la iteración.
- **Código base único:** En el caso en que, por algún motivo, se necesite trabajar en paralelo al desarrollo principal, esto no deberá prolongarse más de varias horas. Es decir, deberá evitarse tener que mantener diferentes programas que compartan parte de su código, ya que es algo que provoca numerosos fallos y es muy costoso de mantener sin errores.
- **Despliegue diario:** Cada día se debe poner el código desarrollado en producción para evitar que haya un desfase entre la última versión de software y lo que hay en cada máquina local.

  Esta práctica no es primaria, ya que, para poderla realizar, es necesario, entre otras cosas, que el número de defectos detectados en el sistema sea muy bajo y que este esté automatizado en gran medida. El despliegue diario es una práctica difícil de alcanzar, pero al menos debe empezarse por tratar de aumentar progresivamente el número de veces que se realizan los despliegues y no limitarlo al de “una vez al año”.

Combinando Scrum y XP

Imaginemos que estamos estudiando violín en el conservatorio y al acabar el primer año nos regalan un Stradivarius. La alegría sería inmensa porque, al tocarlo, comprobaríamos que el sonido es diferente a nuestro violín barato, pero el tener el maravilloso violín entre las manos no nos convertiría en maestros. Imaginemos ahora que al mejor violinista actual le prestamos nuestro básico violín de aprendizaje. Desde luego que la música que escucharíamos sería mucho más agradable que la del alumno de primer curso, pero distaría mucho del sonido habitual al que el maestro nos tiene acostumbrados en los conciertos cuando toca “su” violín. Lo mismo ocurre con *Scrum* y XP. *Scrum* es la herramienta para gestionar nuestro proyecto, “nuestro Stradivarius”, y XP es el oficio, “nuestro concertista”. Necesitamos combinar ambas herramientas para que el resultado sea extraordinario.

Figura 10.10. El maestro combina su conocimiento con un buen instrumento.

Scrum y XP son dos métodos basados en los valores y principios *Agile* y coinciden en su filosofía de trabajo. Ambos métodos comparten también numerosas prácticas que un equipo de desarrollo debería aplicar a su trabajo diario para mejorar no solo desde el punto de vista de la productividad, sino también desde un punto de vista más personal.

La aplicación tanto de *Scrum* como de XP en la creación de un producto implica un cambio de mentalidad en todos y cada uno de los actores implicados. Es necesario dejar a un lado ideas y patrones utilizados en el pasado y dar un giro para buscar la mejor forma de trabajar y la manera de protegerse de las interferencias externas que puedan afectar a la productividad de los equipos.

Se puede afirmar, sin correr demasiados riesgos a equivocarse, que *Scrum* y XP pueden convivir en armonía en el caso en el que el proyecto que se esté llevando a cabo contenga un desarrollo de software. El gran objetivo de XP es desarrollar software con menos defectos, más barato, de forma productiva y con un mayor retorno de la inversión realizada. *Scrum* trata de optimizar la forma en que los equipos organizan y gestionan su trabajo.

Nota:

Scrum es un método ágil que propone un conjunto de buenas prácticas para la gestión, mientras que XP lo que sugiere son una serie de buenas prácticas de programación. Por este motivo, ambos métodos no solo pueden convivir, sino que se complementan el uno al otro.

Algunos de los aspectos que comparten *Scrum* y XP y que hacen que su convivencia sea posible en perfecta armonía son los siguientes:

Figura 10.11. Scrum y XP comparten aspectos que hacen posible que convivan en armonía.

- **Revisión del trabajo:** Ambos métodos se basan en revisar de forma periódica y frecuente el trabajo realizado. Estar dispuestos constantemente a ver qué se puede cambiar y mejorar en todos los niveles del trabajo y llevarlo a la práctica. Ejemplos de ello son la programación por parejas, la automatización de las pruebas (y no solo unitarias), las reuniones diarias, las Retrospectivas y la revisión del *Sprint*.
- **Colaboración:** Una buena relación entre las personas que están trabajando con un objetivo común lleva consigo buenos productos. La productividad y la calidad de lo creado depende de las técnicas utilizadas y de las relaciones personales de aquellos que las aplican. Compartir necesidades y éxitos, problemas y triunfos repercute muy directamente sobre el trabajo realizado. Tal y como dijo Kent Beck, se trata de madurar y dejar atrás el pasado adolescente en el que se piensa que “sé más que nadie y lo único que necesito es que me dejen solo para ser el mejor”. Se trata de buscar “tu sitio” dentro del grupo de trabajo, compartiendo tanto lo que creo como la forma en la que lo estoy construyendo.
- **Implicación directa del cliente:** Por muy bien que se gestione un equipo y por muy excelente que sea el código que produce, si no es exactamente lo que el cliente necesita, de poco servirá. Por este motivo, la colaboración constante con él es clave para asegurar en todo momento que se está construyendo con calidad y de manera

eficaz lo que realmente se está esperando.

- **Simplicidad:** Hacer exactamente lo que se necesita y no perder tiempo haciendo algo que no aporta valor. El tratar de simplificar procesos, código y cualquier aspecto que afecte al trabajo aporta un valor directo y casi inmediato a la productividad. Por ejemplo, *Scrum* limita el tiempo de las reuniones para evitar el "análisis-parálisis" y trata de que toda la gestión asociada a un proyecto se optimice y se reduzca exclusivamente a lo que es útil. La práctica de simplicidad en el código de XP sería su equivalente desde el punto de vista de la programación. En las reuniones, trate exclusivamente los temas importantes y concretos que afectan al siguiente *Sprint*. Programe exactamente lo que el código debe hacer. Ni más, ni menos. Elimine todo lo accesorio.
- **Trabajo en iteraciones o ciclos cortos:** De esta forma se asegura la mejora continua y la obtención de comentarios y sugerencias de manera rápida, eficaz y frecuente. Trabajar con iteraciones permite, a su vez, volver a planificar en función de las necesidades de cada momento. Las iteraciones de XP pueden convivir con facilidad con los *Sprints* de *Scrum*.
- **Respeto:** Sin respeto por los demás y por su trabajo, todo lo demás carece de sentido y ninguna práctica de XP ni de *Scrum* podrá aplicarse, ya que ambos métodos se basan en la profesionalidad y confianza entre los integrantes de los equipos.
- **Valentía:** Aplicar estos métodos implica en muchas ocasiones romper con lo establecido y, de alguna manera, con la tradición. Tal y como hemos comentado anteriormente, es necesario ser valiente para cambiar la forma de trabajar y relacionarse.
- **Priorización:** Tanto *Scrum* como XP necesitan unos requisitos bien elaborados y con una prioridad bien establecida por parte del cliente para asegurar que se está construyendo lo que interesa en cada momento.

Recuerde:

Tanto Scrum como XP basan su filosofía de trabajo en que todos los equipos, incluso los mejores, independientemente de sus circunstancias, siempre pueden mejorar en algún aspecto.

---

42 http://c2.com/cgi/wiki?HistoryOfExtremeProgramming.

43 *eXtreme Programming Explained: Embrace Change*. Kent Beck. Ed. Addison Wesley. (1999).

44 *eXtreme Programming Explained: Embrace Change (2nd Edition)*. Kent Beck with Cynthia Andres Ed. Addison Wesley (2004).

45 *The Art of Agile Development*. James Shore & Shane Warden. O´Reilly. 2008.

46 *eXtreme Programming Explained*: Embrace Change. Kent Beck with Cynthia Andres (2004).*

11 Scrum en la empresa

En este capítulo aprenderá:

- Cuál ha sido la evolución de los métodos ágiles en la empresa.
- Cuáles han sido las principales implicaciones de implantar *Scrum*.
- Qué son los contratos ágiles.

Los métodos ágiles no pueden verse como algo aislado y separado en el conjunto de la estructura de una empresa. Aunque parezca que se dirigen únicamente a los equipos de trabajo, sus implicaciones van mucho más allá y afectan a la gestión de recursos humanos, a la comunicación, a la formación, las relaciones y la propia cultura de la empresa. Las organizaciones deben buscar la mejor manera de encajar los métodos ágiles para sacar el máximo partido de su aplicación.

Se trata de un tema muy extenso tratado por muchos autores por lo que, en este capítulo, solo podremos dar una visión general del tema. Trataremos, sobre todo, tres aspectos: una breve introducción a los métodos ágiles en la empresa y su historia; qué supone implantar *Scrum* a gran escala en una organización; y explicaremos brevemente qué son los contratos ágiles, posiblemente la expresión más extrema y complicada del mundo ágil aplicado a la empresa.

Scrum en la empresa

Los métodos ágiles, especialmente los orientados al desarrollo de nuevos productos y particularmente software, son una alternativa al clásico modelo en cascada o *waterfall*. Este modelo, muy asentado aún hoy, se basa en una serie de premisas asumiendo que:

- Hay una serie de requisitos que definen por completo y de antemano el nuevo producto.
- Estos requisitos permanecen estables y si hay cambios son menores y no afectan al proyecto.
- Si hay necesidad de integrar sistemas o componentes, es un proceso predecible y controlado.
- La innovación y desarrollo necesarios para un nuevo producto son predecibles.

El problema es que estas premisas son esencialmente falsas:

- Rara vez los requisitos están completos y perfectamente descritos al inicio. Suelen contener ambigüedades, lagunas y errores. Requieren habitualmente explicaciones adicionales y aclaraciones.
- Los requisitos varían continuamente y ese cambio aumenta a medida que pasa el tiempo desde que se enunciaron hasta que se concluye el producto.
- Y, en general, la naturaleza impredecible de este tipo de actividades hace inviable las dos últimas premisas. Los sistemas y componentes acaban interaccionando de formas inesperadas e impredecibles y el proceso creativo asociado a la innovación no puede

anticiparse ni pronosticarse.

Las empresas que adoptan métodos ágiles como *Scrum* están tratando de atajar algunos de estos problemas y convertir la incertidumbre en un elemento natural del proceso.

Nota:

Entiéndase que no estamos hablando de LEAN, la fabricación JIT o técnicas similares, sino de otros métodos mencionados en el libro como Scrum, Kanban o XP. Estos últimos no se refieren tanto al proceso de fabricación y producción masiva como a trabajos donde el diseño tiene un papel central, ejemplificados por el desarrollo software, pero que abarcan actividades como la arquitectura, el diseño industrial, el marketing, banca, proyectos de ingeniería, etc.

La adopción de estos métodos ágiles supone una serie de beneficios que no se pueden ignorar, entre los que destacan[47]:

- Los clientes pueden ver resultados coherentes y funcionales casi desde el primer momento, en lugar de esperar al final del proceso; además, esos resultados crecen progresivamente con cada iteración.
- Los clientes también ganan en control y capacidad de influir en el proceso de forma continuada. Es una manera de poder modificar rápidamente sus requisitos ante cambios del entorno o condicionantes. El proceso es más eficiente, ya que reduce la cantidad de esfuerzo invertido en actividades que no generan verdadero valor.
- Por el lado de la empresa, los métodos ágiles permiten contar con equipos motivados, auto-organizados y comprometidos con la calidad y la mejora continua.
- El proceso se vuelve más controlable, medible y cuenta con una forma inequívoca de determinar el cumplimiento de las expectativas de los clientes.
- Hay un control más cercano e inmediato que permite detectar mucho antes puntos de mejora y facilita un uso más eficiente de los recursos.

Sin embargo, aunque es una fuente de toda clase de beneficios y ventajas para el conjunto de una organización, la implantación de los métodos ágiles no puede hacerse de forma descuidada y sin orden. Podría acabar volviéndose contra quienes lo impulsan y, de hecho, hay referencias a empresas que han abandonado *Scrum* o Kanban por malas experiencias iniciales.

Afortunadamente, hay autores que se han preocupado de diseñar una estrategia para ello. Todas las empresas y organizaciones siguen un esquema parecido: la semilla de *Agile* aparece en forma de algún proyecto o actividad pioneros, al que poco a poco se van sumando algunos más. En otros casos, es un conocimiento que se extiende sin que se llegue a aplicar

formalmente, pero que se infiltra en la cultura de la compañía.

De una u otra forma, la Dirección de la organización, ahora consciente de la existencia de los métodos ágiles, plantea una prueba controlada. Si el resultado es satisfactorio, la aplicación de estos métodos continuará extendiéndose. Además, como *Scrum* y otros métodos ágiles son bastante eficientes descubriendo fallos y destacando impedimentos, se adoptan nuevas técnicas de calidad que hacen que la brecha entre proyectos ágiles y convencionales sea cada vez más amplia y se demuestren los beneficios de la nueva forma de trabajo.

Para completar de la forma menos traumática la adopción generalizada de métodos ágiles, hay que trazar una estrategia que los introduzca progresivamente. Hay muchos ejemplos de este tipo de estrategias. Nosotros nos fijaremos en una[48], que define tres pasos:

- **Piloto:** En realidad, ya se ha descrito esta fase: un pequeño grupo de proyectos se llevan a cabo como una prueba controlada y supervisada para validar lo acertado de usar estos métodos. Por ello, normalmente son proyectos pequeños y, a ser posible, a clientes internos. En esta primera etapa, muchas de las cuestiones que aparecen son de carácter técnico (herramientas, metodologías, procedimientos, fijación de estándares) y aparecen las primeras dudas de naturaleza organizativa.
- **Lanzamiento:** Si se superan las primeras reticencias y dudas, llega el momento de extender el uso de los métodos ágiles, lo que tiene consecuencias ya claramente organizativas. Por ejemplo, es el momento de fijar unos estándares y procedimientos propios para la empresa: terminología, herramientas, métricas. También hay acciones formativas para introducir y difundir el conocimiento y las primeras implicaciones desde el punto de vista de gestión, como idear criterios de selección de proyectos para ser abordados con estos métodos.
- **Generalización:** Lleva más tiempo y es mucho más compleja que las anteriores. Ahora los métodos se aplican en todo tipo de proyectos. Es el momento de hacer una comunicación general del uso de estos métodos, de ajustar las políticas de selección, compensación, roles y promociones para que encajen en la nueva forma de trabajo, crear un sistema de aprendizaje con *coaching* y revisar el sistema de financiación de estos proyectos.

Afortunadamente, hay una serie de componentes de los métodos ágiles que escalan con facilidad a lo largo de una empresa, simplificando su difusión[49]:

- **Equipos con capacidad de diseño, creación y pruebas:** El modelo de equipo multidisciplinar, con fuerte presencia y relación con el cliente, puede crecer, ampliarse y abarcar áreas cada vez más amplias de una empresa.
- **Planificación estructurada:** A pesar de eludir la planificación a largo plazo, los métodos ágiles son muy disciplinados y crean pequeñas iteraciones con objetivos y

alcance muy claros que generan un producto de crecimiento continuado y con calidad.

- **El énfasis en la calidad:** Al generar en cada iteración un producto consistente y probado, la calidad entra a formar parte del proceso. Siguiendo el espíritu de Toyota, "la calidad se fabrica", en lugar de simplemente validarla al final del proceso.
- **Revisión y mejora continua:** Al final de cada iteración, el equipo examina su forma de trabajar e identifica puntos de mejora. Este principio se puede extender a toda una organización y aplicarse sobre equipos y áreas cada vez más grandes.

Sin embargo, la parte más compleja del proceso de adopción de métodos ágiles en una empresa es la que se refiere a los cambios en la propia organización. Y eso se debe a que la filosofía del trabajo ágil choca con algunos de los principios establecidos hasta ahora. Por ejemplo, los métodos ágiles priman el desarrollo empírico de productos frente al planificado o deja en manos de equipos motivados y auto-organizados potestades que antes estaban en manos de gestores.

Eso significa que los roles y jerarquías deben adaptarse a la nueva situación, lo mismo que la formación, la política de reclutamiento y retribución y, en general, buena parte de los aspectos más destacados de la gestión de una empresa. Además de cuestiones internas, hay un aspecto externo crítico que también debe ser revisado: la relación con los clientes. Si hablamos de proyectos cuyos requisitos se construyen iteración a iteración, sin unos requisitos fijados al principio, abiertos a cambios y con un equipo que fija sus propios objetivos, parece complicado realizar una contratación con estas premisas. Por ello, vamos a dedicar el último apartado de este capítulo a los contratos ágiles, un tema rodeado de cierta polémica.

Contratos ágiles

Si los métodos ágiles establecen una nueva forma de trabajar y de relación con los clientes, una de las expresiones de esa relación, los contratos, también se verá afectada. Se abandona la orientación tradicional en la que el trabajo está perfectamente definido antes de empezar, por otra aproximación en la que se orienta hacia el cambio y facilita la introducción continua de nuevos requisitos.

La expresión más extrema del contrato ágil se resume en la frase: "*Money for nothing, change for free*"[50] (dinero a cambio de nada, modificaciones gratis), que no deja de ser una forma de definir la confianza mutua, la que tiene el cliente en el equipo al pagar sin un contrato formal que defina detalladamente lo que va a recibir al final del proceso, y la del equipo, que realiza los cambios que se le soliciten sin exigir una evaluación económica a cada paso.

Como se ve, es una filosofía completamente distinta a la de un contrato convencional. Los

contratos son más una herramienta de protección que la definición de una relación de confianza. El cliente no espera recibir nada menos, ni el proveedor ofrecer nada más, y el cambio se gestiona, cuando se hace, a través de complejos mecanismos que buscan la protección antes que la colaboración. Al final, se corre el riesgo de pagar más de lo que se preveía y recibir menos de lo que se esperaba.

Lo cierto es que un contrato debería ser un acuerdo y potenciar que ambas partes salgan beneficiadas, más que proteger a una frente a otra. Un contrato debería reflejar una colaboración y no ser un arma entre las partes.

El acuerdo se fija ante todo en tres dimensiones: alcance (lo que incluye calidad), coste y plazos, y, como mucho, se establecen mecanismos para que la variación de una no afecte a las demás. Sin embargo, en un contrato ágil, un cambio debe afectar a las restantes dimensiones: si cambia el alcance debe variar el tiempo y/o el coste, es inevitable, o la calidad se verá comprometida.

Figura 11.1. Los contratos ágiles se basan en la confianza y colaboración entre cliente y proveedor.

Una forma de amortiguar el impacto del cambio es ser capaces de definir partes del producto imprescindibles, junto a otras de prioridad menor. Trabajar en bloques pequeños y ciclos cortos, propio de los proyectos ágiles, ayuda a reducir la complejidad y permite asumir cambios como algo natural.

Otra premisa importante de los contratos ágiles es asumir que los riesgos son compartidos,

para evitar que el contrato se convierta en un arma arrojadiza. Si no se comparten riesgos en lo que afecta a costes, expectativas y plazos, se reduce la posibilidad de obtener un beneficio mutuo.

En un contrato convencional, que busca cubrir a las partes, se trata de contar con unos requisitos cerrados de antemano y una aceptación única final. Eso hace que se trate de tener la mayor cobertura posible haciendo que se incluyan requisitos "por si acaso", aunque no sean realmente necesarios, lo que da lugar a obtener un producto con funcionalidad que no se necesita y a un desperdicio de recursos.

Tampoco se puede afirmar que un contrato ágil sea necesariamente una operación arriesgada. Lo que se busca con estos contratos es llegar a trabajar como socios, no en una relación cliente-proveedor.

Lo cierto es que todo este proceso se asienta en la confianza y, cuando se inicia una relación entre dos partes, no se puede dar por hecho que esa confianza esté presente. Debe ganarse poco a poco y ser muy cuidadoso a lo largo de la relación que se establezca. Por ese motivo, hay bastantes modelos de contratos ágiles[51], que recogen todo el rango de graduación de confianza entre las partes[52]. En un extremo, estaría el "contrato por *Sprint*", que no es más que el acuerdo entre *Product Owner* y equipo sobre las funcionalidades de cada iteración. Pero, a partir de ahí, tenemos otros tipos de contratos posibles, que se exponen a continuación de forma muy resumida:

- Fijando precio y alcance, pero dejando abiertas el resto de las variables. En realidad, el riesgo pasa al proveedor y a su capacidad de estimar.
- Tiempo y materiales, que supone pagar por unos recursos en un tiempo, aunque dejando abierta la funcionalidad. En este caso, es el cliente el que asume todo el riesgo, ya que no hay compromiso de alcance. Hay variantes para mitigar ese riesgo, como, por ejemplo, establecer límites de coste o acordando un alcance por iteración y permitiendo al proveedor cobrar el importe íntegro, como incentivo, si se consigue cumplir antes.
- Desarrollo en fases. Una aproximación cooperativa en la que se establecen *releases* que limitan el riesgo mientras el resto de los factores se acuerdan entre las partes. Puede complementarse con un sistema de penalizaciones.
- Beneficio fijo. Se establece un beneficio y un coste fijos. Si se obtiene el alcance en fechas, el proveedor recibirá ese beneficio; si no es así, el cliente seguirá pagando el esfuerzo, pero esta vez no habrá beneficio posible para el proveedor.
- "*Money for nothing, change for free*"[53]. La expresión más extrema de los contratos ágiles. El cliente paga por el esfuerzo, pero el compromiso del proveedor es total. Aparentemente, el cliente paga por un alcance que no está completamente cerrado, pero el proveedor se entrega al proyecto y asume todo el esfuerzo preciso para completar el alcance comprometido en cada iteración, así como para respaldar la

calidad del producto. Exige un grado de confianza máximo, requiere una relación muy madura entre ambos y solo funciona si ambas partes trabajan siguiendo metodologías ágiles.

La forma de protegerse de las incertidumbres en los contratos convencionales es crear márgenes y colchones adicionales que vician el proceso y encarecen el producto. En los contratos ágiles, esas prevenciones se quedan en detalles como considerar al *Sprint* 0 por separado o tener partes estables y acordadas y un porcentaje cambiante. La idea tras los contratos ágiles es establecer una relación de confianza.

Uno de los componentes básicos de un contrato ágil es reflejar claramente que habrá revisiones regulares del producto en ciclos cortos. Es decir, definir el concepto de *Sprint*. Otro es la inclusión de una lista de funcionalidades básicas y prioritarias, y otra de secundarias con las que se puede "jugar" a lo largo del desarrollo.

La existencia de esa segunda lista se justifica con la inclusión del concepto de cambios con consecuencias: se aceptan, pero supondrán cambios en otra variable como coste o tiempo, o reduciendo alcance (cambiar unas cosas por otras). Es conveniente que, a lo largo del proyecto, se diferencie entre el conjunto de características de partida y las añadidas o modificaciones, para poder distinguir claramente lo que es una cosa y la otra en cada revisión. Esa información va a ser muy beneficiosa para ambas partes.

Por supuesto, hay que incluir algún mecanismo de limitación: los productos y proyectos no pueden ser indefinidos e infinitos.

Los contratos ágiles son un tema complejo, extenso y muy debatido. Con esta introducción no se pretende ni mucho menos agotarlo. La principal idea que se quiere transmitir y destacar es que el proceso de implantación de *Scrum* no puede ser algo aislado: a medida que se extiende en una organización, acaba dando lugar a consecuencias que van más allá de la organización del trabajo de un proyecto y su culminación es la introducción de los contratos ágiles.

---

47 *Top Eight Reasons Why Organizations Are Making the Switch*, S. Elatta, Scrum Alliance.

48 *An Enterprise Strategy for Introducing Agile*, Kane Mar, 2006.

49 *Scaling Software Agility*, Dean Leffingwell, Addison Wesley, 2007.

50 Acuñada por Jeff Sutherland (), uno de los firmantes del manifiesto *Ágil* y una de las grandes referencias mundiales en métodos ágiles.

51 *10 Contracts for your next Agile Software Project*, Peter Stevens, Agile Software, , 2009.

52 Xavier Albadalejo, .

53 Jeff Sutherland, *Agile Contracts: Money for Nothing and Your Change for Free*, .

12 Escalando Scrum

En este capítulo aprenderá:

- Cómo coordinar y trabajar con equipos grandes y distribuidos.
- La utilidad de crear un equipo de *Product Owners*.
- La dinámica de las reuniones de *Scrum of Scrums*.
- A gestionar dependencias entre equipos.

Estamos convencidos de que Scrum funciona

Un equipo que aplica *Scrum* tiene claro el objetivo que quiere conseguir. La revisión diaria del trabajo, la búsqueda de la mejora y la coordinación continua hace que se puedan obtener resultados visibles con mucha frecuencia. El equipo asume responsabilidades claras y concretas, a la vez que establece las reglas de gestión que resultan más sencillas.

Scrum es un método que, en origen, está concebido para que los equipos tengan total libertad para innovar y experimentar iteración tras iteración. Parte de la potencia de adoptar este método reside en que la mayoría de las necesidades que le vayan surgiendo al equipo y la resolución de sus problemas puedan gestionarse de forma muy sencilla por el mismo equipo sin depender de terceros.

Imaginemos ahora una familia compuesta por cinco personas. El padre es un excelente cocinero y durante años se ha encargado de cocinar exquisitos platos para su familia. Todos le animan a montar un pequeño restaurante, pero él no está seguro de si será capaz de poder cocinar para tanta gente. Su familia le recuerda que no piense que tiene que hacerlo todo él solo. El hijo se ofrece para ir a la compra, la mujer será la encargada de pensar los menús, el padre cocinará y entre todos los demás atenderán las mesas. En definitiva, formarán un equipo. Desde luego la forma de organizarse nada tiene que ver con el día a día en su cocina doméstica, pero ¡claro que es posible montar el restaurante!

Figura 12.1. El reto de atender a muchas personas simultáneamente complica la logística.

Tal y como hemos visto en los capítulos anteriores, *Scrum* es ideal para proyectos pequeños o medianos y que puedan llevarse a cabo por equipos compuestos entre 4 y 12 personas.

¿Pero qué ocurre si tenemos un proyecto grande que debe ser llevado a cabo por un equipo numeroso de personas? ¿Cómo podríamos abordar un proyecto que necesitara coordinar el trabajo de 50, 100 o más personas y contemplar un gran número de requisitos?

Figura 12.2. Un equipo con muchas personas complica la coordinación.

Cuando un proyecto es grande, no basta simplemente con añadir más personas al equipo.

Un equipo tan numeroso de personas hace que la comunicación y la coordinación entre ellas sea complicada. Es materialmente imposible que todos y cada uno de los miembros del equipo puedan conocer en detalle el trabajo del resto de sus compañeros. La comunicación se deteriora, ya que no hay forma de escuchar y sentirse escuchado por todos los miembros de un equipo tan grande. Paralelamente, la gestión del trabajo se complica de forma notable. Las *Daily Meetings* durarán mucho más tiempo y dejarán de tener el valor. El *Product Owner* no podrá gestionar correctamente el enorme *Product Backlog* ni atender a las necesidades de todos los miembros del equipo como debería.

La duración de los *Sprint Planning* se alargará enormemente y se tratarán numerosos asuntos que no afectarán a una gran parte del equipo y un largo etcétera de inconvenientes.

Desde el punto de vista organizativo, los responsables de la empresa necesitan disponer de una foto global del estado del proyecto actualizada y, a la vez, conocer detalles del avance del equipo, pero ¿cómo se puede obtener esta información sin causar demasiado impacto en el

trabajo de los equipos?

Es un reto utilizar *Scrum* en un proyecto grande, sin embargo, los beneficios pueden ser muchos.

A la hora de escalar *Scrum*, es decir, de aplicarlo a proyectos que necesiten llevarse a cabo por equipos con muchas personas y gestionar un gran número de requisitos, se plantean dos retos. Un primer reto es la adaptación de *Scrum* a esa situación especial, ya que, como se ha comentado en los capítulos anteriores, *Scrum* está optimizado para proyectos con dimensiones mucho más pequeñas tanto de personas como de requisitos. El segundo reto que se plantea es el de ser capaces de responder a las necesidades propias de la organización de la empresa. Existe una técnica que ayuda a manejar esta situación. Es la llamada **Scrum of Scrums**.

Veamos en detalle, a continuación, en qué consiste exactamente *Scrum of Scrums* y algunos consejos para implementarlo con éxito.

Aplicando Scrum con equipos grandes y distribuidos

Dentro de una empresa, tiene sentido aplicar *Scrum* en diferentes equipos que trabajan en proyectos independientes, pero ¿qué hacemos si tenemos un equipo muy grande trabajando sobre un único producto? La propuesta consiste en trabajar con varios equipos pequeños, pero, por supuesto, deben trabajar siempre coordinados. Para ello, será necesario escalar el rol del *Product Owner*, manejar correctamente un *Product Backlog* más grande de lo habitual, gestionar las dependencias entre los equipos y escalar las reuniones de planificación del trabajo.

La propuesta para aplicar *Scrum* en estas circunstancias es la siguiente:

- Divida al equipo en varios equipos más pequeños.
- Cree un equipo de *Product Owner* (solo si fuera necesario).
- Mantenga un único *Product Backlog*, ya que tenemos un único producto.
- Coordine el trabajo entre los equipos y gestione sus dependencias: *Scrum of Scrums*.

Veamos todo esto con algo más de detalle a continuación.

Los equipos

Esta es, sin duda, la decisión más delicada cuando se escala *Scrum*. ¿Cuántos equipos formo? ¿Cómo divido el trabajo? ¿Qué personas incluyo en cada equipo? Como ya hemos comentado en varias ocasiones con anterioridad, los equipos que trabajan aplicando *Scrum*

deben estar compuestos, en general, por un número de personas que oscile entre 4 y 12. Así que el primer paso será organizar los nuevos equipos pequeños y repartir el trabajo.

Una muy buena opción a la hora de asignar el trabajo es hacerlo por bloques de historias de usuario completas. De esta forma, cada equipo será capaz de completar la funcionalidad sin depender de ningún otro grupo.

En ocasiones, puede ser necesario montar un equipo especializado enfocado a un asunto en concreto, pero esto debe responder a necesidades puntuales, ya que, para completar el producto, necesitarán la colaboración del resto de equipos y se perderá la rapidez de respuesta que caracteriza a *Scrum*.

Dicho esto, continuaremos este capítulo optando por formar grupos que puedan completar historias de usuario de forma independiente.

Cada uno de estos equipos aplicará *Scrum* y tendrá un *Scrum Master* y un único *Sprint Backlog* priorizado y estimado. Realizará sus reuniones de sincronización diarias, *Reviews*, Retrospectivas, etc.

Sin embargo, no se debe olvidar que su trabajo forma parte de la elaboración de un producto más grande y que su labor consiste solo en la creación de una parte de un todo. Cada uno de los equipos debe conocer la planificación global del producto, el estado del trabajo del resto de los equipos y las posibles dependencias con ellos.

Importante:

En ningún caso debe permitirse que un equipo tenga más de un Product Owner, ya que esto podría derivar en conflictos a la hora de establecer prioridades en el trabajo del equipo.

El equipo de Product Owner

El rol del *Product Owner* es de los más complejos en un proyecto.

Ya hemos visto en capítulos anteriores que el *Product Owner* tiene muchas responsabilidades tanto hacia el equipo como hacia el exterior: participa en las planificaciones, asiste a las revisiones de los *Sprints*, mantiene los requisitos actualizados y priorizados en un único *Product Backlog* y está disponible para resolver las dudas del equipo durante toda la iteración. Por otro lado, el *Product Owner* habla con los usuarios sobre sus necesidades, negocia los requisitos de los clientes, analiza las tendencias del mercado y un largo etcétera.

En un proyecto con un solo equipo esto es mucho trabajo, pero asumible. En el caso de un proyecto con varios equipos, las responsabilidades del *Product Owner* son demasiadas para una sola persona y este es el motivo por el que se debe escalar su rol. Mike Cohn[54] recomienda que lo ideal es asignar un *Product Owner* para cada equipo, pero, si esto no fuera

posible, hay que tratar de que nunca una sola persona sea responsable de más de dos equipos.

La propuesta es crear un equipo de *Product Owners* formado por las personas responsables de mantener un único *Product Backlog*. Si el número de equipos es numeroso, será necesario crear algún tipo de jerarquía dentro de este equipo de *Product Owners.* Es importante que exista una persona "responsable de los *Product Owners*", cuya misión sea coordinarlos y elaborar la estrategia general del producto, analizar el mercado, etc. Como es lógico, no podrá estar trabajando tan próximo a los equipos de desarrollo, pero deberá mantener el contacto con ellos, aunque sea de forma esporádica, para transmitir actualizada la visión global del producto a los equipos y para conocer de primera mano el estado de la construcción del mismo.

Esta persona "Responsable de los *Product Owners*" guiará al resto de los *Product Owners* y facilitará su comunicación para que realicen una correcta gestión de posibles dependencias.

La organización y composición de este equipo dependerá de las necesidades de cada producto en concreto.

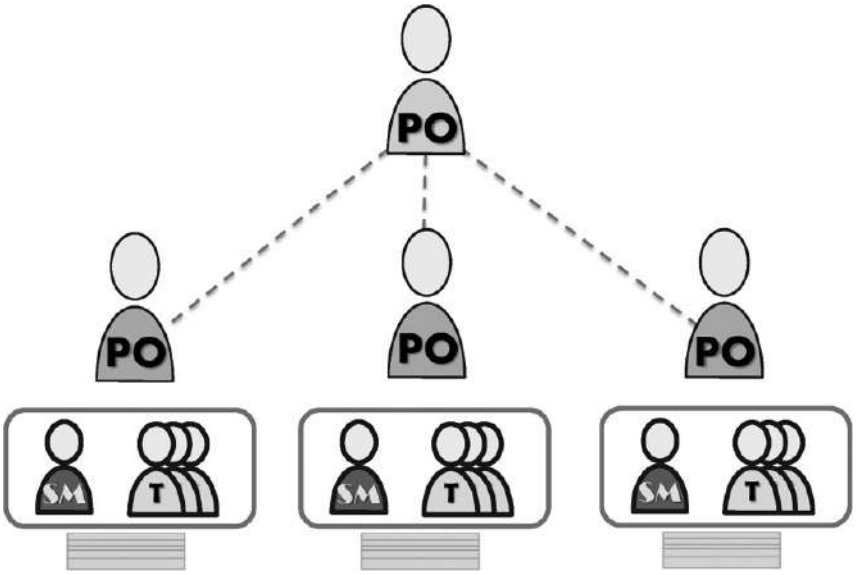


Figura 12.3. Equipo de Product Owners.

Importante:

El equipo de Product Owners al completo debe asistir a las Reviews de todos los equipos para tener una visión global del producto y poder continuar trabajando con el Backlog del producto para gestionar dependencias y prioridades.

Un único producto, un solo Product Backlog

Estamos hablando de que nuestro proyecto es grande y necesitamos, para construirlo, un grupo numeroso de personas. Ya se ha propuesto crear equipos reducidos que trabajen de

forma independiente pero coordinada con el resto de los equipos. Es crucial no olvidar que se está construyendo un único producto y no caer en la tentación de crear varios *Product Backlogs*. El hecho de que haya un único *Backlog* de producto tiene muchas ventajas. De esta forma, se podrán detectar de forma temprana las relaciones entre las historias de usuario de los diferentes equipos y se podrán tomar las medidas necesarias para construirlas de forma satisfactoria.

Por otro lado, el tener una única pila de producto permite tener una visión global del mismo y es más sencillo realizar un plan de trabajo para los diferentes equipos. También simplifica notablemente la elaboración del plan de entregas.

Si se mantienen *Backlogs* separados, es más fácil perder esta conexión entre los equipos y el producto global. Además, la gestión de prioridades se complica, ya que puede darse el caso de que una historia de usuario con baja prioridad en un *Backlog* sea más prioritaria en el producto global que la historia de usuario más prioritaria de otro de los *Backlogs*, qué lío, ¿no? Pues este lío se soluciona teniendo un único repositorio donde trabajar con los requisitos.

Para que este *Product Backlog* único sea manejable hay una serie de condiciones "extra" que debe cumplir.

La primera condición es que el *Product Backlog* debe tener un tamaño razonable y que no suponga un esfuerzo inmenso trabajar con él. No debe aceptarse como excusa el que, por tratarse de un producto grande, sea necesario manejar un *Product Backlog* infinito.

Es necesario mantener el equilibrio entre un tamaño razonable sin que esto suponga sacrificar el que sea completo.

Es de gran ayuda trabajar con temas y épicas, tal y como se comentó en el capítulo 4. Se deben detallar únicamente las historias de usuario más prioritarias y no bajar tanto de nivel las que lo son menos.

Otra condición importantísima para trabajar con este tipo de *Backlogs* es que se puedan realizar vistas parciales del mismo. De esta forma, se podrán filtrar los ítems que interesen a cada equipo en su trabajo del día a día sin que se "contaminen" con demasiada información no interesante en ese momento del ciclo.

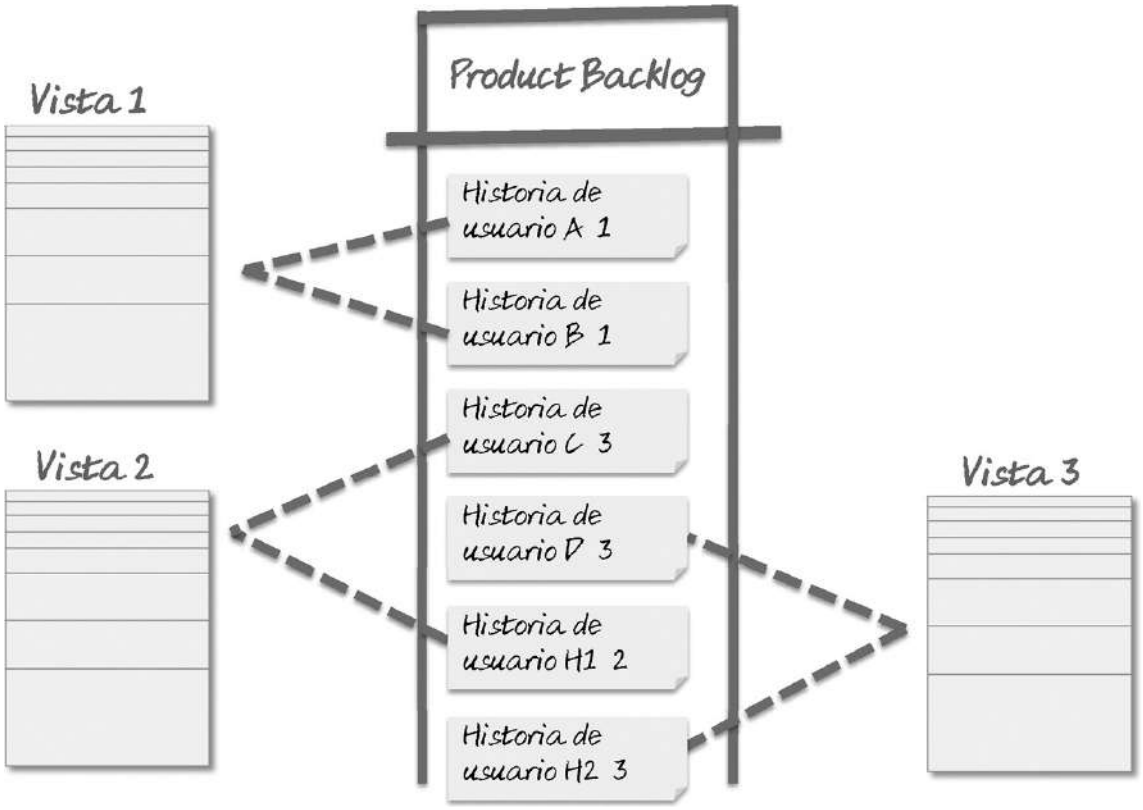


Figura 12.4. Vistas del Product Backlog.

Hemos insistido en la importancia de mantener un único *Backlog* de producto. Sin embargo, cada equipo manejará su *Sprint Backlog* de forma independiente. Una vez que cada equipo seleccione los ítems del *Backlog* completo en los que se va a trabajar durante la iteración, se centrará en la estimación y división en tareas, al igual que lo haría en caso de ser un equipo aislado.

Scrum of Scrums

Tal y como se ha comentado, existe el riesgo de que cada uno de los equipos se centre en "su" *Sprint Backlog* y en "su" objetivo para el *Sprint* y acabe desconectándose del resto de los equipos.

Esto no significa que los equipos no estén haciendo bien su trabajo. Aunque los miembros de cada uno de los equipos sean unos excelentes profesionales, se corre el riesgo de que el objetivo global no se cumpla.

Para minimizar este riesgo, existe una técnica que permite la sincronización y coordinación de todos los equipos y evitar los aislamientos. Son las reuniones conocidas como **Scrum of Scrums**. En ellas, se trabajará la integración de unos con otros, así como la detección de posibles dependencias.

Estas reuniones se centran principalmente en prevenir y solucionar los solapes entre los

diferentes equipos.

Para que estas reuniones funcionen correctamente, hay una serie de reglas que conviene aplicar:

- **Un representante** de cada equipo se reunirá con el resto de los representantes de los demás equipos. Esta persona la elegirá el mismo equipo y no tiene por qué ser siempre la misma. Habitualmente suele asistir el *Scrum Master* y podrá ir acompañado de cualquier persona que pueda ofrecer información relevante al resto de equipos o que simplemente quiera conocer el estado del trabajo colectivo.
  Es crucial que los asistentes al *Scrum of Scrums* transmitan puntualmente la información obtenida en esta reunión al resto de sus equipos.

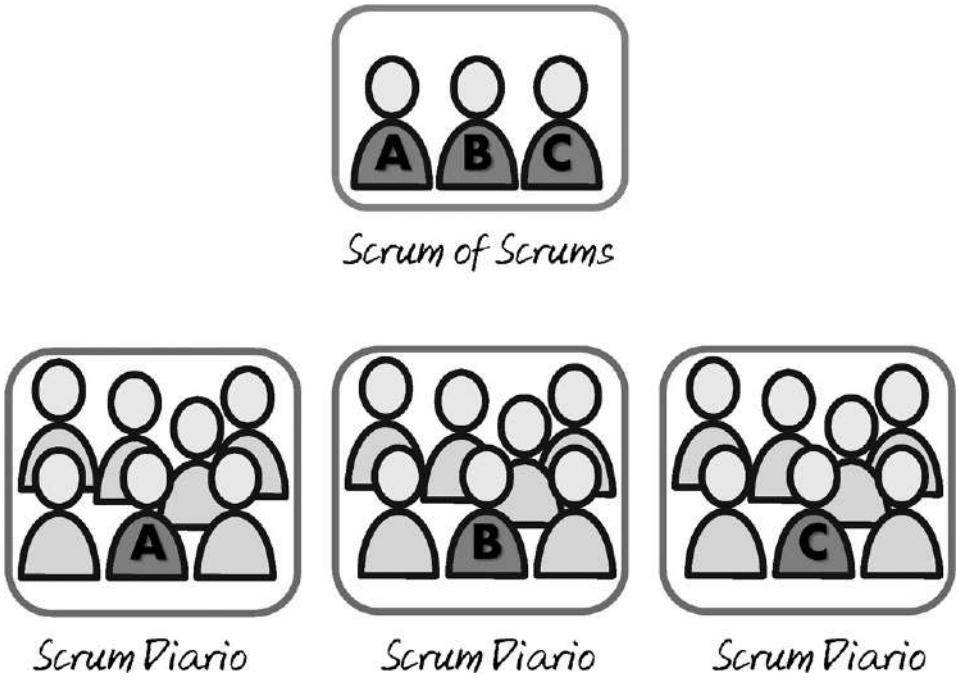


Figura 12.5. Reuniones de Scrum of Scrums.

- **La estructura** no es la misma que la de las reuniones diarias o *Daily meeting* típicas de *Scrum* para un equipo pequeño. En estas reuniones de *Scrum of Scrum*, la sincronización debe realizarse a nivel de equipos y no tanto de requisitos concretos.
  Las preguntas son parecidas a las de la *Daily Meeting*, pero siempre pensando en el equipo completo. Además, se añadirá una última pregunta característica al aplicar *Scrum of Scrums* que será necesaria para la correcta coordinación del trabajo entre los distintos equipos. Una posible fórmula para realizar este tipo de reuniones podría ser la siguiente:
  1. ¿Qué ha hecho su equipo desde la última reunión?
  2. ¿Qué va a hacer su equipo hasta la próxima reunión?
  3. ¿Tiene su equipo algún impedimento?

4. ¿Tiene algo que comentar que nos afecte a todos?

- **La duración** de estas reuniones debe seguir la misma regla que el resto de reuniones de *Scrum*, es decir, ser tan corta como sea posible pero valiosa. La regla de los 15 minutos puede adoptarse para responder a las tres primeras preguntas, pero, para tratar el cuarto punto, no debe acotarse demasiado el tiempo empleado. Durante esta reunión hay que tratar de resolver los temas pendientes entre los equipos o cualquier tema que afecte a todos los representados. También debe revisarse el estado del *Backlog* general y completo del producto.
- **La frecuencia** de estas reuniones dependerá de la necesidad concreta de cada proyecto. Lo habitual es que tengan lugar 2 o 3 veces por semana, pero, como todo, dependerá de las necesidades específicas de cada situación.

En el caso de que existieran numerosos equipos trabajando sobre un mismo producto, podrán escalarse también las reuniones de *Scrum of Scrums* tantas veces como sea necesario. Un representante de cada reunión de *Scrum of Scrums* asistiría a la reunión del siguiente nivel de *Scrum of Scrums*.

Recuerde:

Cada equipo mantendrá su reunión de sincronización diaria y, además, periódicamente, representantes de todos los equipos mantendrán una reunión para la sincronización entre ellos, conocida como **Scrum of Scrums**.

Gestión de dependencias entre equipos

Mantener esta compleja maquinaria funcionando es muy delicado y requiere de precisión a la hora de coordinar a los equipos en su trabajo. Las reuniones de *Scrum of Scrums* pueden no ser suficientes y podemos seguir algunas recomendaciones que ayudarán a coordinar y a gestionar las dependencias entre los equipos que trabajan en un mismo producto. Estas son:

- **Combinar a los miembros de los equipos:** Entre *Sprint* y *Sprint*, es una buena práctica intercambiar a algún miembro de un equipo a otro. Con esto, se favorece la comunicación entre equipos, se transmitirá la forma de trabajar y pensar entre ambos grupos y se eliminarán disonancias entre los equipos.
- **Hacer reuniones de planificación de las entregas o releases:** Reunir a todos los equipos que están construyendo un mismo producto al inicio de una nueva etapa es una práctica extraordinaria para alinear sus objetivos y crear equipo recordando el propósito general del proyecto.
- **Trabajar en la planificación del trabajo futuro:** Tener presente el trabajo de los dos

Sprints siguientes ayuda a detectar, de forma prematura, las posibles dependencias con otros equipos y a buscar la solución de forma temprana.

- **Invitar a las Sprint Reviews:** Hacer *Reviews* abiertas e invitar a miembros de otros equipos es la manera más directa de compartir el estado del producto que está creando un equipo en concreto. En estas reuniones, se detectarán puntos de enlace entre equipos, así como oportunidades para aprender cómo trabajan otros. El equipo de *Product Owners* al completo debe asistir a las *Reviews* de todos los equipos para tener una visión global del producto y poder continuar trabajando con el *Backlog* del producto para gestionar dependencias y prioridades.
- **Realizar Retrospectivas generales:** Al final de una etapa es un buen momento para que todos los equipos realicen un análisis de cómo están trabajando para buscar las mejoras colectivas que podrían aplicarse en la siguiente etapa. La dinámica y objetivos serían los mismos que los de una Retrospectiva habitual de equipo, pero a gran escala. La periodicidad debe decidirse por todos los equipos, pero un momento extraordinario es al final de cada *Release*. Por supuesto, una retrospectiva general puede ser convocada en cualquier momento si se considera necesaria.
- **La sincronización de los Sprints:** Todas las recomendaciones anteriores para la gestión de dependencias entre los equipos son muchísimo más fáciles de aplicar si se trabaja con *Sprints* sincronizados, es decir, tratar de hacer coincidir el inicio y fin de los *Sprints* de todos los equipos que trabajan en el mismo producto.

  No es necesario ser muy estrictos con las fechas, puede manejarse con facilidad un margen de 2 o 3 días de desfase. La sincronización de *Sprints* puede realizarse manejando múltiplos, es decir, los *Sprints* de un equipo logran durar una semana, los de otro equipo dos y los de un tercer equipo cuatro semanas.

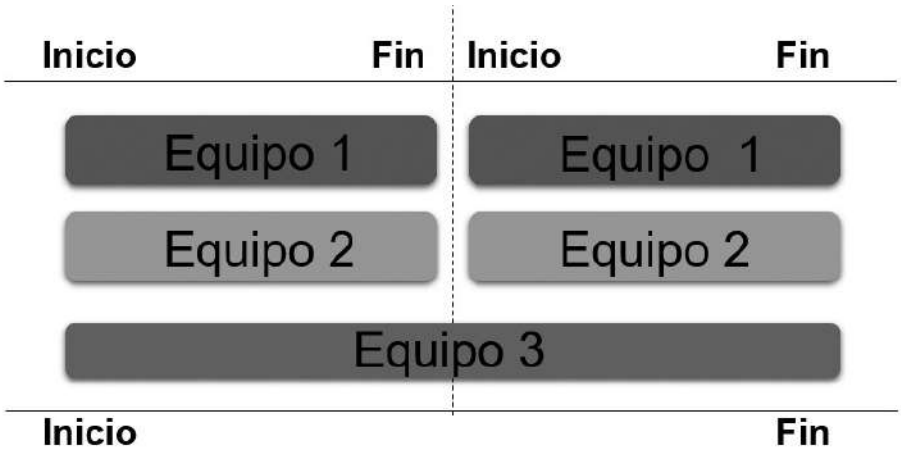


Figura 12.6. Sincronización de los Sprints.

Recuerde:

Si las personas están distribuidas geográficamente, es necesario que se disponga de las herramientas necesarias para que la comunicación sea la mejor posible. Nada como la comunicación cara a cara, pero, si esto no es posible, al menos, se debe procurar que haya ocasionalmente reuniones presenciales con la totalidad del equipo, así como disponer de videoconferencias, herramientas que permitan hablar sin cortes ni interrupciones, etc.

Grupos transversales

Al margen de la comunicación dentro de cada equipo, es habitual que exista comunicación entre las personas con perfiles afines, pertenezcan o no al mismo equipo.

Esta comunicación favorece el aprendizaje continuo, así como el intercambio de experiencias, y es algo enriquecedor tanto para un producto concreto como para la empresa en general.

Es frecuente que grupos diferentes dentro de una misma organización estén abordando un mismo problema de manera diversa. Compartir la forma de enfrentarse a retos comunes está claro que solo tiene consecuencias positivas cuando se aplican a cada proyecto en concreto.

Por otro lado, será imprescindible mantener una constante formación y *coaching* para evitar que los valores de *Scrum* se vayan distorsionando con el tiempo y se adquieran vicios en la manera de trabajar que disminuyan el beneficio obtenido de aplicar estas prácticas. Como se vio en el capítulo dedicado a *Scrum* en la empresa, la implantación de estas técnicas tiene consecuencias que afectan al conjunto de la organización, a su forma de trabajar y de relacionarse, y debe superar barreras de todo tipo. Solo el apoyo decidido y que implique a todos los estamentos de una empresa puede garantizar el éxito de *Scrum*.

Consejo:

La agenda de los grupos transversales no debe ser rígida pero sí fiable para evitar que dejen de reunirse. Inicialmente podrían fijarse reuniones periódicas hasta que este equipo empiece a ser más maduro. Podrán existir tantos grupos transversales como sean necesarios y podrán coordinarse de manera más o menos formal. En ocasiones, estarán liderados por alguna persona en concreto, pero esto es algo que no debe imponerse.
Es necesario crear una cultura en la empresa en la que se esté escalando Scrum, ya que, al trabajar de esta forma numerosos equipos, puede implicar cambios en la organización para adaptarse a los nuevos roles.

Atención:

Para que escalar Scrum funcione, tan importante como mantener una visión global del

producto es poder transmitir un único y claro objetivo a cada equipo para cada Sprint.

---

54 *Succeeding with Agile. Software Development Using Scrum.* Mike Cohn.

13 Kanban, el otorgador de permisos

En este capítulo aprenderá:

- El origen de Kanban.
- Qué es y cómo trabajar con Kanban.
- Algunos trucos para aplicar Kanban con éxito.

Kanban, el otorgador de permisos

Es muy útil tener presente que hay otros métodos ágiles para la organización del trabajo. Además de *Scrum*, uno de los más extendidos es Kanban. Kanban se fundamenta en los principios *Lean* que se tratan en el capítulo 14 y se centra en eliminar constantemente el "desperdicio", todo aquello que no aporta valor.

En el día a día de un equipo de soporte y mantenimiento, en el de una redacción de un periódico o en un departamento de logística, tratar de fijar *Sprints* de duración determinada puede no tener mucho sentido. Cada día, los equipos tienen que trabajar en las urgencias del momento, que muy probablemente aparecerán de forma impredecible. Kanban puede ser la solución que les ayude a trabajar de forma ordenada y productiva.

La primera vez que se aplicó Kanban en un proyecto de ingeniería de software fue de la mano de David J. Anderson[55] en el año 2004 en Microsoft, con un equipo encargado del mantenimiento de software. El resultado fue un éxito absoluto, ya que la productividad del departamento se multiplicó por tres y los plazos de entrega disminuyeron en un 90%.

Kanban es una palabra de origen japonés que significa signo, señal o tarjeta y debe ser entendido como un "otorgador de permisos". Pero ¿otorgar permisos para hacer qué?

La forma de trabajo con Kanban está basada en definir un número máximo de tarjetas admitido para cada estado del proceso. Será este número el que indique si se puede empezar a realizar un trabajo o hay que esperar a que "haya sitio" para una nueva tarjeta en ese estado. Se trata de esperar antes de continuar avanzando en un punto determinado del trabajo para evitar que se acumule el trabajo en otro punto del proceso. Proceder así es una manera muy efectiva para detectar, de forma temprana, atascos, cuellos de botella o impedimentos. De esta manera, se puede pensar en la solución antes de continuar y evitar que los problemas se hagan mayores y, en consecuencia, más difíciles de solucionar.

Kanban surge de la necesidad de entregar a tiempo un producto y de buscar la mejora en los procesos. Se parte de la premisa en la que las personas implicadas en un proceso son las más capacitadas para encontrar soluciones y mejoras eficaces, ya que son precisamente estas personas las que lo conocen en profundidad.

Figura 13.1. Kanban, palabra de origen japonés que significa signo, señal o tarjeta.

Es interesante destacar que, para poder aplicar Kanban en una organización, es necesario que ya existan unos procesos establecidos en la empresa. Kanban simplemente, y no es poca cosa, ayudará a optimizar estos procesos.

Al empezar a aplicar este método, se debe cambiar lo menos posible la forma en la que se trabaja y, poco a poco, ir detectando oportunidades de mejora para aplicarlas progresivamente.

Las características de Kanban hacen posible su aplicación a cualquier tipo de proyecto o situación. Kanban puede aplicarse con éxito a cualquier proyecto que se lleve a cabo con un equipo de personas y que tenga que realizar entregas en un plazo determinado, sin necesidad de que los ciclos sean periódicos.

La propuesta de Kanban es trabajar de forma que el estado del proyecto tenga una transparencia constante para todos los implicados. Esto permite que todo el equipo pueda identificar cuellos de botella y todo aquello que no es valioso. Cualquier persona puede y debe tratar de eliminar, o al menos reducir, todos aquellos impedimentos que entorpecen la productividad del equipo y que rompen el flujo del trabajo.

Kanban trata de eliminar la ambigüedad y la complejidad. Todo debe estar reflejado y ser

tratado de forma simple y clara. La clave está en reflejar solo y exclusivamente lo necesario. Ni más ni menos.

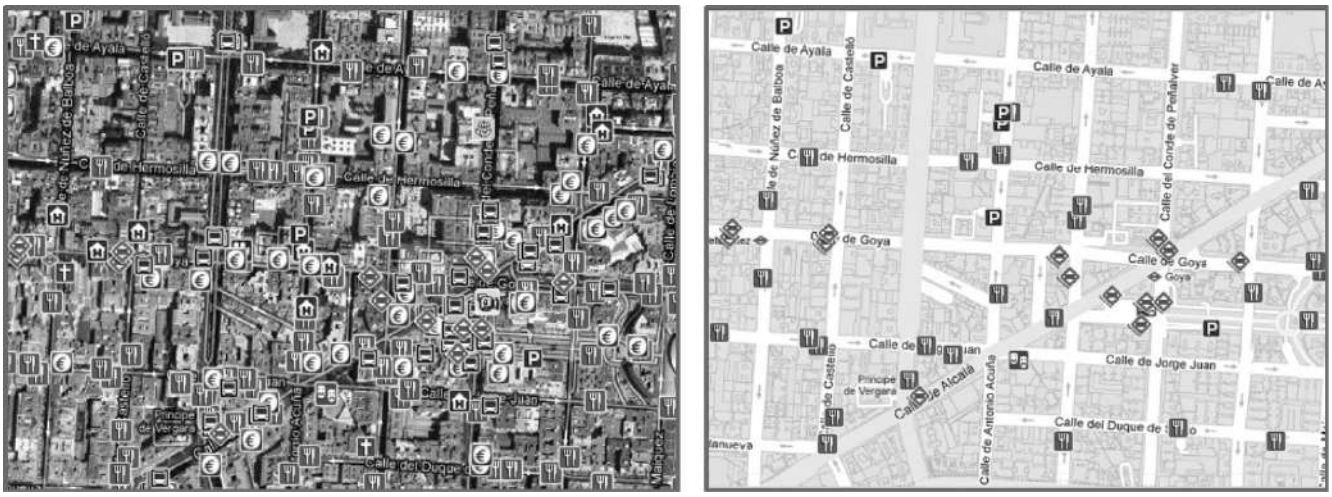


Figura 13.2. Lo complejo frente a lo sencillo.

Cuidado:

Un exceso de información hace que lo importante quede diluido entre datos prescindibles.

Los miembros del equipo deben trabajar en estrecha colaboración y con predisposición para la innovación y la ayuda mutua. Kanban se basa en una confianza plena en el equipo y en su profesionalidad, ocupe el rol que ocupe cada persona. El equipo podrá tomar, en muchos casos, decisiones de actuación en su trabajo diario sin necesidad de que estas decisiones sean supervisadas por un superior.

Pero ¿cómo se trabaja con Kanban?, ¿cómo empiezo? y ¿qué tengo que hacer? Veamos a continuación los pasos que hay que dar para aplicar Kanban a un proyecto.

Kanban en la práctica

Tal y como se ha comentado, Kanban ayuda a mejorar y optimizar los procesos ya establecidos y utilizados en cada organización. Hay muchas formas de empezar a aplicar Kanban a un proyecto, pero, se haga como se haga, hay una serie de etapas que siempre han de realizarse y que se enumeran a continuación:

1. Definir el **mapa del flujo** de su trabajo actual.
2. Crear el "**tablero**" de forma que cada columna represente un estado del flujo definido anteriormente.
3. Definir el **WIP** (*Work In Progress*) admitido para cada columna. Es decir, la cantidad

de tarjetas permitidas en cada estado del flujo.

4. Reflejar en forma de **tarjetas** los ítems que representan cada uno de los trabajos que hay que realizar y colocarlos en el tablero dentro de la columna que le corresponda del flujo.
5. **Medir el tiempo** empleado por cada ítem desde que entra en el tablero hasta que se da por terminado.

En Kanban, las tareas o requisitos que hay que realizar se escriben en tarjetas y se define el número máximo de ellas permitidas en cada etapa del proceso. En función de esta sencilla regla, se tomarán las decisiones correspondientes para actuar como mejor convenga en cada momento.

Veamos, a continuación, todo esto con algo más de detalle.

Definición del mapa de flujo del trabajo

Lo primero que hay que hacer es tener claro cómo se está trabajando, es decir, qué secuencia sigue un requisito o necesidad desde que se detecta hasta que se da por finalizada. Para ello, una acción realmente útil es definir el flujo de trabajo en detalle.

Al hacer esta definición del flujo de trabajo, es importante reflejar los tiempos empleados en completar cada etapa, así como identificar el tiempo de espera empleado por cada ítem entre paso y paso. Por último, este mapa debe completarse con cualquier información necesaria que indique qué políticas seguir para poder dar por finalizada una etapa y poder avanzar a la siguiente.

Los elementos de un flujo están identificados por símbolos definidos de manera estándar, pero su uso correcto es algo que no debe preocuparnos demasiado en este punto. En definitiva, en este primer paso para comenzar a aplicar Kanban, lo fundamental es poder detectar:

- Las diferentes etapas.
- Los tiempos empleados (en cada etapa y entre ellas).
- Los criterios o políticas seguidas para avanzar entre las diferentes fases o etapas.

A modo de ejemplo, en la siguiente figura se muestra un sencillo mapa de flujo de trabajo.

Una vez que se tiene este mapa, se puede empezar a buscar fuentes de “desperdicio” en el proceso, es decir, todo aquello que no añade valor.

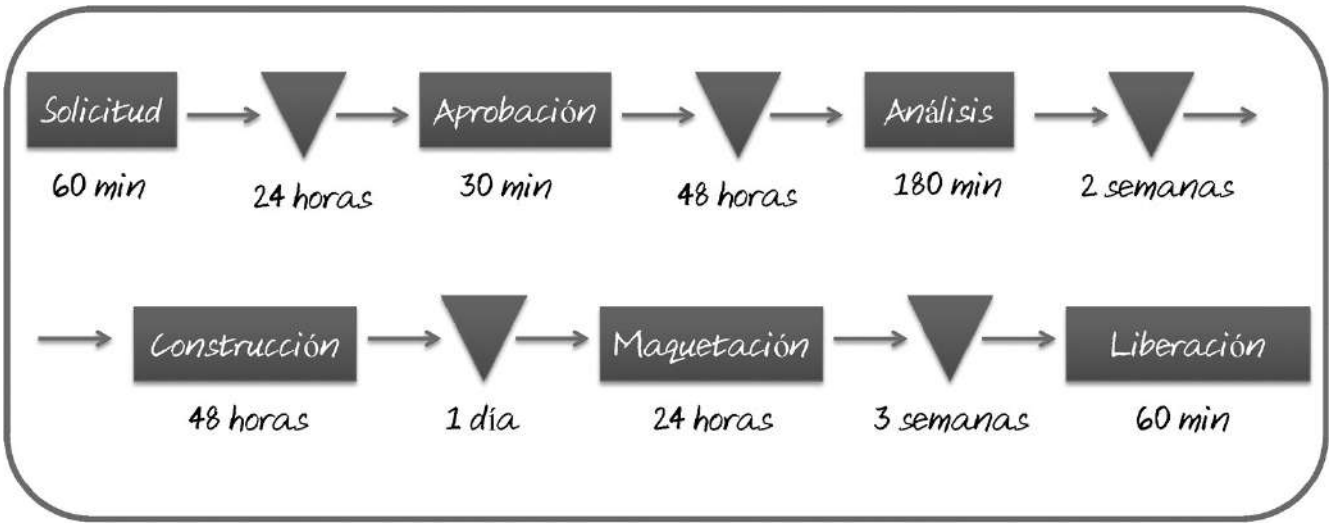


Figura 13.3. Mapa de flujo.

Nota:

Según la filosofía Lean, filosofía en la que se fundamenta Kanban, la principal fuente de desperdicio es la sobreproducción o producción innecesaria en una determinada etapa del flujo, es decir, hacer más de lo que se puede asimilar. Otros focos de desperdicio son los tiempos de espera, los pasos innecesarios o la repetición de tareas.

Creación del tablero de Kanban

La creación de un tablero básico es sencilla. El panel debe disponer de una columna para cada etapa del flujo definido anteriormente. Por ejemplo, para el diagrama representado en la figura anterior, una primera aproximación al tablero Kanban sería la que se muestra en la siguiente imagen:

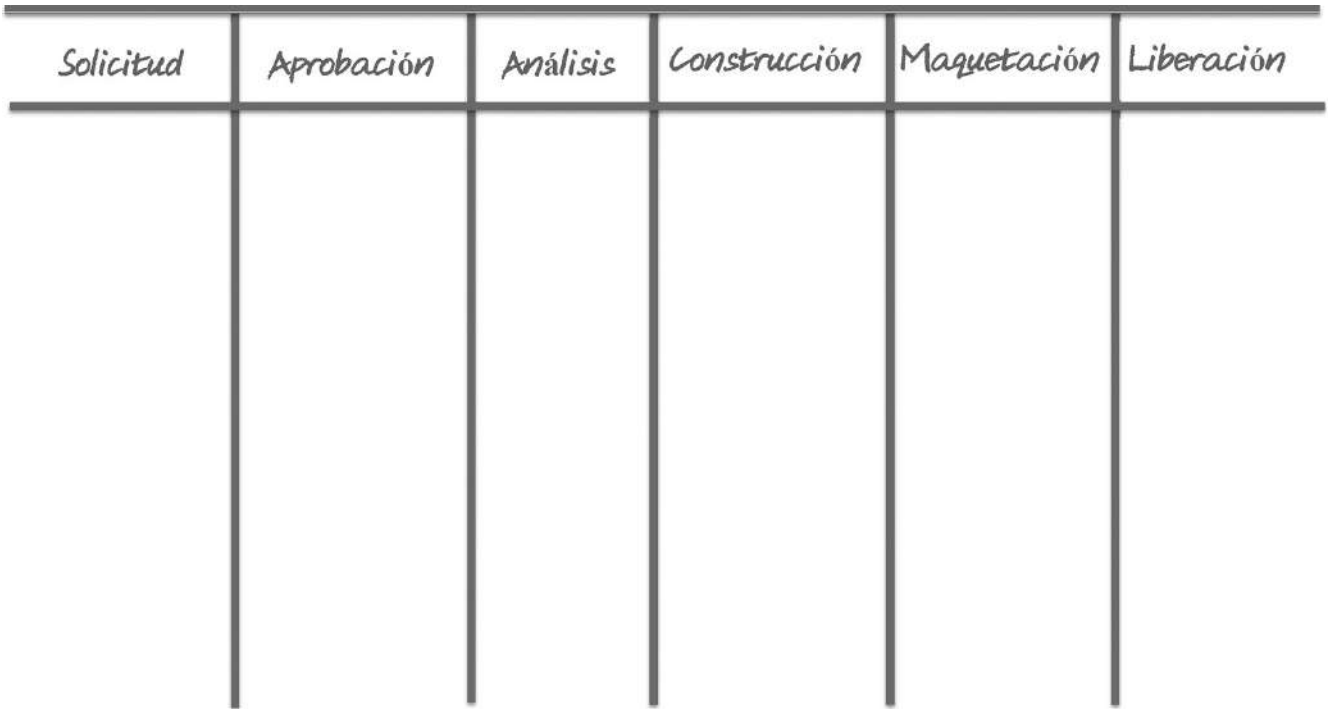


Figura 13.4. Tablero básico.

A este panel le faltan dos cosas fundamentales: el limitador de trabajo por columna, es decir, el WIP, y las tareas. Estos aspectos se tratarán en breve, pero ahora se continuará hablando del tablero en sí.

Para que un tablero de Kanban sea útil, es muy importante decidir en qué momento del flujo empieza y termina la representación del estado del trabajo en nuestro panel. Esto es clave, ya que se está tratando de organizar el trabajo sobre el que tenemos capacidad de acción. Sería un error tratar de "imponer" a personas de fuera de nuestro grupo esta forma de trabajar. Lo que sí se debe articular desde el principio de la adopción de Kanban es la manera de interactuar con los grupos que trabajan en la etapa anterior y posterior a las que nos ocupa como equipo. Hay que buscar la compatibilidad entre la adopción de Kanban en nuestro entorno con el ecosistema que le rodea.

Otro tema muy delicado que hay que tener muy presente cuando cree su tablero es que este debe reflejar el flujo actual de su proceso y no el que le gustaría seguir. Poco a poco, podrá ir adaptándolo y optimizándolo a medida que vaya incorporando mejoras progresivas en su forma de trabajar.

Pero insistimos en que un tablero Kanban debe reflejar el flujo de trabajo real y no el deseado para un proceso.

Importante:

No hay dos tableros de Kanban iguales. Cada equipo debe generar su tablero para que refleje su flujo de trabajo con sus etapas y características específicas. Habrá que ir

adaptando el tablero en función de las necesidades concretas de cada situación.

Definición del WIP

El trabajo en progreso o *Work In Progress* indica el número máximo de tarjetas o ítems permitidos por columna en el panel de Kanban. Este número deberá ir modificándose en función de la demanda, pero hay que tener la precaución de no cambiarlo a la ligera y sin criterio.

Recuerde que las tarjetas de Kanban deben entenderse como un "otorgador de permisos", es decir, no podremos comenzar un trabajo en una etapa si no se ha recibido el permiso para hacerlo. Se sabrá si se tiene este permiso en función del WIP. Si el número de tarjetas en una columna no ha alcanzado su valor máximo, es decir, es inferior al WIP asociado a la misma, sí es posible comenzar a trabajar sobre una nueva tarea en esta columna. Si la columna está "completa", habrá que esperar a obtener dicho permiso, es decir, a que se termine alguna tarea antes de incorporar una nueva a esa etapa.

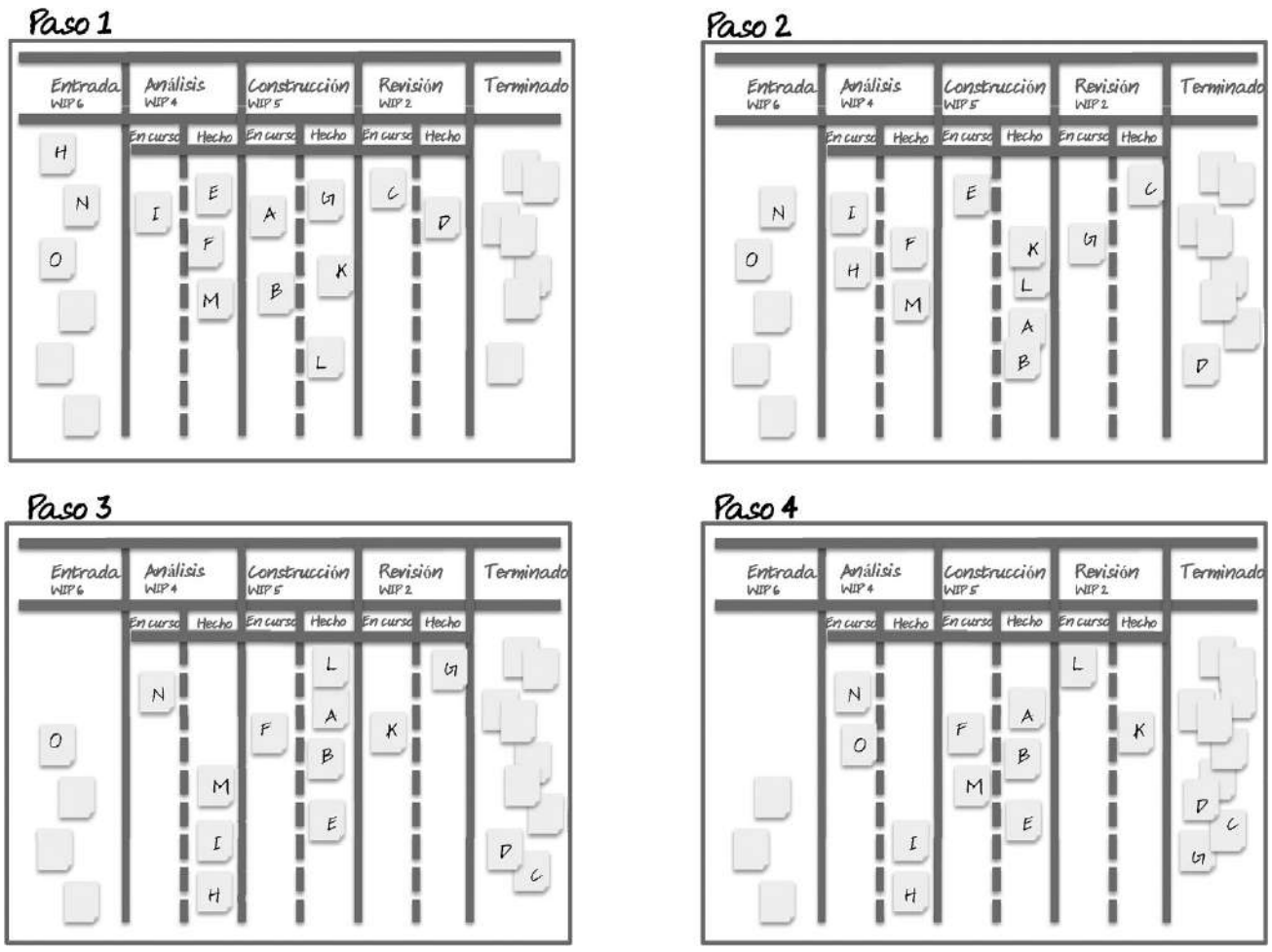


Figura 13.5. Flujo de trabajo.

El asignar un valor al WIP en cada estado o columna no significa que tenga que estar siempre al máximo. Por ejemplo, si el WIP de la etapa de análisis es 5, significa que se podrá estar analizando hasta un máximo de 5 ítems simultáneamente, pero nunca más de 5.

Nota:

El valor para los WIP debe decidirse de manera colectiva.

Una regla sencilla para saber qué WIP asignar a cada columna del tablón Kanban es hacer una relación directa con el número de personas responsables de esa etapa. Si tengo 2 analistas y 5 desarrolladores, para comenzar definiré el WIP del análisis en 2 y el de desarrollo en 5. Esto es una primera regla para empezar, pero luego habrá que revisar estos valores observando el comportamiento del sistema y adaptarlos a medida que avance el proyecto. Pero, tal y como se comentó anteriormente, no se trata de estar cambiándolo a cada momento.

Al utilizar estos límites de trabajo permitido por columna, WIP hace que se detenga el trabajo en un punto dado y que el equipo se centre en “desatascar” los cuellos de botella o tapones que aparezcan en el proceso. El equipo entero tendrá una visión clara de dónde se está produciendo el parón. En cualquier caso, es fundamental mirar más allá del tablero para entender qué es lo que está ocurriendo. Puede que haya un bloqueo muy temporal y que no sea necesario cambiar nada o, por el contrario, puede ser un síntoma de un problema real.

Asignar un límite al trabajo permitido en cada estado aporta un enorme beneficio al producto, ya que evita que se aborden muchas tareas de manera simultánea. El problema de trabajar en muchos temas a la vez es que conviven numerosos temas incompletos durante un tiempo indefinido y esto implica directamente acumular desperdicio y ser fuente de posibles errores. Como dice el conocido refrán: “Quién mucho abarca, poco aprieta”.

Asignar WIP a cada columna es un buen mecanismo para crear una cultura de mejora continua a todos los miembros del equipo.

Atención:

El tablero de Kanban nos da mucha información sobre el proyecto, pero no toda.
Es frecuente que un problema se haga visible en un punto del tablón Kanban, pero que no sea allí donde esté el origen del conflicto. Hay que mirar más allá de lo que abarca el tablero Kanban para tener la información completa.

Incluir los ítems de trabajo

Ya tenemos nuestro tablero, pero ahora queda algo clave y es que hay que poblarlo con el trabajo que hay que realizar.

Los tipos de tareas que un equipo debe realizar pueden ser muy variados y tener orígenes muy distintos.

Un equipo puede tener que incorporar una nueva funcionalidad a un sistema o corregir un defecto. Pueden existir trabajos de mantenimiento, tareas para solucionar un tema bloqueante o simplemente realizar trabajos para la mejora de algo ya existente. En definitiva, una larga lista de trabajos que tiene que llevar a cabo un equipo.

David J. Anderson propone clasificar los distintos trabajos o tareas en clases de servicio y esta clasificación se realiza en función del impacto en el negocio que tiene la realización en tiempo o el retraso de una tarea. Las clases de servicio más generales son las siguientes:

- **Estándar:** Reflejan los requisitos o necesidades del usuario que el producto debe cumplir. Se corresponderían directamente con las historias de usuario más tradicionales.
- **Urgente:** Estas tareas son indeseables, pero también son inevitables. Las tareas urgentes son peligrosísimas porque se incorporan al flujo de trabajo del equipo de forma inesperada y rompen su orden y el ritmo. Hay una práctica muy útil para tratar de reducir el número de urgencias en curso de manera simultánea y funciona de la siguiente manera: se crea una fila adicional en la parte superior del panel a modo de "autopista" y se aumenta en uno (+1) el WIP de todas las columnas del tablero. Esto implica que solo se pueden tratar las tareas urgentes de una en una. Es decir, hasta que no acabe con ella no puedo empezar con otra. Al estar en la parte superior, refleja urgencia y prioridad en su realización. La idea es quitarse las urgencias lo antes posible para que el equipo recupere cuanto antes su ritmo de trabajo. Tratar así a las "urgencias" tiene un gran impacto, ya que es un procedimiento muy llamativo (véase la figura 13.6).

  Esta práctica puede ser adaptada a su situación concreta y tal vez no le quede otro remedio que aumentar el WIP en otro valor que no sea simplemente +1 para tareas urgentes. Lo que sí se debe vigilar es que las tareas urgentes no se conviertan en una forma habitual de trabajo y tratar de limitar su capacidad a +1 tan pronto como sea posible.

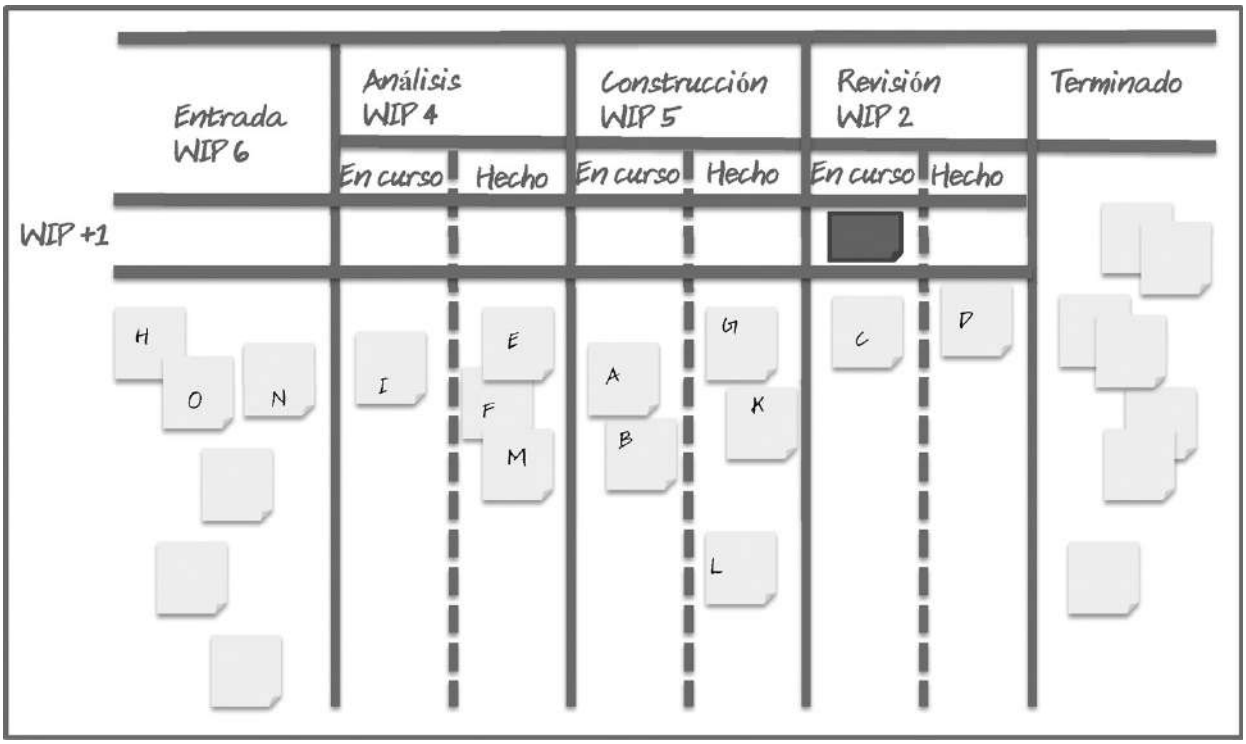


Figura 13.6. Carril para tareas urgentes.

Nota:

Hay que tratar a toda costa que las tareas urgentes aparezcan solo ocasionalmente.

- **Fecha de entrega fija:** Este tipo de tareas corresponden a trabajos que deben finalizarse en una fecha concreta y que el incumplimiento de dicha fecha pueda tener un gran impacto en el negocio.
  En la tarjeta debe aparecer de forma muy visible la fecha de entrega para que se tenga muy presente este importante dato.
  Por ejemplo, si se va a lanzar para la campaña de Navidad un producto al mercado en Portugal y existe una tarea que sea "traducir la guía de usuario al portugués" con fecha de entrega el 30 de noviembre de 2018, se trata realmente de una fecha fija. Esa guía debe estar disponible para poder ser entregada al usuario final junto con el producto. El impacto si esta tarea no se realizara a tiempo sería muy grande.

Nota:

Es necesario asegurarse de que estas tareas tienen una fecha de entrega fija y que esta no es el "capricho" de alguien.

- **No funcional o de infraestructura:** Este tipo de tareas también son conocidas como intangibles. Son trabajos que hay que realizar y que no tienen que ver directamente con

el usuario. Un ejemplo de este tipo de tareas puede ser refactorizar software o hacer un documento interno para el grupo.

Habitualmente, no son tareas urgentes, pero sí es importante realizarlas y el impacto de no completarlas es incluso mayor que el de las tareas estándar.

Anderson representa de forma muy sencilla el impacto que tiene el retraso en la entrega de las diferentes clases de tareas (véase la figura 13.7).

Una práctica tremendamente útil es utilizar un código de colores para identificar de manera rápida y visual la clase a la que pertenece cada tarea. Para identificar las tareas urgentes, una buena elección es aplicar un color muy llamativo, por ejemplo, un rojo intenso.

Ya hemos clasificado las tareas, pero ¿qué información debe contener una tarjeta en un tablero Kanban?

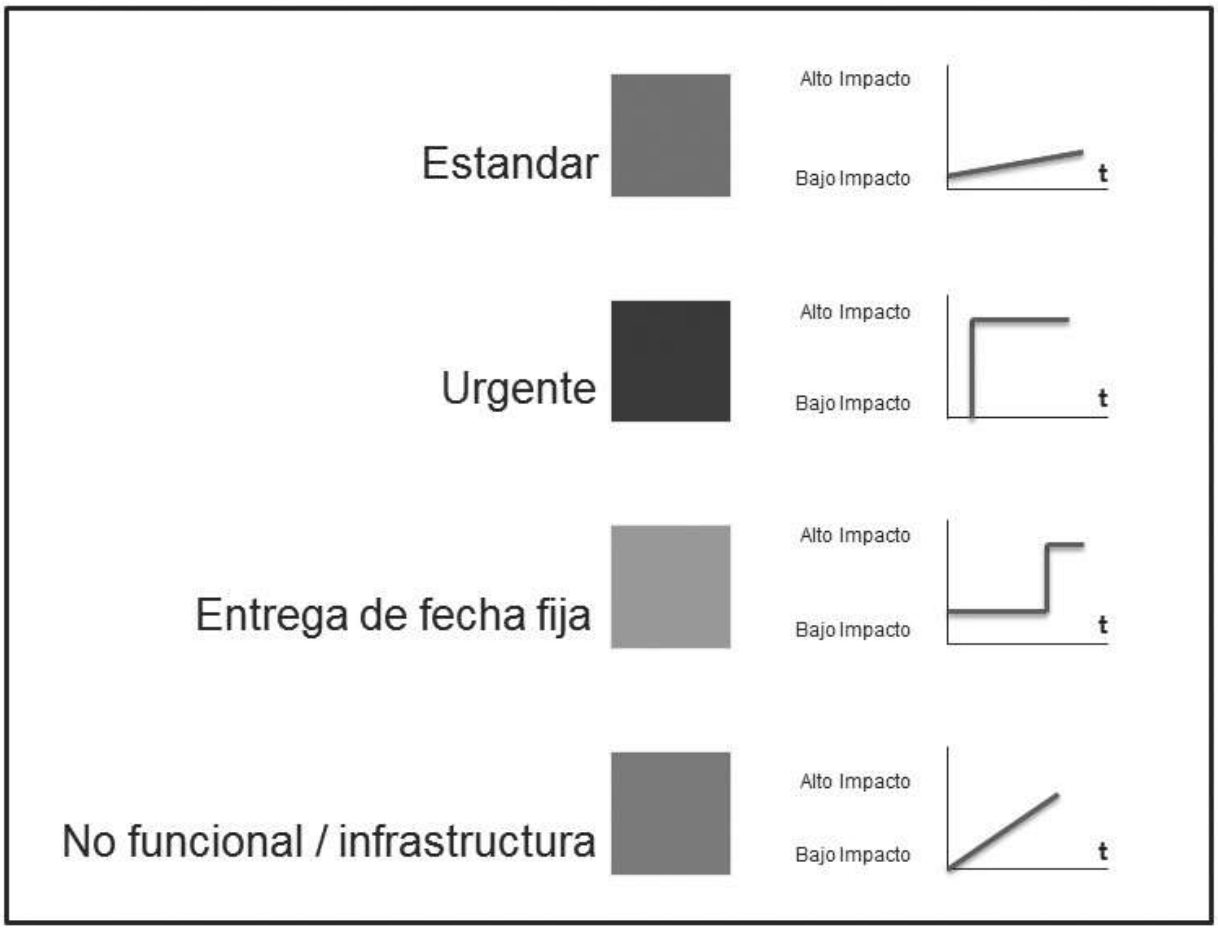


Figura 13.7. Impacto del retraso en la entrega de tareas.

Una tarjeta debe contener toda la información necesaria asociada a ella, pero solo y exclusivamente la necesaria (véase la figura 13.8). De forma muy resumida, toda tarjeta Kanban debe contener:

- Un **identificador** único, de manera que se tenga una trazabilidad de la tarjeta y se

puedan obtener métricas de forma más sencilla.

- Un **título** que identifique de qué se trata el trabajo.
- Una brevísima **descripción** de la tarea, explicando exclusivamente los detalles necesarios para la correcta realización de dicha tarea.
- Las **fechas** de entrada y de salida de la tarjeta en el flujo o tablón Kanban para poder conocer el tiempo completo del ciclo. Puede ser útil también reflejar las fechas de entrada y salida en las diferentes fases. Como todo, dependerá de cada equipo esta decisión.
- Las **personas** que han trabajado en ella principalmente para tener localizadas a las personas que preguntar los temas relacionados con la tarea.
- Cualquier **información** relevante (dependencias, bloqueos, etc.).
- El **origen** de quién solicita un trabajo o por qué surge esta necesidad.

Figura 13.8. Contenido de una tarjeta.

Truco:

Si es posible, trate de dividir los ítems o tareas de forma que tengan un tamaño equivalente para que, de un vistazo, pueda evaluar la cantidad de trabajo en cada estado o columna.

Tiempo del ciclo

Se entiende como tiempo del ciclo el que se emplea en trabajar en una determinada tarea. Es decir, el tiempo real desde que un ítem se incorpora al tablón de Kanban hasta que se da

por finalizado completamente. El objetivo final debe ser tratar de mejorar el proceso, eliminando todo aquello que no aporta valor y, en consecuencia, disminuir el tiempo del ciclo de las tareas.

Algunas ideas para aplicar Kanban con éxito

Tal y como se ha comentado, no hay dos tablones de Kanban iguales, ya que cada equipo debe construirse el suyo adaptado a su situación y proyecto.

Sin embargo, existen una serie de prácticas que tal vez puedan ser valiosas en un proyecto determinado:

- Hay equipos que priorizan los ítems en la columna de la izquierda del panel de manera que el ítem más prioritario estará en la parte superior de la columna y el menos prioritario estará en la parte inferior. De esta forma, en cuanto pueda entrar una tarea en la siguiente columna, lo hará la más prioritaria. A lo largo de todo el flujo, los ítems superiores en cada columna se deberán trabajar en primer lugar.

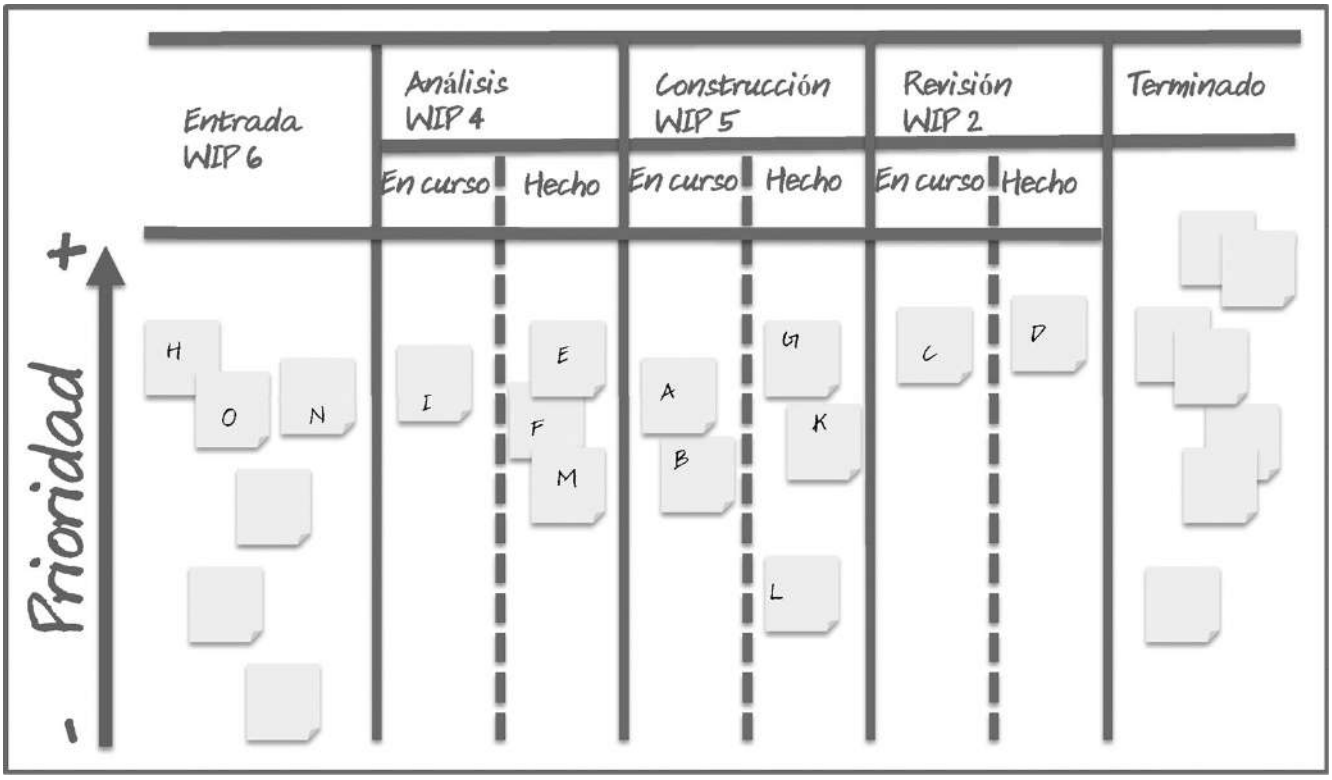


Figura 13.9. Tener claras las prioridades siempre es de gran ayuda para el equipo.

- Un equipo que trabajaba en la actualización y mantenimiento de la página Web de una revista de actualidad detectó que atendía de forma mucho más temprana a las peticiones que realizaba el equipo de publicidad y que las peticiones realizadas por el

equipo de diseño pasaban siempre a un segundo plano. Esto se debía a que las peticiones de publicidad afectaban directamente a ingresos y, sin embargo, las necesidades del equipo de diseño eran mejoras aplazables en el tiempo. Esto llevaba consigo el descontento del equipo de diseño y el peligro de quedarse desfasado en dicho terreno.

La solución puede pasar por definir carriles horizontales, conocidos como carriles de nado por su similitud con las calles de una piscina de competición. Se ha de establecer un número determinado de tareas para realizar en función del origen de la petición y este número debe convivir con el WIP acordado para cada etapa del proceso.

- Todo el equipo debe conocer de forma explícita y clara cuáles son los criterios de actuación para trabajar y mover las tarjetas, así como las políticas que hay que seguir en el día a día del trabajo. Debe estar claro para todos en función de qué criterio se elige una tarjeta u otra para continuar con el trabajo. Cómo saber si se puede dar por finalizada una tarea en una determinada etapa, cuántas tareas urgentes son permitidas simultáneamente en el tablón, cómo se actuará en el caso de que un miembro del equipo no tenga trabajo y otros estén saturados, etc.

Importante:

Es un error confundir Kanban con la improvisación constante e incontrolada. Deben existir políticas de actuación acordadas y conocidas por todo el equipo.

- Eliminar los cuellos de botella debe ser una prioridad para todo el equipo, aunque no afecte inicialmente a todos sus miembros por igual. Si algo es beneficioso para el proyecto completo, también lo es para cada uno de los participantes. Kanban basa su éxito en la transparencia, de forma que los impedimentos o “tapones” serán visibles para todos simultáneamente. La forma de proceder será la siguiente:

  1. Identificar el cuello de botella.
  2. Investigar la restricción.
  3. Subordinar todo lo demás.
  4. Eliminar la restricción.
  5. Volver al paso inicial.

Figura 13.10. Eliminar los embotellamientos beneficia a todos antes o después.

Roles y reuniones en Kanban

Al inicio de este apartado, se comentó que, para poder aplicar Kanban en una organización, es necesario que existan unos procesos ya establecidos. Esto significa que se pueden adaptar los roles que existan en su organización y empezar a trabajar con Kanban.

Si existe un jefe de proyecto, esta persona podrá encargarse tal vez de priorizar requisitos en el panel, de eliminar los impedimentos del equipo, de generar las gráficas de trabajo, etc. Kanban se basa en el principio de “menos es más”, de forma que no incorpore roles nuevos si no van a aportar un valor directo al proceso o van a entorpecer el buen funcionamiento y coordinación del equipo.

Nota:

Kanban no prescribe roles. Simplemente recomienda que se incorporen los roles que vayan siendo necesarios para el correcto funcionamiento del equipo. En caso de que ya existan, se debe tratar de optimizarlos.

Respecto a las reuniones, Kanban no establece nada formal. En cualquier caso, sí recomienda que diariamente el equipo se reúna para analizar el trabajo y detectar tanto problemas como mejoras en el proceso. Estas reuniones diarias están enfocadas a comentar el estado del panel de Kanban y las tareas en curso. No se personaliza como en las *Daily Meeting* de *Scrum*, ya que no se habla tanto del trabajo de las personas, sino del estado de las

tarjetas en el tablero.

Las reuniones de planificación del trabajo en Kanban deben realizarse, pero estas tendrán lugar “fuera” del trabajo reflejado en el tablero. La frecuencia de estas reuniones será marcada por las necesidades de cada proyecto.

En resumen

Kanban es un método útil y efectivo tanto para el desarrollo de productos de software como para la organización de un departamento de recursos humanos o de un equipo de soporte, entre otras muchas aplicaciones.

Recuerde que la forma de comenzar a aplicarlo es adaptando el método a los procesos, roles y rutinas que tenga su empresa y, poco a poco, en función de las necesidades que se vayan detectando, ir incorporando las mejoras de manera progresiva. Debido a la transparencia que supone trabajar con Kanban, las deficiencias de un sistema saldrán a la luz y será posible corregirlas de manera temprana.

Aplicar Kanban es una buena elección en aquellos proyectos que no pueden funcionar con iteraciones de duración fija, en proyectos con un alto grado de inestabilidad en la aparición de los requisitos o en cualquier proyecto de gestión o mantenimiento.

En definitiva, ¡pruebe, experimente y elija para su proyecto el método que más valor le aporte!

---

55 Kanban: Successful Evolutionary Change for Your Technology Business. David J. Anderson. Ed. Blue Hole Press. 2010.

14 Lean Software Development

En este capítulo aprenderá a:

- Comprender la visión ágil que propone *Lean*.
- Conocer los principios y herramientas de *Lean Software Development*.
- Ver un ejemplo de *Lean* en funcionamiento.

De la fabricación a la programación

Lean Software Development tiene su origen en la filosofía de fabricación *Lean* que tiene sus raíces, como otros muchos elementos de los métodos ágiles, en la compañía Toyota. Se considera que el punto de partida de estas nuevas formas de trabajo fue la visita de una serie de expertos norteamericanos en los años 50 para ayudar en la reconstrucción de la industria japonesa. Uno de esos expertos era William Edwards Deming, que introdujo conceptos relacionados con la calidad que fueron aplicados con entusiasmo en Japón. De la combinación de la necesidad de recuperar la industria con pocos recursos e infraestructuras muy dañadas y dar una gran relevancia a la calidad del proceso y el producto, surge esa nueva aproximación, *Lean Manufacturing*, que se resume muy bien en tres puntos:

- Fabricar solo lo necesario.
- Eliminar el desperdicio, lo que no añade valor.
- Cero defectos.

Otra cara del *Lean Manufacturing* es la mejora continua o Kaizen, un término de origen japonés, al igual que otros aplicados en el mundo de los métodos ágiles, como Kanban. La mejora continua es uno de los elementos definidos en el manifiesto *Ágil* y no es otra cosa que la acción proactiva para experimentar e identificar nuevas mejoras sin miedo a equivocarse, sin culpables. Más que un proceso o una serie de técnicas, es una mentalidad, ya que debe calar en las personas y hacer que cambien su actitud habitual.

Lean Manufacturing es un conjunto de técnicas muy probado y experimentado, que ha beneficiado ampliamente a la fabricación industrial. Dado que define más una filosofía de trabajo que un conjunto de técnicas, herramientas o procesos, es posible adaptarlo a otras esferas de la actividad humana.

En este capítulo, veremos cómo se ha traducido al mundo del software dando lugar a una de las filosofías (más que método) más aplicada y cómo ha sido puesto en práctica por algunas organizaciones.

Lean aplicado al software

Lean Software Development como método se fundamenta en un libro, posiblemente uno de los más citados en el mundo de los métodos agiles: *Lean Software Development. An Agile*

Toolkit[56], de los hermanos Poppendieck. En él, se habla de una serie de principios, técnicas, componentes, procesos y aspectos metodológicos que definen una traslación al mundo del software de los principios de los métodos *Lean* procedentes del mundo de la fabricación industrial.

Lean Software Development tiene tres objetivos principales: reducir drásticamente el tiempo de entrega de un producto, reducir su precio y reducir también el número de defectos o *bugs*, es decir, mejorar su calidad.

El libro de los Poppendieck define siete principios y 22 herramientas que ayudan a cumplirlos. En realidad, todos ellos están presentes de una forma u otra en el presente libro, lo que ayuda a entender lo influyente que es el original *Lean Software Development*. Estos principios han sido reformulados y adaptados en más de una ocasión. En el capítulo 1 de este libro se enumeró una lista y aquí se expondrá en detalle la original de 2003, para dar una perspectiva diferente y más amplia.

Eliminar desperdicio

El desperdicio se define como todo aquello que no aporta un beneficio al cliente, que no es valioso para él. En un sentido amplio, esto implica también todo aquello que dificulta o entorpece el proceso para llegar al producto. Seguramente, es uno de los principios de más alcance e impacto para el conjunto de los métodos ágiles.

Los ejemplos de desperdicio en el mundo de la fabricación son la sobreproducción, el exceso de inventario, el transporte innecesario, los defectos, las esperas... La primera herramienta de *Lean Software Development* define las pautas para ayudar a identificar el desperdicio, que, además, son una traducción del desperdicio de la fabricación al mundo del software:

- **Trabajo a medias:** Un trabajo no completado no aporta valor y, al mismo tiempo, obliga a desviar atención hacia él. No solo eso; si no se actualiza y completa, pierde su capacidad de aportar valor a medida que el producto evoluciona. Es doblemente pernicioso. Sería el equivalente del inventario en la fabricación.
- **Funcionalidades o código innecesario:** Desvían recursos para completar algo que, en realidad, no se demanda. A veces, hay funcionalidades muy complejas que apenas son utilizadas. Equivaldrían a la sobreproducción.
- **Burocracia o documentación excesiva:** El proceso no debe utilizar más recursos de los necesarios para construir el producto. Un ejemplo es la documentación, que debe ser concisa y útil, no exhaustiva y amplia sin necesidad.
- **Defectos y funcionalidades erróneas:** El esfuerzo invertido en construir algo con defectos o que no los tenga pero que no haga lo que debe implica que habrá que añadir más esfuerzo para corregir esa situación. Es uno de los desperdicios más evidentes y

con mayor impacto. Se soluciona trabajando con calidad, desde el primer momento tanto en el producto como en el proceso.

- **Las esperas:** Por ejemplo, para solventar una duda sobre una funcionalidad demandada. Esperar para aclarar un aspecto sobre el producto en construcción supone no poder completar el trabajo a tiempo o hacerlo defectuosamente por no demorarlo más. Por ello, se insiste tanto en los métodos ágiles en el *feedback* temprano y continuado.
- **La pérdida de foco:** Esto ocurre cuando se está cambiando continuamente de tarea. El software es una actividad técnica que requiere toda la atención de las personas que participan en su construcción, es decir, necesita máximo foco en su trabajo. La tendencia hacia la "multitarea" cambiando continuamente el punto de atención de las personas solo da lugar a demoras o a pérdida de calidad (y frecuentemente a las dos cosas).

Una vez identificado el desperdicio, la segunda herramienta que se expondrá ayuda a reducirlo. Consiste en aplicar una serie de acciones como:

- Reducir las actividades de gestión innecesarias.
- Evitar ciclos de aprobación (y con ellos esperas) innecesarios.
- Aplicar una visión crítica e inconformista para ayudar a identificar el desperdicio.
- Analizar el flujo de trabajo y, sobre todo, de aporte de valor, para detectar y eliminar ineficiencias.

Para este último aspecto, el análisis para detectar y eliminar ineficiencias encaja perfectamente el uso de retrospectivas. Tal y como se vio en el capítulo 8, las retrospectivas son un proceso colaborativo, iterativo y continuado de mejora continua.

En definitiva, se trata de la traducción al mundo del software del principio Kaizen de mejora continua. Las retrospectivas tienen la ventaja de la participación del equipo a la hora de identificar los problemas y los puntos de acción. Esta participación del equipo ayuda también a que hagan suyas las acciones correctivas, reduciendo las resistencias habituales en gente acostumbrada a aplicar un determinado modo de trabajo (aunque no estén satisfechos con él).

Nota:

Es habitual utilizar prácticas o procesos en el trabajo que por costumbre o tradición siempre se han seguido, sin pararse a analizar si realmente es necesario seguir trabajando de esta forma. Por ello, todos los implicados en la construcción de un producto deben buscar los factores y procesos que reduzcan el valor del mismo para analizar su origen y eliminarlo. Si no pudiera eliminarse el impedimento, al menos se debe tratar de reducir su impacto.

Amplificar el aprendizaje

La construcción de software, como todo proceso creativo de alta incertidumbre, no busca producir resultados repetibles, sino crear soluciones para problemas únicos. Por ello, son útiles los mecanismos de descubrimiento como los ciclos cortos y repetidos de investigación, experimentación y validación. En estos ciclos, la creación de nuevo conocimiento es crítica, tanto como el aprendizaje progresivo a medida que se profundiza en el problema.

Las herramientas propuestas ya están en parte definidas en la propia naturaleza del software:

- La primera es el feedback, realimentación, comentario o reacción (realmente no hay una buena traducción para este término). Esto supone una comunicación muy fluida y frecuente, especialmente entre quien plantea el problema (cliente, usuario, fuente de requisitos) y quien da una solución en forma de software. La calidad, entendida como prueba del código, es otra fuente muy importante de *feedback*, al tiempo que ayuda a reducir el desperdicio. De la misma forma, priorizar la recepción de *feedback* frente a actividades formales ayuda a reducir desperdicio y ganar foco.
- La segunda son las **iteraciones**, que ya se han mencionado como un elemento decisivo a la hora de reducir la incertidumbre. Como se ha visto en los métodos ágiles descritos en este libro, las actividades se ordenan casi siempre en iteraciones cíclicas y, a ser posible, cortas. Es una forma de recibir *feedback* temprano.
- La **sincronización** ayuda a reducir el desalineamiento por falta de información y va de la mano del trabajo en iteraciones, ya que favorece la sincronización de todos los actores en el proceso. Sincronizarse requiere una comunicación fluida y continuada y un esfuerzo adicional de gestión, pero a cambio reduce desperdicio y aumenta la productividad, por ejemplo, al paralelizar actividades.

Decidir tan tarde como sea posible

Es una respuesta a la incertidumbre: se trata de llevar el momento de tomar una decisión al más lejano posible para contar con la mayor cantidad de información. Claro está, siempre y cuando ese retraso en la decisión no afecte a otros factores como la planificación o a otras tareas dependientes.

Nota:

En Lean se utiliza con frecuencia la expresión "último momento responsable", que lo que

pretende transmitir es que se debe aprender e investigar todo lo posible antes de tomar una decisión que afecte al producto. Pero esa decisión debe tomarse en el momento justo, no más tarde. En definitiva, huir del exceso de análisis que pueda paralizar al equipo (el llamado "análisis-parálisis"), pero también de la toma de decisiones de forma precipitada.

Otra forma de ver este principio es como una forma de "mantener las opciones abiertas" mientras sea posible. En áreas de gran incertidumbre y cambio, esto es especialmente importante.

Otra consecuencia es añadir flexibilidad a las decisiones e incorporar por sistema un análisis de opciones y alternativas, valorando impacto y consecuencias.

Una forma de demorar la toma de decisiones es el *Sprint Planning* de *Scrum* (véase el capítulo 5), en el que la lista de las funcionalidades que se van a incorporar a un producto no se define en el momento de iniciar el proyecto. En su lugar se realizan en cada iteración, tomando los resultados y la experiencia ganada hasta ese momento.

Entregar tan rápido como sea posible

Complementa a la anterior: si se pueden hacer cambios y entregarlos en un breve espacio de tiempo, también se pueden demorar las decisiones hasta ese "último momento responsable" sin que la calidad se vea afectada.

La entrega rápida de software es también una forma de recibir *feedback* temprano, al tiempo que se reduce el riesgo de que los clientes duden e introduzcan cambios que tengan mucho impacto sobre el trabajo ya realizado. Completar el trabajo cuanto antes también permite la detección temprana de defectos y facilita su resolución cuando el impacto y coste son menores.

Y hay una razón de negocio obvia: ser los primeros añade una ventaja competitiva accediendo antes al mercado. De esa forma, se puede llegar con funcionalidades básicas, que son insuficientes cuando otros competidores están ya en el mercado y hay que comparar el producto con otros ya existentes en el mercado.

Con frecuencia, las personas quedan bloqueadas: falta de información, incertidumbre o, simplemente, la mentalidad de "no es mi trabajo". También es habitual que la entrega se demore por la acumulación innecesaria de elementos formales y técnicos que van más allá de los requisitos de negocio: la llamada "sobreingeniería".

Todo esto y algunos factores más impiden la reacción necesaria para asegurar una entrega rápida. Por ello, las herramientas para asegurar ese principio inciden en detectar y eliminar estos bloqueos.

Por ejemplo, una herramienta de ayuda es enfocar el trabajo como un "*Pull system*", es

decir, en el flujo de trabajo, la siguiente etapa no espera a recibir, sino que “tira” de las actividades de la anterior. Es el caso de Kanban (véase el capítulo 13): si se tiene una etapa de despliegue, el equipo correspondiente no espera pacientemente a recibir el software con todas las formalidades cumplidas, sino que activamente “tira” de él para llevarlo a su etapa y hacerlo progresar por la cadena. Es decir, se trata de mover activamente el trabajo, pasando la presión a la etapa anterior y no al revés. Esta mentalidad proactiva hace que, ante bloqueos, sean las personas quienes los superen, poniendo en primer lugar el producto y dejando en segundo plano los aspectos formales.

La Teoría de colas es otra herramienta que ayuda a identificar ineficiencias y cuellos de botella y facilita el acelerar el proceso. Una vez más, Kanban es un buen ejemplo de trabajo *Lean*: fija la capacidad de trabajo en cada etapa evitando la saturación, lo que supone además una forma de mostrar las ineficiencias para corregirlas (sobrecapacidad, por ejemplo, construyendo el software, junto a infracapacidad a la hora de testearlo); o crea carriles de aceleración para gestionar las actividades que no puedan esperar, pero haciendo que cuenten como parte del WIP (el *Work In Progress* de Kanban).

Fragmentar el trabajo de forma que se divida en tareas más pequeñas y manejables ayuda a acelerar la entrega. La razón es que los trabajos grandes requieren mayor esfuerzo para completarse, mucho mayor que la suma de las divisiones si podemos gestionarlas de forma independiente. Una actividad pequeña requiere implicar a menos personas, tiene procesos formales más ligeros y añade funcionalidad más limitada y controlada, más fácil de probar, y que puede recibir un *feedback* más preciso. Por el contrario, una actividad grande va a requerir sincronizar a más personas para llevarla a cabo, procesos más pesados, mayor complejidad a la hora de la definición y generación de *feedback*, mayor complejidad y, con esta complejidad, mayor riesgo para la calidad. Finalmente, una actividad más grande va a estar más tiempo en progreso, sin poder ser completada, y esto es considerado como una fuente de desperdicio.

Piénsese en la diferencia entre rediseñar completamente una aplicación, por ejemplo, de finanzas o en dividir ese rediseño en actividades más pequeñas independientes. Si se va a cambiar, por ejemplo, la forma en la que se reciben notificaciones como una actividad separada, se podrá progresar antes al mercado por ser más pequeña. Al mismo tiempo, la calidad mejorará, ya que es más fácil acotar las pruebas de la nueva funcionalidad; el feedback será más preciso, pues solo se referirá a esos cambios y llegará antes al mercado.

Esa misma funcionalidad de notificaciones, embebida en una serie de cambios mayor, tendrá más interdependencias, con más complejidad y riesgo en la calidad por ello, llegará más tarde al mercado (lo hará junto con el resto de la nueva aplicación) y el feedback será más general porque se referirá a otros aspectos.

Otra herramienta para gestionar la entrega rápida es medir el coste del retraso. Se pueden dar muchas razones para recomendar el llevar cuanto antes los productos al mercado, pero tener delante un número que permita cuantificarlo ayudará, como poco, a valorar la necesidad de progresar con rapidez. Esto es, claro está, un riesgo: las métricas deben analizarse

cuidadosamente antes de introducirse.

No solo porque puede ser difícil que reflejen la realidad, sino porque pueden tener consecuencias negativas no deseadas. La medición de un proceso es un aspecto crítico que no se aborda muchas veces con la prudencia necesaria. Es una simplificación de la realidad y, por ello, puede transmitir mensajes simples (en algunos casos equivocados) y condicionar el modo de trabajo: los equipos conscientes de una métrica pueden acabar trabajando para ella, no para el objetivo que pretendía medir. Si, por ejemplo, se mide el tiempo medio empleado en completar una funcionalidad de manera aislada, es posible que los equipos la progresen ahorrando, por ejemplo, en la calidad. Si solo se mide el número y severidad de defectos, se puede provocar un fuerte enfrentamiento entre las personas que desarrollan el software, que querrán bajar esa severidad, y los que lo prueban, que tratarán de subirla. Por eso, hay que aplicar cordura a la hora de definir una métrica y, sobre todo, ser capaces de imaginar sus consecuencias, siendo muy rápidos en cambiar lo que se mide si esas consecuencias empiezan a ser negativas.

Todo esto se puede conseguir con una serie de acciones simples, pero no necesariamente fáciles:

- **Tener las personas adecuadas:** No solo con los conocimientos y experiencia adecuados, sino con la mentalidad y actitud constructiva necesaria: proactividad, flexibilidad, autonomía, capacidad para pensar por sí mismos y solucionar problemas.
- **Simplicidad:** Reducir la complejidad innecesaria aligera todos los procesos. Es la base para poder abordar trabajos más pequeños y atómicos haciendo progresar rápidamente por la cadena estos trabajos.
- **Trabajar como un equipo:** Incluso en distintas fases del proceso. Crear tribus, las fronteras, el “ellos y nosotros” es una forma de ineficiencia y desperdicio.
- **Eliminar desperdicio:** Que ya se ha visto como uno de los principios directores de *Lean*.
- **Calidad:** Debería ser la prioridad número uno que no debería sacrificarse bajo ninguna otra consideración. Si la velocidad se hace a costa de la calidad, se está generando desperdicio: un producto defectuoso cuyos problemas saldrán tarde y serán más costosos de solucionar.

Dar responsabilidad al equipo

El papel del equipo de trabajo es determinante. Uno de los cambios del *Lean Manufacturing* es dejar atrás la visión de cadena de montaje, donde lo que se pide a las personas es realizar una funciones estandarizadas y fácilmente automatizables. Los cambios introducidos en Toyota supusieron dar más capacidad a las personas para, por ejemplo, identificar puntos de mejora y aplicarlos.

La construcción de software es un trabajo de naturaleza intelectual y con una alta incertidumbre: no es posible trabajar como en una cadena de montaje. Eso obliga a tratar de forma distinta a los equipos y proporcionar el marco de trabajo más adecuado para sacar lo mejor de cada persona. Desde la perspectiva *Lean* se ha demostrado muy eficaz el dar responsabilidad (*empowerment*) al equipo.

Tener equipos responsables es un signo de una organización madura. Una organización madura es aquella en la que todo el mundo es consciente del objetivo principal y no antepone intereses menores y parciales. En una organización de este tipo, se pone el foco en el aprendizaje y la autonomía para que las personas que hacen el trabajo tomen sus propias decisiones.

Es por eso que una de las herramientas que se propone en *Lean Software Development* es la autodeterminación. Con frecuencia, se trata de trasladar prácticas que funcionan en un entorno a otro distinto y eso es un error porque cada entorno, equipo y circunstancias requiere una aproximación distinta. En la transformación de las prácticas y procesos, una forma muy eficaz de involucrar a las personas del equipo para que hagan suyo el cambio es hacer que participen en su definición. Incluso las mismas medidas impuestas son peor recibidas si no han sido definidas por los miembros del equipo. De esa forma, el papel de los gestores es más de consejeros o *coach* que de aplicar el "ordeno y mando".

Al final, esta autodeterminación es una forma de motivación, otra de las herramientas de *Lean* para el equipo. Las personas motivadas se involucran más en el trabajo, siendo más productivas y creativas. Hay muchas formas de motivar a las personas, pero, en relación a un proyecto, tener la sensación de participar e influir en el trabajo es una de ellas. Conocer el alcance final del trabajo, tener objetivos claros, información, acceso al cliente para *feedback* y la sensación de influir en el trabajo, así como en su resultado, son buenas formas de promover esa motivación. A algunas organizaciones les resulta difícil alcanzar ese grado de delegación y confianza, pero los beneficios cuando se alcanza compensan a la incertidumbre y la falsa sensación de perder control.

El que un equipo sea capaz de tomar sus propias decisiones no quiere decir que la organización sea anárquica y descabezada. Es difícil alcanzar los objetivos sin contar con liderazgo. Pero hay que entender que un líder no es una persona que ordena, es más una persona que inspira, un referente, alguien que consigue con influencia y no solo por autoridad jerárquica llevar al equipo a donde se quiere.

Hay que entender también que no todas las personas se sienten cómodas tomando todas las decisiones. De hecho, suele haber un rechazo interno a la autodeterminación en los equipos. Para muchas personas, ejecutar órdenes, incluso aunque no gusten, es una actitud cómoda, que no requiere un esfuerzo mental. Además, permite hacer que la responsabilidad recaiga en otros. Tener que tomar decisiones se ve como un inconveniente al hacerse responsable de ellas. El papel del líder ayuda también a reducir esa resistencia, ya que sirve de guía al equipo y puede encarnar esa figura de referente orgánico en el que delegar que algunas personas demandan. Los métodos ágiles ponen a los equipos en una situación

incómoda, pero, a cambio, mejora su trabajo, su entorno y aumenta su grado de satisfacción.

Otra forma de dar capacidad al equipo es por medio del conocimiento, especialmente del conocimiento experto. Cuando alguien siente que controla el dominio de conocimiento en el que se desarrolla su actividad, gana también en seguridad para tomar sus propias decisiones. Por ello, hacer que las personas ganen ese conocimiento experto es otra de las herramientas de *Lean*. Hay varios medios para alcanzarlo, el más evidente es la formación, pero mejor aún es la autoformación, es decir, aquella impulsada y dirigida por la propia persona en función de sus intereses y circunstancias. Las comunidades de práctica ayudan a alcanzar ese conocimiento, a identificar a las personas que lo pueden proporcionar y a crear un espíritu transversal de equipo. El tener mentores para que guíen a las personas que se están introduciendo en un tema o quieren profundizar en él ha demostrado ser un mecanismo muy efectivo.

Para mejorar a las personas y a los equipos teniendo impacto en sus resultados, hay que enfrentarse a muchas situaciones que influyen negativamente. Hay tres que destacan especialmente:

- El individualismo, que afecta a la colaboración.
- El exceso de rotación, que impide crear la confianza necesaria para un trabajo colaborativo efectivo.
- La falta de sensación de permanencia a un equipo, aunque hay que modularla cuidadosamente para que no termine en la “tribalización”, mencionada anteriormente.

Construir con integridad

La integridad afecta tanto al comportamiento del producto como a su estructura interna. Los requisitos que la definen deben materializarse de forma coherente y cohesionada. Cuando el cliente revise el producto, debe percibir que sus requisitos han sido incluidos de una manera armónica y no sentir que está ante una colección de funcionalidades sin mucha relación entre sí.

Normalmente, se habla de dos tipos de integridad:

- **Integridad percibida:** Se refiere a la forma en la que un usuario ve el sistema desarrollado. Es decir, ¿es intuitivo?, ¿resuelve el problema para el que fue construido?, ¿refleja adecuadamente el concepto en el que se basa? y ¿cómo encaja en el mercado al que se dirige?

  Precisamente, para mejorar esa integridad percibida, *Lean* cuenta con una herramienta o, más bien, una serie de técnicas como hacer que un mismo equipo se haga cargo del desarrollo de un sistema o funcionalidad, siempre y cuando el tamaño lo permita, claro, y también hacer iteraciones cortas para recibir cuanto antes *feedback* de clientes y

usuarios ajustando el software para cumplir sus expectativas. Una forma de adquirir ese *feedback* es por medio de pruebas con los usuarios, planteadas para examinar su interacción ante el software que se ha construido y se quiere validar.

- **Integridad conceptual:** Si la anterior se refiere al *qué*, esta examina el *cómo*, es decir, de qué forma ha sido construido el software. También hay una serie de acciones que permiten mejorarla y que constituyen otra de las herramientas de *Lean*. Una, por ejemplo, sería evitar, o al menos reducir, la complejidad del sistema (lo que es una forma de reducir esfuerzo y riesgo, y con ello mejorar la calidad). Otra forma de mejorar la integridad sería aplicar la técnica de resolución integral de problemas: mantener un flujo continuo y bidireccional de información entre quien define el producto y quien lo implementa; dar preferencia a la interacción cara a cara frente a la escrita; o simultanear entendimiento del problema y su resolución.

Una forma de aumentar la integridad del sistema es la técnica de la refactorización del software, otra de las herramientas aportadas por *Lean*. La refactorización es una acción de mejora del software que implica revisar y reescribir el código periódicamente para aplicar mejoras y para evitar que un software que ha ido evolucionando se convierta en una colección de parches aportados por varias personas con visiones y sensibilidades distintas.

La refactorización debe incidir en aspectos que, además de mantener la integridad, mejoren la calidad del código. Uno de los propósitos de la refactorización, ya mencionado varias veces, es la simplicidad. Un sistema más simple es también más fácil de entender, de modificar, de depurar, y todo ello acaba mejorando la calidad del producto. Además de hacerlo más simple, la refactorización debería dejar el código más claro, legible, mejorando su mantenimiento.

Es también una herramienta para eliminar redundancias, y elementos, incluso funcionalidades, innecesarios. Las capacidades añadidas y no usadas en el software son un desperdicio: una fuente de fallos, que además requiere atención y mantenimiento.

Finalmente, las pruebas, el *testing*, son la última herramienta de *Lean* para mejorar la integridad del software. En este caso, las pruebas son la medida de la calidad del producto (no falla) y del grado de ajuste a los requisitos definidos. Las pruebas deben hacerse desde el primer momento, ya que, cuanto antes se detecte un defecto o *bug*, más sencillo y económico será solucionarlo. Las pruebas son la actividad más visible del aseguramiento de la calidad, pero no son la única acción. La calidad es la característica central e indiscutible y no debe ponerse en riesgo ni sacrificarse ante ninguna otra consideración. Reducir pruebas para acelerar la disponibilidad es otra forma de generar desperdicio: código de baja calidad que mostrará sus defectos tarde, frente a sus usuarios, obligando a hacer un esfuerzo en solucionarlo muy superior a si se hubieran detectado y arreglado antes. Eso sin contar con la insatisfacción y desconfianza que generará en clientes y usuarios.

Visión global

El último de los principios de *Lean Software Development* enumerados en el libro de los hermanos Poppendieck es el de tener una visión global del producto, frente a la percepción de una colección de elementos o componentes. Ponen como ejemplo estos círculos viciosos:

- Un cliente quiere nuevas funcionalidades “para ayer”.
- Los desarrolladores lo entienden como un “¡hacedlo todo rápido! ¡Cueste lo que cueste!”.
- Como resultado, se hacen unos cambios con poco rigor y cuidado en el software.
- La complejidad del código aumenta.
- El número de defectos del código aumenta y, como consecuencia, o no se puede entregar rápido para arreglarlos o se entrega con problemas.

El segundo círculo vicioso dice:

- Las personas a cargo de las pruebas tienen trabajo atrasado.
- Como resultado, las pruebas se hacen mucho tiempo después de que el código se escribiera o se reducen para tratar de recuperar el ritmo.
- Los desarrolladores no reciben *feedback* temprano de su trabajo o algunas partes no se prueban adecuadamente.
- Hay más defectos en el código.
- Los *testers* tienen más trabajo.
- El *feedback* para los desarrolladores y las mejoras en la calidad se demoran aún más.

Estos círculos viciosos acaban teniendo como consecuencia un empeoramiento de los tiempos para añadir funcionalidades, así como una degradación de la calidad, lo que acaba impactando en los usuarios y afectando al producto en su conjunto.

Una organización *Lean* hace todo lo posible por optimizar el conjunto, no solo actividades o equipos individuales. Es muy habitual que los retrasos en proyectos y procesos se acaben debiendo a falta de comunicación y comprensión. Es un hecho que cruzar los límites de una organización, incluso internamente, es costoso.

Uno de los principios de los métodos ágiles es hacer que los equipos sean completos, multidisciplinares y, a ser posible, ubicados en el mismo espacio físico. Es una forma de romper las barreras de organización y comunicación. Claro que esto no es siempre posible, pues equipos muy grandes pueden ser inmanejables e ineficientes.

Pese a ello, muchos de los problemas se deben a que las organizaciones se estructuran en torno a funciones, roles o habilidades, no a productos o proyectos. Cuando se hace agrupando a las personas en torno a un objetivo concreto, como sacar adelante un producto o funcionalidad, hay un mayor sentido de control, propiedad o responsabilidad. El espíritu de

equipo y la colaboración se enfoca en la misma dirección, en lugar de ir hacia objetivos divergentes. Y todo ello redunda en un flujo de trabajo optimizado, que repercute en una mejora de la calidad.

Para empujar a los equipos en la dirección de la visión global, *Lean* propone el uso de métricas como herramienta o, más bien, un uso inteligente de las métricas. Porque cuando las personas se sienten medidas, reaccionan normalmente buscando mejorar la percepción que traslada la métrica. Pero, ojo, si no se definen cuidadosamente, en especial analizando sus consecuencias, las métricas pueden propiciar comportamientos poco eficientes. Piénsese en métricas ya dejadas de lado, como la cantidad de líneas de código. Si un desarrollador sabe que será medido por la cantidad de código que escriba, ¿qué aliciente tiene para hacerlo más compacto y simple? Al contrario, puede tener el aliciente negativo de agregar código innecesario que aumente el contador de líneas, aunque aumente la complejidad y deteriore la calidad.

Sin necesidad de usar medidas inadecuadas, se pueden propiciar comportamientos indeseados. Por ejemplo, si se piensa que los defectos del software son “culpa” del desarrollador, se estará fomentando una visión defensiva por parte de los programadores que se sentirán atacados cada vez que aparezca uno, negando incluso su existencia. Si el *tester* se siente dotado del “poder” de señalar *bugs*, se estará fomentando comportamientos bastante tóxicos, especialmente si, en este caso, desarrolladores y *testers* trabajan en equipos separados. Esto mismo puede decirse de otros colectivos y de todo tipo de métricas, que, aunque creadas con un propósito loable, pueden acabar como armas arrojadizas.

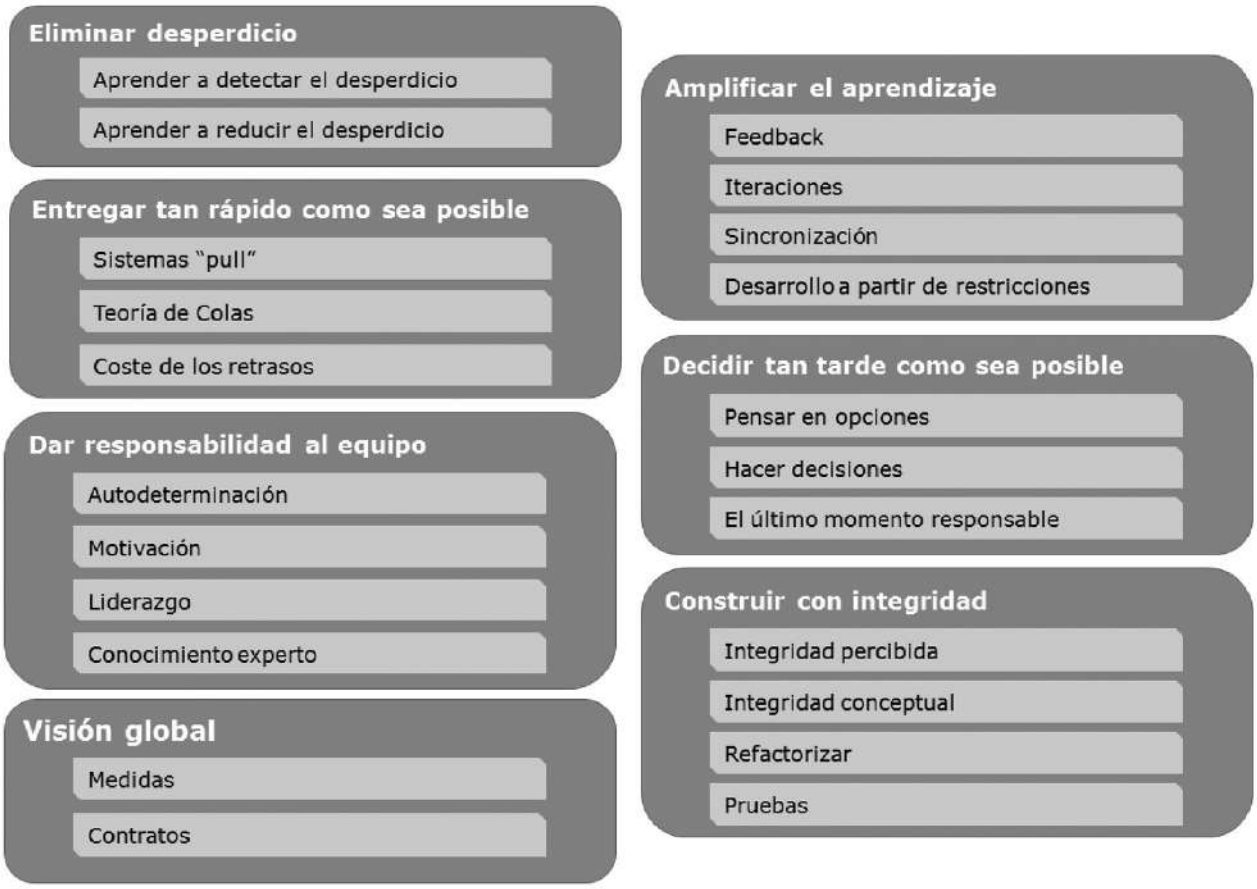


Figura 14.1. Principios y herramientas de Lean Software Development.

Reformulación de los principios de Lean Software Development

Como se ha comentado anteriormente, en el año 2006, los hermanos Poppendieck revisaron los principios, cambiando algunos de ellos. La nueva lista contiene varios de los principios ya enumerados: "eliminar el desperdicio", "crear conocimiento", "aplazar las decisiones" y "entregar tan pronto como sea posible".

Como novedad, se aportan los siguientes principios:

- **Calidad integrada:** El producto debe construirse con una calidad óptima desde el primer momento. Debe cubrirse todo tipo de pruebas, de forma que los defectos se corrijan lo antes posible. Esto implica hacer una construcción dirigida por una actividad constante de pruebas.

  Asimismo, debemos construir el producto de manera que no existan dependencias y que su arquitectura permita añadir nuevas funcionalidades en cualquier momento.

Nota:

No se debe esperar a probar el producto en la fase final: el coste de solucionar los problemas aumenta a medida que se avanza en la creación del producto. Cuando antes se detecte y solucione un defecto, más sencillo y económico resultará arreglarlo y será menor su impacto.

- **Respetar a las personas:** El foco de la mejora debe centrarse en las personas y en los procesos que hacen posible construir un producto, y no en mejorar exclusiva y directamente el producto en sí. De esta forma, se mejorará el producto actual y el sistema estará listo para poder crear otros productos con éxito en el futuro.
- **Optimizar el todo:** Hay que pensar desde un punto de vista global y orientado al largo plazo. La optimización de una pequeña parte del sistema puede afectar negativamente al conjunto del mismo.

  Desde el punto de vista de la construcción del producto, es frecuente solucionar localmente un problema, olvidando que esta parte que se acaba de arreglar forma parte de un conjunto más amplio y que un pequeño cambio local puede afectar a todo el sistema. Es necesario no perder la perspectiva de dónde se encuentra y tener muy presente la relación con las otras partes del producto, así como las dependencias de unas con otras.

Nota:

El cliente necesita un todo. No le aporta mucha información ver pequeños trozos de lo que espera sin saber cómo va a encajar el puzle final. Necesita, desde el principio, tener una visión global de lo que va a recibir. Esta es una característica de los métodos ágiles: la construcción incremental del producto.

Una aplicación de Lean Software Development al mundo real

Hay muchas experiencias de la aplicación de *Lean* a proyectos software, pero ninguna tan bien documentada y amena como la de Henrik Kniberg[57], una de las personas más conocidas y reputadas en el mundo *Agile*.

Entre sus varios libros y artículos, está "*Lean from the Trenches*[58], un texto disponible también como borrador de manera gratuita en Internet[59]. Escrito de manera muy directa y accesible (en inglés, lamentablemente no hay aún traducción española), expone la experiencia de un proyecto concreto abordado desde los principios de *Lean* y usando Kanban

como método. El proyecto es el desarrollo de una herramienta software para la policía sueca que no solo se desarrolla usando la filosofía *Lean*, sino que pretende inculcar esa misma filosofía al trabajo de la propia policía.

En su descripción, explica aspectos básicos de los requisitos y los criterios para desmenuzarlos, de la iteración con el cliente y cómo se gestiona y recoge el *feedback* de las iteraciones de trabajo. La organización del equipo de trabajo es también muy importante, entre otras cosas porque se trata de un proyecto grande que requiere un escalado de *Agile* y puede aplicarse la visión transversal que Kniberg ayudó a poner en marcha en empresas como Spotify: una división por área de conocimiento y otra más operativa de equipos que integran a personas de todos los perfiles requeridos. El proyecto en sí funciona como una especie de "Scrumban", mezclando elementos de *Scrum* y Kanban:

- Reuniones diarias al estilo de las *Daily meetings* para sincronización, aunque con una diferencia importante: hay reuniones de los equipos de trabajo; a continuación, de las personas de las distintas áreas funcionales; y, después, de representantes de cada equipo y área. De esa forma se trata de mantener la necesaria sincronización en un equipo de unas sesenta personas.
- Un tablero de trabajo, aunque con una orientación decididamente Kanban, ya que, por ejemplo, incluye las limitaciones del WIP (*Work In Progress*): es decir, si la limitación de trabajos que puede asumir el equipo de pruebas, por ejemplo, es 7, no se admitirá ningún trabajo nuevo mientras no quede un hueco libre.

  Dado que no se trabaja con el concepto de *Sprint* en Kanban, el equivalente del *Sprint Backlog* es una columna limitada en tamaño con las funcionalidades que esperan su turno para ser implementadas.
- Se mantienen algunas de las reuniones que marcan el ritmo en *Scrum*: una reunión de planificación periódica para solucionar desajustes y Retrospectivas periódicas para poner foco en la mejora continua (a las que llama "talleres de mejora" o *improvements workshops*).
- Usa métricas de velocidad, aunque distintas de las de *Scrum*, ya que se basa en contar las tareas completadas sin recurrir a puntos de historia o cualquier otro mecanismo que considere el "tamaño" del trabajo. Otra medida es el tiempo medio empleado en cada tarea o, más bien, el número de ciclos que se ha invertido desde su entrada hasta la salida de la cadena de trabajo. Luego, se usa esa medida o, más bien, los valores medios, para determinar si hay o no una mejora en el equipo y si se gana en velocidad (menos tiempo en la cadena).

En el proyecto, se puso una atención especial a la calidad y la forma de gestionar los defectos encontrados en el producto. Por ejemplo, en lugar de concentrar las pruebas como una parte final del ciclo de trabajo, las desplazan y fragmentan para que sean una actividad continua que se desarrolla a la vez que la construcción del software. También insiste mucho

en la solución temprana de *bugs*.

Lo que quizá es menos convencional es tratar los *bugs* en flujo separado tipo Kanban y limitar el número que ponen en la cola para ser solucionados. Debidamente priorizados, se limita la cantidad de esfuerzo que se dedica a la solución de defectos. La idea es evitar que la resolución de *bugs* compita con los recursos necesarios para hacer progresar la construcción del producto y transmitir cierta forma de tranquilidad psicológica al evitar ver que la cantidad de defectos es inabordable. Ambos puntos de vista son bastante discutibles, pero la experiencia de hacerlo así ha sido satisfactoria según el autor.

Un último aspecto bastante llamativo, dada la cantidad de personas que trabaja en el proyecto, es el uso exclusivo de paneles físicos para hacer el seguimiento del proyecto, en lugar de utilizar herramientas informáticas. Es decir, toda la gestión se hace por medio de tarjetas de papel, paneles en las paredes y *post-it*. Teniendo en cuenta el tamaño del equipo y del proyecto, supone un esfuerzo considerable de gestión, aunque se gane enormemente en claridad y comprensión.

En definitiva, en el libro de Kniberg se puede ver en funcionamiento un equipo en el que se ha hecho un gran esfuerzo en simplificar y reducir el desperdicio, anteponer la calidad, acelerar la entrega de producto y la obtención de *feedback*, y se cuida especialmente al equipo para ofrecerle un entorno altamente productivo. Exactamente lo que *Lean* propone.

---

56 *Lean Software Development. An Agile Toolkit*. Mary and Tom Poppendieck. Alistair Cockburn and Jim Highsmith, Series Editors. 2003.

57 Página Web de Henrik Kniberg en su empresa.

58 *Lean from the Trenches*. Henrik Kniberg. Pragmatic Bookshelf. 2011.

59 *Lean from the Trenches*. Henrik Kniberg. 2011.

Tercera parte
El éxito en los proyectos

15 Lean Startup

En este capítulo aprenderá a:

- Comprender la visión *Lean* aplicada a la empresa.
- Conocer las fases de *Lean Starup*.
- Entender conceptos como PMV y pivotar.

Empresas ágiles: Lean

En el capítulo anterior, se ha hablado de la aproximación *Lean* aplicada a la fabricación industrial y, sobre todo, de cómo esos principios ágiles ayudan a desarrollar proyectos software. Sin embargo, se ha insistido varias veces a lo largo de este libro en que los métodos ágiles pueden formar parte de actividades de todo tipo, especialmente de aquellas que contengan una alta incertidumbre.

Lean Startup se conoce popularmente como un método que ayuda a crear nuevas empresas de corte tecnológico, pero, en realidad, es una forma ágil (iterativa, flexible, incremental, eficiente...) de definir un modelo de negocio. O, al menos, es así como lo describe una de las firmas de referencia en esta comunidad, Eric Ries, autor de *Lean Startup*[60], uno de los textos más conocidos sobre esta forma de trabajo.

De acuerdo con este autor, una *startup* sería una organización que define un modelo de negocio, mientras que una empresa sería la que lo ejecuta. De esta forma, una *startup* no tiene porqué ser una empresa de pequeña creación (y en absoluto tecnológica), puede ser perfectamente un departamento de una organización ya asentada que quiere explorar nuevos caminos[61].

Sea pequeña o parte de algo más grande, la *startup* se encuentra en un entorno de gran incertidumbre, una característica común a los distintos métodos ágiles presentados a lo largo de este libro. Por ese motivo, en *Lean startup* se habla con frecuencia de experimentar y de hipótesis que se validan.

Se presenta la actividad de una *startup* como una investigación científica, donde se formulan hipótesis, se define la forma de validarlas, se realizan ensayos, se obtienen métricas y se contrasta la realidad con las ideas inicialmente formuladas.

Esas hipótesis se van afinando en ciclos iterativos hasta alcanzar el objetivo del experimento: el producto, servicio o modelo de negocio que definirá a la empresa que posteriormente lo llevará al mercado.

El otro autor más reconocido en el mundo del *Lean Startup* es Steve Blank, profesor de Eric Ries y autor de *The Four Steps of Epiphany*[62] y, en colaboración con Bob Dorf, de *The Startup's Owner Manual*[63]. Estos dos textos son más académicos y complejos que el de Ries y también menos populares, aunque en realidad son los que aportan mayor consistencia intelectual al método.

Todos ellos, junto a otros autores menos conocidos, son muy activos en Internet donde mantienen *blogs* y publican artículos, y participan continuamente en conferencias y congresos. *Lean Startup*, como el resto del mundo *Agile*, es una comunidad activa y en movimiento.

Todos estos autores invocan los mismos procedentes que mencionábamos en el capítulo anterior: *Lean manufacturing*, Toyota, *Lean thinking* (el precedente del *Lean software development*) y ya, en el campo empresarial, la "Teoría de las Restricciones" (TOC, *Theory of Constraints*). Un caso mucho más cercano es el método llamado "*Customer development*", desarrollado por Steve Blank, precedente de su primer libro y que ya contenía muchos de los elementos posteriores (como el concepto de pivotar, el producto mínimo viable o la idea de validación de hipótesis).

Los principios básicos que sustentan *Lean Startup* ya deberían ser viejos conocidos para un lector de este libro: minimizar el desperdicio, una cultura de mejora continua y la importancia de tomar medidas.

A continuación, se describen los fundamentos de este método y filosofía de trabajo. Obviamente, será una visión limitada y centrada sobre todo en aspectos de método, que no puede reflejar la riqueza y complejidad de *Lean Startup*, pero que puede servir como introducción y punto de partida para conocer más.

No se puede terminar esta sección sin agregar otra referencia sobre el mundo *Lean Startup*, ahora en español. La persona más relevante y reconocible en esta comunidad en España es Mario López de Ávila, a quien se puede seguir en su *blog Nodos en la Red*[64], que coordina alguno de los foros más activos en Meetup como "*True Lean Entrepeneurs*"[65] y es coautor del libro *España Lean Startup 2013*[66]. Mario, además, prologa la edición actual de este libro.

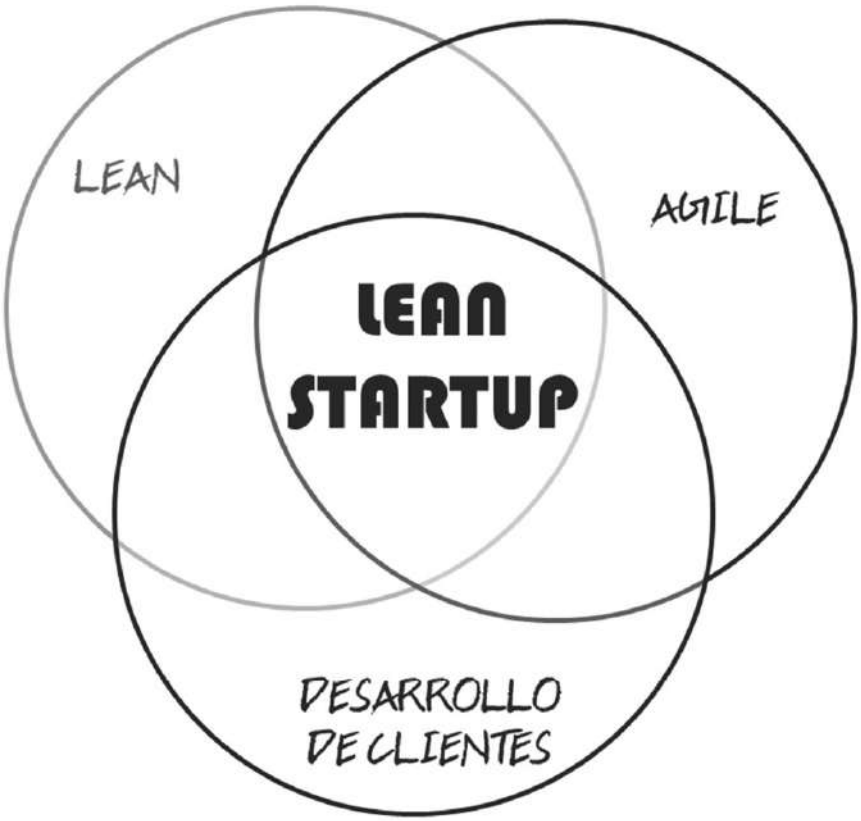


Figura 15.1. Lean Startup nace de tres influencias diferentes.

Startups como proyectos iterativos

El imaginario colectivo está lleno de historias de jóvenes brillantes desarrollando ideas rupturistas en el garaje de su casa y, a continuación, gobernando empresas colosales que cambian la vida de las personas. Como mito está bien, pero la realidad, como de costumbre, lleva esos sueños a tierra.

Las ideas brillantes por sí solas no bastan. Para empezar, el brillante producto o servicio que llega a manos del público no nació así: era algo más limitado, posiblemente con elementos que se acabaron desechando, en muchos casos completamente distinto de la idea original. Es decir, el producto evoluciona.

Al mismo tiempo, la forma de gestionar un equipo de pocas personas no es válido para organizar una empresa compleja. Es decir, las *startups* necesitan modelos de gestión para cumplir su propósito principal y para evolucionar y tomar forma de empresa. Ya se ha mencionado al principio qué distinguiría a una *startup* de una empresa: una diseña un modelo de negocio y producto, y la segunda lo ejecuta. Bajo esta definición, una *startup* puede encajar tanto en la imagen habitual del pequeño grupo de jóvenes entusiastas en su garaje, como en un proyecto o equipo dentro de una empresa constituida y asentada. De una o de otra forma, una *startup* es un artefacto que nos permite reducir la incertidumbre. Ese es su principal cometido y, como ya hemos visto por otros métodos ágiles, eso requiere de una aproximación progresiva e iterativa y obtener *feedback* tan pronto como sea posible.

Mientras una empresa requiere un plan detallado y es muy eficiente en su ejecución, una *startup* precisa de herramientas para aprender de su experiencia y eso implica ser capaz de equivocarse y sacar partido de sus errores. Es decir, mientras una empresa trata de no fracasar en la ejecución, para una *startup* el fracaso forma parte del ciclo de aprendizaje. Una *startup* fracasaría si no fuera capaz de aprender de sus errores. Como se ve, es un significativo cambio de paradigma.

Como en otros métodos ágiles, la *startup* necesita de *feedback*. Ya se ha visto que en *Scrum*, al final de cada iteración o Sprint se presenta el producto evolucionado al *Product Owner* y los *stakeholders*. Es ahí donde se espera su aceptación o correcciones para aplicarlos en las próximas iteraciones y ayudar a definir los próximos cambios. En *Lean Startup*, de una forma similar, el producto se presenta a la opinión de los clientes, para enriquecer o corregir su definición.

En una empresa convencional, la forma de lanzar un nuevo producto o servicio se basa en la planificación: estudios de mercado para conocer e interpretar la opinión del público; trabajos del departamento de I+D o marketing para definir completamente en qué va a consistir el producto y cómo construirlo y comercializarlo; detallados modelos de negocio

para analizar su viabilidad y rentabilidad. Cuando el producto llega al mercado, ya es un producto final, no un prototipo o un artefacto en evolución.

Sin embargo, en *Lean Startup*, el producto o servicio llega al mercado cuando es poco más que un diseño y solo es capaz de cubrir sus funciones más básicas. El estudio de mercado es aquí la reacción del consumidor: si lo usa o lo compra, o si no lo hace. El diseño del producto está sujeto a cambios y adaptaciones continuos, que incluso pueden alterar completamente su naturaleza. El modelo de negocio es otra de las incógnitas que hay que despejar y se define y refina progresivamente, no se parte de un detallado plan de antemano.

Para alguien familiarizado con los métodos ágiles, vivir con estas incertidumbres e incógnitas y pensar en resolverlas gradualmente no supone esfuerzo ni inquietud. Cuando viene del mundo en el que los planes se diseñan meticulosamente y se ejecutan sin cambios, vivir sin certezas produce vértigo. Pero ya sabemos que el mundo en realidad es así: cambio, incertidumbre, riesgo, dudas. Las previsiones, modelos, planes, son refinadas representaciones de la realidad que nos trasladan una falsa sensación de control sobre el futuro. Como cualquier otra actividad, desarrollada en un mundo lleno de incertidumbres, las *startups* necesitan otras herramientas y otra visión que saque partido en lugar de luchar inútilmente contra la cambiante naturaleza de lo desconocido.

No se olvide tampoco lo que decía al hablar del *Sprint* 0, ese momento en el que se definía la visión del proyecto, que va a ser, tal y como decíamos en el capítulo 2: “... un resumen de las metas a medio y largo plazo a las que se quiere llegar. Es una imagen mental de dónde se quiere estar o qué se quiere tener en un determinado plazo de tiempo. Es una información de alto nivel y de propósito general. No tiene que ser detallada, pero sí tiene que ser sencilla y clara. Sin ambigüedades. La sencillez es la clave para definir el objetivo que se quiere alcanzar...”.

Esa Visión será la referencia estable, la Guía, el Norte, el punto de referencia, mientras se pone en cuestión todo lo demás, y se inicia la búsqueda que lleve a esa Visión... o se decida cambiarla.

La principal herramienta que aporta *Lean Startup* para cumplir la Visión es una aproximación iterativa y cíclica a la hora de encontrar una solución al problema de satisfacer una necesidad del mercado y hacer de ello una actividad sostenible económicamente.

Así pues, *Lean Startup* trabaja en ciclos o iteraciones que en su caso reciben el nombre de “ciclo Crear-Medir-Aprender” (*Build-Measure-Learn*), en el que se enuncian hipótesis, se diseñan y ejecutan experimentos que las validen. El objetivo se convierte en obtener *feedback* que indique que se debe perseverar en esa dirección o cambiarla, pivotar, en el próximo ciclo.

Crear

Recuerde *Scrum*. Se decía que un plan inamovible y completamente definido de antemano

no sirve para nada. Sin embargo, al comienzo de cada iteración, se planificaba qué se iba a hacer durante un periodo definido o *Sprint*. De la misma forma, aunque se reniegue de los diseños cerrados y guiados por costosos y detallados estudios de mercado, eso no quiere decir que el punto de partida no tenga un diseño y un plan. Lo que cambia es la naturaleza, ya que está pensado para cambiar, y el alcance, que solo se desarrolla durante la iteración.

Así pues, el primer paso es crear o definir el producto.

El empuje inicial para esa creación es hacer una serie de asunciones: "Todo el mundo necesita llevar en su bolsillo un ordenador conectado a Internet con capacidades multimedia", "la gente desea expresar continuamente sus pensamientos en textos muy cortos", "hace falta una tienda por Internet que ofrezca todo tipo de productos", "el público está deseando desentenderse de conducir y espera que los coches lo hagan por sí mismos", "el público desea tener una experiencia de primera mano en un parque temático poblado con bestias extintas"... Muchas de estas ideas fueron disparatadas en el momento de crearse, algunas han llegado a la realidad demostrando su validez, otras siguen sin tener sentido o son simplemente irrealizables.

En un modelo de empresa convencional, se haría un diseño muy detallado del producto, se analizarían sus costes, se elaboraría un meticuloso plan de negocio, se planificarían todos los pasos hasta el lanzamiento, que se ejecutarían de acuerdo al plan. En *Lean Startup*, se hace todo lo posible por poner un producto en el mercado y validar si es eso lo que esperan los consumidores o no. Por eso, más que alcanzar la definición del producto o servicio, lo importante es saber qué preguntas queremos plantear a nuestros posibles clientes y hacerlo rápido, porque no tenemos los recursos para completar todo el plan. Solo, y como mucho, para hacer unas cuantas preguntas.

Estas asunciones y preguntas son las hipótesis que se quieren validar. Y para responderlas hay que seleccionar cuidadosamente los medios para hacerlo.

Piénsese la idea de una tienda por Internet. Ahora se sabe de antemano que sí es algo que aceptan los consumidores y que requiere una compleja estructura logística, de procesos, con medios de pago, gestión de la información, soporte y un largo etcétera de funciones avanzadas. Ahora pensemos en un tiempo en el que nadie sabía si una tienda por Internet era una buena idea o un sinsentido. ¿Qué era necesario para despejar esa incógnita? Solo un pequeño sitio Web, mantenido manualmente, con un limitado catálogo, donde los pedidos se hacen por correo y se cobran por contrarrembolso. Ante un caso como este, la primera prioridad sería saber si el público estaría dispuesto a comprar por Internet. Más tarde, se tendría que averiguar si, además, estaría dispuesto a usar una tarjeta o a considerar otros medios de pago, buscar en un catálogo, recibir recomendaciones...

Una leyenda apócrifa, pero válida para explicar este concepto, cuenta que el primer cajero automático era en realidad una maqueta en la que una persona realizaba las acciones que en el futuro debería llevar a cabo un ordenador[67]. Lo que se quería era comprobar la reacción de los clientes. Si esta era favorable, se procedería a completar el desarrollo de las funciones

necesarias para convertirlo en un sistema realmente automático. Lo mismo se puede decir de nuestro ejemplo de la tienda en Internet: hay que poner cuanto antes en la calle una Web que permita comprobar si es o no una idea aceptable. Si lo es, se irán planteando y validando otras hipótesis que enriquezcan el producto.

Así pues, ya se tendrá un producto, con sus características esenciales. Ese tipo de producto, recibe en *Lean Startup* un nombre y significado especiales: PMV.

Producto Mínimo Viable

El PMV o MVP (*Minimum Viable Product*) es uno de los conceptos característicos de *Lean Startup*.

Tal y como se ha contado, se trata de una herramienta para obtener respuestas rápidas. De ahí que tenga que ser simple, sin elementos superfluos, porque su cometido es llegar cuanto antes al mercado para obtener el *feedback* de los clientes potenciales. A la hora de decidir en qué consiste ese PMV, hay que poner el foco en las preguntas que se quieren ver respondidas y reconocer que las respuestas que se obtengan tampoco serán completas y definitivas. Hay que hacer un ejercicio de priorización muy riguroso y, al mismo tiempo, ser conscientes de que las respuestas que se reciban no serán completas y tras la prueba permanecerá un cierto grado de incertidumbre. Se podría diseñar y refinar para tratar de alcanzar el producto perfecto para nuestros fines, pero se podría caer también en una “parálisis por análisis”, en lugar de ganar tiempo, que es lo que realmente tiene importancia. Un PMV es una herramienta de aprendizaje y la palanca para obtener *feedback* rápido y con el menor esfuerzo. Y no es un fin en sí mismo: el PMV evoluciona en cada iteración a partir del conocimiento adquirido.

A cambio, sobre todo en las primeras etapas, el PMV puede parecer tosco, simple y no estar construido con la máxima calidad. De hecho, el propio grado de calidad es uno de los atributos que se puede querer validar: ¿el posible público prefiere un producto barato y funcional, aunque con fallos?, ¿o, por el contrario, no tolera el menor defecto?

Como se podrá entender fácilmente, estos productos de naturaleza cambiante, cuya definición se ajusta en cada iteración y que pueden incluso cambiar radicalmente, encajan mejor en ciertas categorías, en concreto el software. La naturaleza intangible y moldeable de un programa o un servicio basado en software facilita esa evolución, además de ser también una herramienta a la hora de simular aplicaciones o servicios complejos. Eso no quiere decir que el software sea el único medio o el más adecuado: piénsese en una tienda que puede ir variando el catálogo de productos sobre una infraestructura muy básica, como unas estanterías, unas mesas, unas pocas muestras. O una revista cuyo contenido y orientación se revisa cada semana. O un servicio de transporte cuyas rutas, frecuencias y tarifas cambian, y que puede apoyarse en una flota alquilada y también cambiante mientras acaba de descubrir su lugar en el mercado.

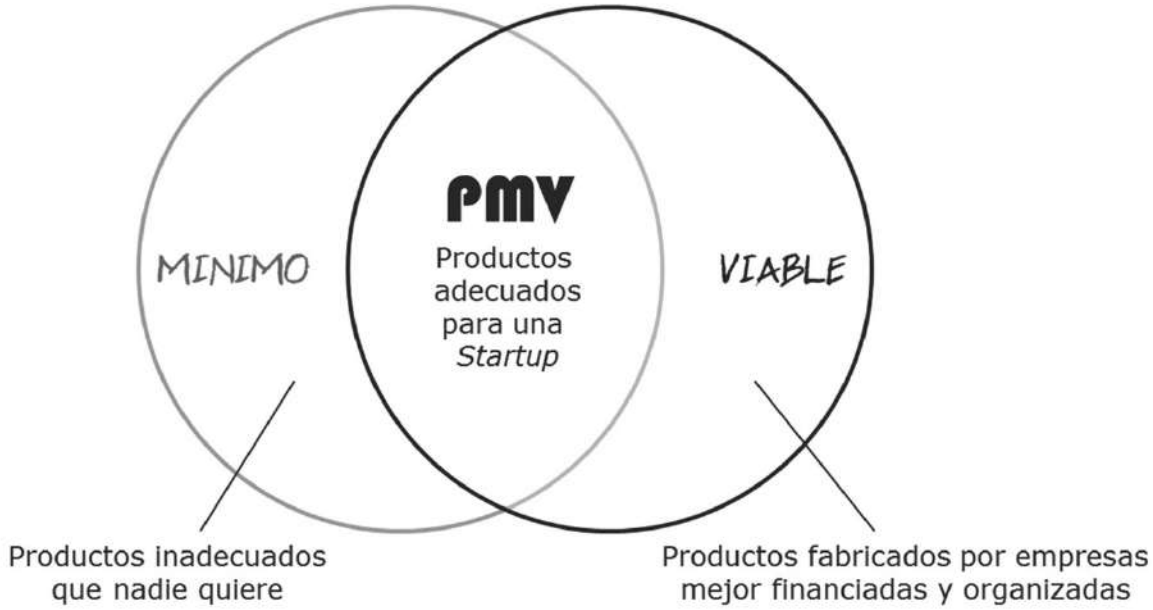


Figura 15.2. El Producto Mínimo viable es un compromiso.

A veces el PMV ni siquiera es un producto. ¿Qué es más barato y rápido para medir la aceptación de un concepto que un anuncio? Si los posibles clientes quieren saber más, incluso adquirirlo a partir de publicidad, se tiene una prueba palpable de la validez del concepto a un coste muy inferior a desarrollar el producto. Piénsese en el ejemplo del Tesla Model 3, con reservas del orden de cientos de miles de unidades (incluyendo el pago de una señal) a partir del anuncio de la futura fabricación del vehículo.

Medir

El PMV definido en el primer paso del ciclo es una herramienta de aprendizaje que nos ayuda a despejar dudas, pero ¿cómo lo hace exactamente? Junto a las hipótesis que el Producto Mínimo Viable nos ayuda a validar, hay que definir la forma de evaluarlas. El paso al mercado para hacer esa evaluación es lo que en *Lean Startup* se denomina medida, y ayudará a obtener un aprendizaje valioso a partir de la reacción, del *feedback* de los clientes.

¿Por qué es tan importante medir? Porque es un conocimiento contrastado y validado. No se trata de una presunción, una intuición, una idea preconcebida o un deseo. Son valiosos datos empíricos y *feedback* de primera mano.

Además, hay que recordar que, en un entorno de alta incertidumbre, los objetivos que se fijen serán de poca utilidad. En una empresa convencional, antes del lanzamiento de un producto al mercado, se fijarían unos objetivos de venta. Esos objetivos no dejan de ser una forma de predecir el futuro. En cambio, desde un punto de vista *Lean*, no se predice el futuro, sino que se va a averiguar si ese producto tiene mercado y en qué medida. Fijar un objetivo

de venta no tiene ningún sentido cuando se carece de referencias previas.

Una vez se tienen esas referencias y el PMV empieza a evolucionar, ya es posible ver cómo se dirige hacia un objetivo ideal derivado de la visión. A partir del punto inicial de partida, se va poniendo el foco en aspectos definidos y controlados del producto o servicio para evaluar su aceptación de forma individual.

Por ejemplo, en una tienda electrónica, ¿tendrá mayor aceptación un buscador o un sistema de pago seguro? Se pueden añadir esas funcionalidades una a una para estudiar el impacto de hacerlo antes de consolidarlas o de sustituirlas por otras.

La forma de valorar la mejora debe estar claramente definida de manera que sea una medida a salvo de interferencias, fácil de entender, fácil de obtener, con impacto y repercusiones, y que no influya provocando consecuencias no deseadas. Esas métricas son las que van a dirigir el proceso, permitiendo ver si la evolución es positiva, por lo que hay que perseverar o, si no lo es, por lo que hay que cambiar, o, en una expresión propia de *Lean Startup*, "pivotar".

Hay dos tipos de métricas o indicadores: los vanidosos y los accionables.

Los primeros pueden medir de manera indirecta el éxito o crear una falsa sensación de que se está alcanzando. Tómese, por ejemplo, el número de visitas a nuestra tienda de electrónica. Puede ser una medida de su popularidad y podría indicar que hemos acertado al atraer el interés del público, pero ¿se transforma necesariamente en una venta, en un rendimiento económico?

El porcentaje de visitas que se convierten en ventas, el volumen total de estas o el valor medio de la compra por cliente sí que son indicadores accionables que realmente muestran la validez de las hipótesis planteadas.

De hecho, un crecimiento de un indicador vanidoso como el número de visitas que no tenga la contrapartida de crear valor, está mostrando que el esfuerzo invertido e inútil y que se está invirtiendo en un aspecto que no produce retorno, generando en su lugar desperdicio.

Aprender

La última etapa del ciclo *Lean startup* es el momento de ver qué se ha aprendido: se ha planteado una hipótesis, se ha diseñado un experimento (PMV) que la valide (métricas), así que se tienen las herramientas para saber si se ha acertado.

Si las métricas se han diseñado adecuadamente para medir el progreso de la idea y no para resaltar aspectos que puedan parecer interesantes pero también irrelevantes desde el punto de vista de negocio, se habrá obtenido información valiosa y aprendido algo nuevo para refinar la idea de negocio. Ese nuevo conocimiento, que ha sido contrastado de forma empírica en condiciones de mercado, tiene una utilidad clara a la hora de definir los próximos pasos.

En *Lean Startup* solo hay dos acciones posibles: perseverar o pivotar. Perseverar quiere

decir que nuestra hipótesis era adecuada y que debemos aumentar e incluso potenciar ese aspecto que estábamos valorando. Pivotar, por el contrario, quiere decir que la hipótesis no era válida, que hay que desecharla y hacer un cambio. Ese cambio puede ser un ajuste puntual, pero la expresión pivotar se aplica más sobre cambios radicales que alteran significativamente la naturaleza del producto e incluso del negocio.

Hay ejemplos muy conocidos de empresas que han cambiado o pivotado completamente su actividad. En algunos casos dentro de un mismo segmento de negocio, como Netflix, que inicialmente funcionaba como un servicio de alquiler de DVD por correo. En otros, cambiando de arriba abajo, como el juego de rol *Neverending* que acabó siendo la Web Flickr para aficionados a la fotografía; o Nintendo, originalmente un fabricante de naipes.

No se podrá insistir lo suficiente en la importancia de las métricas, ya que de ellas depende tomar una decisión acertada o errónea. Perseverar en un camino que no lleva a ningún sitio o pivotar creyendo que la hipótesis no se ha verificado. En el primer caso, se está hablando de generar desperdicio; en el segundo, de perder una oportunidad; y, en los dos, de gastar una de las pocas oportunidades con las que cuenta la *startup* para reducir la incertidumbre y alcanzar su propósito.

Si el experimento está bien diseñado, la hipótesis tiene sentido, y la métrica se ajusta a su propósito, se podrá contar con un conocimiento adicional que permitará planificar el próximo paso. En realidad, este modo de trabajo no es diferente del aplicado en métodos como *Scrum*, donde el producto está delineado en sus aspectos más básicos, y un proceso de descubrimiento *sprint* a *sprint* por parte de *stakeholders* y equipo permite ahondar en el diseño final de lo que se está construyendo y completarlo.

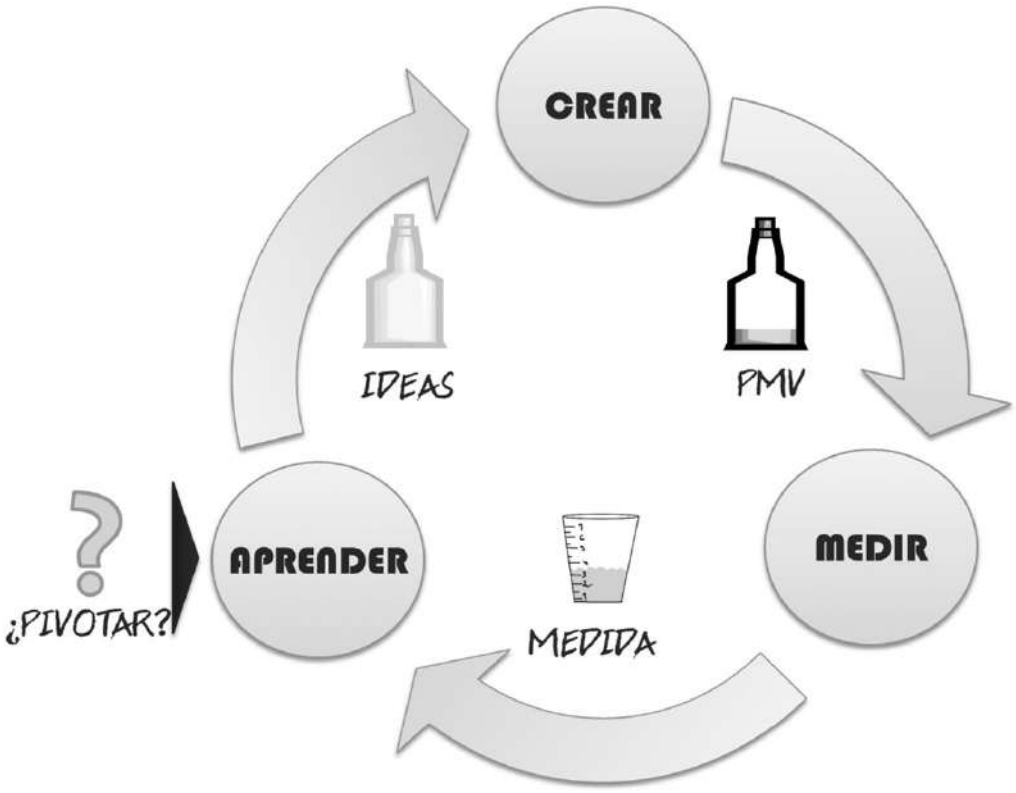


Figura 15.3. El ciclo Crear-Medir-Aprender de Lean Startup.

Seguramente la decisión más difícil en una *startup* es la de pivotar, ya que debe romper una doble inercia: la de la historia previa y, sobre todo, la de las asunciones e ideas interiorizadas en todo el equipo. Pivotar es una corrección diseñada para probar una nueva hipótesis básica sobre el producto y la estrategia de la *startup*. Acertar en el momento y camino al pivotar puede marcar la diferencia.

Se distinguen varios tipos de pivotes:

- Acercamiento, que potencia una característica individual hasta convertirla en el propio producto (caso de Flickr).
- Alejamiento, por el contrario, convierte el conjunto del producto en una parte de otro mayor. Paypal empezó como un sistema de criptografía para intercambiar dinero entre dispositivos Palm; eso dio paso a un medio de pago universal basado en Internet.
- Segmento de consumidor, para enfocarse a un tipo diferente de clientes.
- Necesidad del consumidor, cuando el conocimiento adquirido en el proceso permite descubrir un problema potencialmente más relevante para los clientes y también con más impacto si se soluciona.
- Arquitectura del negocio, para cambiar entre alguno de los paradigmas de negocio, como ir a un mercado masivo o a otro especializado.
- Captura del valor, para decidir la mejor forma de obtener ingresos. Por ejemplo, se puede dar el salto de un servicio bajo suscripción a otro apoyado en la publicidad o, simplemente, en crecer lo suficiente como para interesar a un tercero que adquiera la empresa para incorporarla a su portfolio.
- Motor del crecimiento, pivotando entre atraer a los clientes con publicidad, por ejemplo, o hacerlo sin ella, de forma viral. O fomentar la “adicción” del servicio para tratar de vender otros.
- Canal, para vender el producto en según qué tipo de tiendas físicas, o solo por Internet o solo puerta a puerta.
- Tecnología, basta con recordar el ejemplo de Netflix: dejar de enviar contenidos físicamente por correo y usar el *streaming* a través de Internet.

Al final de este punto en el ciclo, ha llegado el momento de seguir adelante, preparando un nuevo experimento con una nueva hipótesis que validar. También puede ser el momento de hacer el pivote más difícil: abandonar. O de considerar que el camino avanzado hasta ahora, con un producto o servicio ya presente en el mercado, lleva a la *startup* a convertirse en una verdadera empresa, con un modelo de gestión diferente, pensado para sacar el máximo partido a nuestro trabajo en el mercado, más que a seguir experimentando y aprendiendo.

Por el camino, además de ciclos en los que la idea inicial ha sufrido todo tipo de cambios

y mutaciones, se ha obtenido información muy valiosa, incluso en el caso en el que se abandone por completo la idea (ese hecho es en sí mismo consecuencia de determinar que no se estaba dirigiendo al mercado correcto o con el producto oportuno).

El conocimiento adquirido se plasma de varias formas, especialmente como el producto o servicio que se ha diseñado, que recoge todas las ideas e hipótesis validadas en el proceso. Hay, además, otros resultados, que se suelen asociar a *Lean startup*, los distintos *canvas* o lienzos de negocio. Se ha preferido evitar mencionarlos para dejar que se desarrollen en el siguiente capítulo dedicado específicamente a las herramientas para el diseño de productos y requisitos.

---

60 *Lean Startup*. Eric Ries. Deusto. 2012.

61 Para entender cómo se aplica *Lean Startup* en grandes empresas, se recomienda el artículo *Lean Elephants*, por Susana Jurado y Maria Olano, con participación de alguno de los autores. http://www.tid.es/sites/526e527928a32d6a7400007f/assets/53bfe9f128a32d6733001f37/ Lean_Elephants.pdf o en http://es.slideshare.net/InstitutLeanFrance/leanelephants-lean-product-development-in-a-large-organization-by-susanajurado.

62 *The Four Steps to the Epiphany: Successful Strategies for Products That Win (Inglés).* Steve Blank. K&S Ranch. 2005.

63 *The Startup's Owner Manual (Inglés).* Steve Blank, Bob Dorf. K&S Ranch. 2012.

64 *Nodos en la Red*: http://nodosenlared.com/.

65 *True Lean Entrepeneurs*: https://www.meetup.com/es-ES/trueleanentrepreneurs/.

66 http://nodosenlared.com/espana-lean-startup-2013/.

67 En realidad, el primer cajero, instalado en Londres en 1967, ya era un sistema automatizado, aunque no se basara en el uso de tarjetas, sino de cheques impresos con tinta radioactiva.

16 El lienzo de modelos de negocio

En este capítulo aprenderá:

- A identificar las cuatro áreas principales de un negocio.
- Los nueve elementos de un modelo de negocio.
- Cómo utilizar el lienzo del modelo de negocio.

Cuando tienes una gran idea que te puede hacer especial en el mercado, te invaden numerosas dudas. ¿Será rentable? ¿Qué recursos necesito para hacerla realidad? ¿Cómo la doy a conocer? ¿Cómo la hago llegar a los clientes?

Un alto porcentaje de los proyectos emprendedores fracasan por fallos o problemas con su modelo de negocio. Tal y como se indica en el capítulo 15, el modelo de negocio es una las incógnitas que hay que despejar y se define y refina progresivamente; no se parte de un detallado plan de antemano.

En este capítulo, vamos a exponer cómo crear el modelo, representando de manera sencilla cómo una organización crea, entrega y captura valor. Tenerlo claro es clave para discutir con efectividad y tomar decisiones acertadas. También ayuda a que todos los interlocutores usen un lenguaje común y así evitar ambigüedades o malos entendidos.

El hecho de trabajar con métodos ágiles no es una excepción. En el capítulo 3 se expuso la necesidad de un *Sprint* 0 o etapa previa a la construcción de un proyecto. Es necesaria una gestación que facilite el resto de las actividades que se realizarán posteriormente.

Piénsatelo antes de arrancar un negocio

¿Tienes espíritu emprendedor? ¿Estás constantemente pensando sobre cómo crear valor y construir nuevos negocios o cómo mejorar y transformar tu organización? ¿Estás tratando de encontrar formas innovadoras de hacer negocios para reemplazar a las desactualizadas o antiguas? Si tu respuesta es sí a alguna de estas preguntas, ¡bienvenido a nuestro grupo!

Así es cómo comienzan su libro *Business Model Canvas*[68] los autores Alexander Osterwalder e Yves Pigneur.

Osterwalder completó su tesis doctoral en el año 2004 y fue dirigida por el profesor Yves Pigneur. El tema tratado fue la innovación en los modelos de negocio. Su propuesta de innovación se hizo muy popular a través del *blog* de Alexander y numerosas compañías lo aplicaron, y siguen aplicando, con éxito para la creación de sus modelos de negocio. Fue en 2006 cuando se inició la aventura de plasmar en un libro todo este conocimiento y, en 2009, Osterwalder, Pigneur y un equipo de 470 co-autores de 45 países diferentes publicaron el libro *Business Models Canvas*, compartiendo todos los detalles de su atractivo y popular modelo. Lo introducen afirmando que “el lienzo de un modelo de negocio proporciona herramientas para describir, visualizar, evaluar y modificar modelos de negocio con un lenguaje común”.

Este lienzo está diseñado para ayudar a ejecutivos, consultores, emprendedores, gestores y líderes de todo tipo de organizaciones. Parten de la base de que un negocio tiene cuatro áreas principales: la oferta, los clientes o usuarios, la infraestructura necesaria y la viabilidad financiera. Estas cuatro áreas pueden, a su vez, dividirse en nueve elementos o bloques:

segmentos de mercado o usuarios, propuesta de valor para cada segmento, los canales para llegar a los clientes, relaciones establecidas con los clientes, fuentes de ingresos, actividades y recursos necesarios para generar valor, colaboradores o *partnerships* claves y coste de la estructura del modelo.

Figura 16.1. Areas de negocio.

La propuesta de Osterwalder es representar estos nueve bloques en un lienzo o *canvas* y definir las relaciones entre los bloques. Para identificar el contenido de cada uno de ellos, plantean una serie de preguntas, que se indican en los siguientes apartados.

Los nueve elementos

Tenemos una idea, una propuesta de valor que puede hacernos especiales en el mercado. Es lo que nos hace diferentes y, por eso, lo situamos en el centro, en el corazón de nuestro lienzo. Sin usuarios y clientes no somos nadie y, por eso, los situamos a la derecha de nuestro núcleo. Se tiene pues que identificar para qué organizaciones o personas se está creando valor, darles a conocer el producto, hacérselo llegar utilizando canales y crear vínculos con ellos. Hay que mirar ahora hacia dentro, hacia lo que los demás no ven y que son todos esos recursos clave para nuestro negocio y esas actividades que hay que hacer y se representará a la izquierda del lienzo. Se acompaña de los posibles socios y aliados necesarios para el modelo. Finalmente, a modo de pilares que sostiene el modelo, en la base del lienzo se representará, a un lado la estructura de costes del modelo y al otro las fuentes de ingresos y beneficios.

Véase ahora el detalle de cada uno de estos nueve bloques.

Segmentos de mercado

¿Para quién creamos valor? ¿Cuáles son nuestros clientes más importantes?

Sin usuarios o clientes "rentables" no hay negocio y esto, que parece obvio, con frecuencia es olvidado. Muchos proyectos nacen y se construyen enfocados al producto y no a los clientes y usuarios potenciales. Esto es catastrófico porque, aunque la idea de negocio sea maravillosa y el producto esté construido estupendamente bien, tal vez no tenga ningún interés comercial y puede ocurrir que no se encuentre a nadie dispuesto a pagar por él. Y aquí comienza el drama, cuando se descubre que la idea genial no cubre ninguna necesidad ni soluciona ningún problema.

El modelo de negocio se debe basar en los clientes, así que lo primero que se tiene que hacer es "salir a buscarlos", empezar a estudiarlos y a conocerlos para ofrecerles algo que necesiten o les interese.

Una organización sirve a uno o varios segmentos de clientes. En este bloque se debe reflejar las personas u organizaciones para las que nuestro producto ofrecerá valor. Aquí se tienen en cuenta tanto a los usuarios individuales como a los clientes que pagarán por el servicio, así que se debe plantear a qué grupo se puede dirigir. Osterwalder y Pigneur indican que los grupos de clientes pertenecen a segmentos diferentes cuando:

- Sus necesidades requieren y justifican una oferta diferente.
- Son necesarios diferentes canales de distribución para llegar a ellos.
- Necesitan un tipo de relación diferente.
- Su índice de rentabilidad es muy diferente.
- Están dispuestos a pagar por diferentes aspectos de la oferta.

Se deben agrupar a los clientes o usuarios con perfiles similares en segmentos definidos y, a continuación, investigarlos. Recopilar la máxima información posible sobre ellos: ¿dónde viven?, ¿dónde trabajan?, ¿qué costumbres tienen?, ¿cuáles son sus gustos?, ¿qué necesidades o problemas tienen?, etc. Algunos ejemplos clásicos de segmentos de clientes son:

- **Mercado de masas:** Se enfocan al público en general. Un ejemplo de este modelo de negocio es la electrónica de gran consumo.
- **Nicho de mercado:** Se centra en un segmento especializado con clientes muy específicos. Por ejemplo, el fabricante de repuestos para maquinaria agrícola.
- **Mercado segmentado:** Es el que abarca varios segmentos con pequeñas diferencias, por ejemplo, una misma entidad bancaria que atiende grandes cuentas y a usuarios modestos.
- **Mercado diversificado:** Con grupos con necesidades muy diferentes. Por ejemplo, Amazon que ofrece BBDD para desarrolladores y a la vez venta de productos frescos.

- **Mercados multilaterales:** Es decir, segmentos diferentes y complementarios del mercado como, por ejemplo, los usuarios de tarjetas de crédito y los comercios que las aceptan.

Truco:

No hay que obsesionarse por encontrar clientes y pensar que cuanto más mercado se abarque mejor. Hay autores que recomiendan poner foco e iniciar un negocio buscando nichos desatendidos. De esta forma, es posible centrarse en resolver una necesidad concreta de un grupo muy específico y que estará dispuesto a pagar por ello. Y, a partir de ahí, ir ampliando el negocio a otros segmentos.

Propuesta de Valor

¿Qué valor proporcionamos a nuestros clientes? ¿Qué problema de nuestros clientes ayudamos a solucionar? ¿Qué necesidades de los clientes satisfacemos? ¿Qué paquetes de productos o servicios ofrecemos a cada segmento de mercado?

En el capítulo 3 se insistió en la importancia de trabajar para conocer la visión del producto. Pero tener la visión clara no es suficiente. Es necesario definir correctamente la propuesta de valor o de no hacerlo, la probabilidad de fracaso se eleva de forma considerable.

La propuesta de valor, junto con los segmentos de mercado, son el núcleo del negocio.

La propuesta de valor es aquello que te hace diferente de la competencia y por lo que los clientes querrán pagar.

Se tendrá que definir el conjunto de productos y servicios que un segmento del mercado precisa. La propuesta puede ser nueva y original, ofreciendo un producto o servicio no existente hasta el momento en el mercado, o bien puede ofertar algo ya existente en el mercado, pero incorporando alguna característica que le haga diferente. El valor ofrecido puede ser cuantitativo, es decir, que lo podremos medir como el precio o el tiempo de respuesta, o cualitativo, como la experiencia de uso, un trato especial o el diseño.

Para crear valor, se pueden tener en cuenta estos elementos: la novedad, el precio, una mejora en el rendimiento, la personalización de un producto, proporcionar “un trabajo hecho”, el diseño diferente y especial, relacionar una marca con un estatus social, ayudar a los clientes a reducir sus costes o sus riesgos, dar acceso a un servicio a segmentos que anteriormente no podían disfrutar de él y, finalmente, mejorar la comodidad y utilidad de un servicio ya existente en el mercado.

Canales

¿Qué canales prefieren nuestros segmentos de mercado? ¿Cómo establecemos actualmente el contacto con los clientes? ¿Cómo se conjugan nuestros canales? ¿Cuáles tienen mejores resultados? ¿Cuáles son más rentables? ¿Cómo se integran en las actividades diarias de los clientes?

Una vez identificados nuestros clientes y usuarios, se debe poner en contacto con ellos. Este bloque del lienzo incluye el detalle de los canales que se van a utilizar para dar a conocer el producto a las personas u organizaciones interesadas en él y explicarles en qué consiste nuestra propuesta de valor. También se incluye la información necesaria para hacérsela llegar, vendérsela y ofrecer un servicio de postventa. En resumen, se necesita identificar los canales de comunicación, distribución y venta.

Todos los canales tienen cinco fases, aunque no siempre se siguen todas ellas. Estas fases son:

1. **Información:** Fase en la que se dan a conocer los productos y servicios de la empresa.
2. **Evaluación:** Se debe ayudar a los clientes a evaluar la propuesta de valor.
3. **Compra:** Se facilita a los clientes la compra de los productos y servicios.
4. **Entrega:** Se entrega a los clientes lo ofertado en la propuesta de valor.
5. **Posventa:** En ocasiones, es importante ofrecer un servicio de atención posventa.

Se tiene que estudiar y evaluar a través de qué canales los clientes quieren ser contactados, cuáles tienen mejores resultados, los más eficientes también desde el punto de vista económico y cómo integrarlos dentro del día a día de los clientes. El conjunto de canales de una empresa será su red de distribución.

Los canales pueden clasificarse a su vez en dos grupos: canales propios o canales de socios.

- Los **canales propios** proporcionan más beneficios, pero su coste es más elevado y la puesta en funcionamiento es más compleja. Este tipo de canal suele ser directo, es decir, sin intermediarios, como es el caso de un equipo comercial o las ventas por Internet. También pueden ser indirectos como es tener una tienda propia.
- Los **canales de socios** ofrecen menos beneficios, pero permiten aumentar el ámbito de actuación y aprovechar otros canales de distribución ya existentes. Estos canales son siempre canales indirectos, como es el caso de una tienda asociada o trabajar con mayoristas y minoristas.

Existe la posibilidad de utilizar y combinar diferentes canales (directos e indirectos, propios y de socios) y, tal y como comenta Osterwalder, “el truco consiste en encontrar el equilibrio adecuado entre los tipos de canales para integrarlos de forma que el cliente disfrute de una experiencia extraordinaria y los ingresos aumenten lo máximo posible”.

Relación con clientes

¿Qué tipo de relación esperan los diferentes segmentos de mercado? ¿Qué tipo de relaciones hemos establecido? ¿Cuál es su coste? ¿Cómo se integran en nuestro modelo de negocio?

Se entra ahora en un terreno muy delicado: se tiene que "captar" y "cuidar" a los clientes y usuarios. Se debe decidir cuántos recursos, tanto económicos como de tiempo, se quieren utilizar para mantener el contacto con ellos. Se quiere conservarlos como clientes, motivar que vuelvan a comprar los productos y, por supuesto, atraer nuevos clientes.

Las formas más comunes de relación con los clientes son:

- **Asistencia personal:** Relación directa y personal entre un cliente y un dependiente. Suele producirse en un punto de venta o a través de un teléfono de contacto, correo electrónico, chat, etc.
- **Asistencia personal exclusiva:** Similar a la atención anterior, pero aún más exclusiva para un cliente en particular. Este tipo de atención es habitual en productos de lujo o en grandes cuentas.
- **Autoservicio:** No hay relación directa con los clientes, pero la empresa sí pone los medios necesarios para que los clientes se puedan servir a sí mismos.
- **Automatizada:** Consiste en automatizar procesos simulando una relación personalizada. Un ejemplo de esta relación son las recomendaciones automáticas que ofrecen algunas páginas de venta por Internet.
- **Comunidades:** En ocasiones, algunas compañías forman comunidades para intercambiar información sobre sus clientes y fomentan la relación entre ellos para que se resuelvan los problemas cuando sea posible. Un ejemplo es Amazon cuando pone en contacto a unos usuarios con otros para resolver dudas.
- **Creación colectiva:** Consiste en implicar a los usuarios en los procesos del ciclo de negocio. Por ejemplo, incluyendo las opiniones sobre los productos que han comprado de forma que otros usuarios puedan verlas. Otro ejemplo son las páginas que permiten a los usuarios diseñar productos en línea y ganar comisión por la venta de los mismos.

Figura 16.2. Los nueve módulos.

Es en este apartado del lienzo es donde es posible enterarse si se hizo bien el trabajo, si el producto está gustando o no y qué se podría cambiar para mejorarlo.

Nota:

Desde luego, no es necesario mantener el mismo tipo de relación con todos nuestros segmentos de clientes. Lógicamente, si un servicio o producto es más caro, los clientes esperarán y podrán exigir una atención más cercana y de mayor calidad.

Fuentes de ingresos

¿Por qué valor están dispuestos a pagar los clientes? ¿Por qué pagan actualmente? ¿Cómo pagan actualmente? ¿Cómo les gustaría pagar? ¿Cuánto reportan las diferentes fuentes de ingresos al total de ingresos?

Se ha comentado en el apartado “Segmentos de mercado” que sin clientes no hay negocio, pero necesitamos negocios “vivos” y la vida la dan las fuentes de ingresos. Osterwalder y Pigneur comentan que las fuentes de ingresos se pueden generar de varias formas:

- **Venta de activos:** Esta fuente es la más conocida y es la venta de los derechos de propiedad sobre un producto físico. Un coche, un libro, un ordenador o un teléfono móvil son comprados y el cliente pasa a ser propietario de ese producto que luego podrá usar o revender.
- **Cuota por uso:** Basada en el pago por uso de un servicio, como, por ejemplo, una

entrada de cine, una noche de hotel o minutos consumidos de comunicación telefónica.

- **Cuota de suscripción:** Consiste en pagar una cantidad fija para poder disfrutar de un servicio durante un periodo de tiempo. Por ejemplo, una suscripción a una revista o la cuota de un gimnasio.
- **Préstamo/alquiler/leasing:** Consiste en la concesión temporal de un derecho exclusivo a cambio de un pago por un periodo de tiempo establecido. Supone una fuente de ingresos recurrente y un beneficio para el usuario, ya que no tiene que asumir el coste total del bien disfrutado.
- **Concesión de licencias:** Es muy habitual en las empresas de multimedia o tecnológicas cobrar por permitir el uso de un producto.
- **Gastos de corretaje:** También conocida como cobro de comisiones a intermediarios que hacen posible que la propuesta de valor llegue al usuario. Por ejemplo, las agencias de viajes que, por cada gestión que se realiza con ellas, reciben una cantidad.
- **Publicidad:** Es una fuente de ingresos cada vez más extendida y se aplica desde en los negocios por Internet hasta en la organización de eventos. Consiste en facilitar espacios o medios dentro de tu modelo de negocio para que otros se anuncien pagándote por ello.

Además de las diferentes formas de generar ingresos, existen dos mecanismos para fijar los precios: Fijo, con los precios basados en variables estáticas, y Dinámico, en el que los precios varían dependiendo del mercado.

Recursos clave

¿Qué recursos clave requieren las propuestas de valor, canales de distribución, relaciones con clientes y fuentes de ingresos?

Todo lo expuesto anteriormente está muy bien, pero ¿cómo se puede hacer realidad? Pues aquí es donde toca identificar los activos necesarios para que el negocio funcione, es decir, con qué medios voy a contar para crear y ofrecer mi propuesta de valor. También es necesario identificar con qué recursos se va a llegar al mercado, establecer las relaciones y, finalmente, recibir los ingresos.

Los recursos necesarios pueden ser clasificarlos en varias categorías:

- **Físicos:** Son los recursos materiales como vehículos, máquinas, locales, tiendas, redes de distribución, etc. Estos recursos favorecen una clara ventaja frente a la competencia.
- **Intelectuales:** Este tipo de recurso es cada vez más importante en un modelo de negocio con fundamento, ya que son “únicos”. Cuesta mucho esfuerzo conseguirlos, pero su valor es estratégico. Un ejemplo de este tipo de recursos intelectuales son las patentes, marcas o la cartera de clientes.

- **Humanos:** El trabajo humano es imprescindible en todas las empresas, pero, si además se está inmerso en un ámbito creativo o donde se requiera de un conocimiento experto, los recursos humanos son fundamentales.
- **Económicos:** A ninguna empresa le viene mal disponer de recursos económicos, pero, si no se dispone de ese capital, tal vez solicitar un crédito puede ayudar en un momento dado. Contar con una línea de crédito, préstamos a bajo interés, inversores o una opción de venta pueden ser recursos financieros clave.

Una empresa puede disponer de los recursos anteriores en propiedad, alquilarlos y adquirirlos a través de un socio clave.

Actividades clave

¿Qué actividades clave requieren las propuestas de valor, canales de distribución, relaciones con clientes y fuentes de ingresos?

En el puzle del modelo de negocio, se está frente a una de las piezas más complicadas de definir, ya que se trata de las actividades y procesos necesarios para que "funcione" la maquinaria del producto. En definitiva, las actividades y los recursos clave son las bases que sostienen el negocio. Para llegar aquí, ya se tiene definida la propuesta de valor, los canales de distribución y las relaciones con los clientes. Ahora toca definir las actividades necesarias para poder entregar la oferta o, dicho de otra forma, ¿qué es lo que se tiene que construir para que alguien pague por ello? La siguiente es una clasificación sencilla y de gran ayuda a la hora de establecer las actividades clave:

- **Actividades de producción:** En empresas de fabricación y reparación, los procesos de diseño, fabricación y entrega son actividades clave. Y más, si hay que realizarlos a gran escala.
- **Actividades de solución de problemas:** Si la oferta es aportar creatividad y conocimiento para solucionar problemas a los clientes, obviamente la actividad será solucionar esos problemas. Hospitales, consultoras y agencias son claros ejemplos de ello. También se tendrá la actividad de la formación continua para mantener el conocimiento al día.
- **Actividades de plataformas y redes:** En el caso de que el negocio tenga como recurso clave una plataforma, su constante actualización y mantenimiento son fundamentales.

Osterwalder lo ilustra con algunos ejemplos claros: "La actividad clave del fabricante de software Microsoft es el desarrollo de software, mientras que la del fabricante de ordenadores Dell es la gestión de la cadena de suministros. A su vez, una de las actividades clave de la consultora McKinsey es la resolución de problemas".

Nota:

Es fundamental reconocer cuáles son las actividades clave para destacar y lucirse en ellas.

Asociaciones clave

¿Quiénes son los socios clave? ¿Quiénes son los proveedores clave? ¿Qué recursos clave se adquieren a nuestros socios? ¿Qué actividades clave realizan los socios?

No se tiene por qué navegar solos por el océano del negocio. Por ese motivo, son cada vez más frecuentes las alianzas entre empresas. Las asociaciones pueden ayudar a reducir riesgos en entornos con mucha incertidumbre, obtener recursos sin necesidad de partir de cero en su construcción y, hasta en ocasiones, ayudar a mejorar el modelo de negocio. Las asociaciones se pueden agrupar en cuatro tipos:

1. Empresas no competidoras pueden realizar **alianzas estratégicas** y obtener beneficio mutuo.
2. **Coopetición** o coopetencia es la asociación temporal entre empresas que SÍ son competidoras.
3. **Join Ventures** o empresas conjuntas que se unen para crear nuevos negocios.
4. Las alianzas **cliente-proveedor** para así garantizar la fiabilidad de los suministros.

Estructura de costes

¿Cuáles son los costes más importantes inherentes al modelo de negocio? ¿Cuáles son los recursos clave más caros? ¿Cuáles son las actividades clave más caras?

Ya solo queda hablar de todos los costes necesarios para poner en marcha el modelo. La creación, entrega y mantenimiento del producto tienen un precio. Aunque minimizar los costes es un objetivo en cualquier negocio, está claro que, dependiendo del modelo de negocio, la estructura de costes será más o menos importante. Los expertos hablan de dos tipos de estructura de costes:

- **Según costes:** Basada en recortar gastos de donde sea posible con propuestas de valor de bajo coste. Un claro ejemplo de esta estructura son los hostales de jóvenes o las compañías aéreas *low cost*.
- **Según valor:** Aplicada en empresas donde el precio no debe condicionar el servicio ofrecido. Es el caso de los hoteles de lujo o los vuelos en categoría *business*.

En la mayoría de los casos, la estructura de costes es una combinación de las dos

anteriores.

Business Model Canvas o lienzo del modelo de negocio

Conocer los nueve módulos no es suficiente. Es necesario colocarlos en un lienzo de forma ordenada. Esto es lo que se conoce como "*Business Model Canvas*" y es la herramienta que ayudará a describir, analizar, diseñar e inventar nuevos modelos de negocio. La propuesta es trabajar con el lienzo como si fuéramos un pintor y esbozar el modelo entre todos los implicados. Para ello, lo primero que se ha de hacer es imprimir una plantilla del lienzo en tamaño muy grande y colgarla en una pared para que se pueda escribir en ella, pegar *post-it*, fotos, notas o lo que sea interesante en cada apartado. Los interesados observarán el lienzo como una obra de arte en creación, de forma que surja el debate, el análisis y la creatividad.

Para que el lienzo sea valioso, se necesitan conectar los distintos elementos y hacer que fluya "el valor del negocio" entre los diferentes bloques.

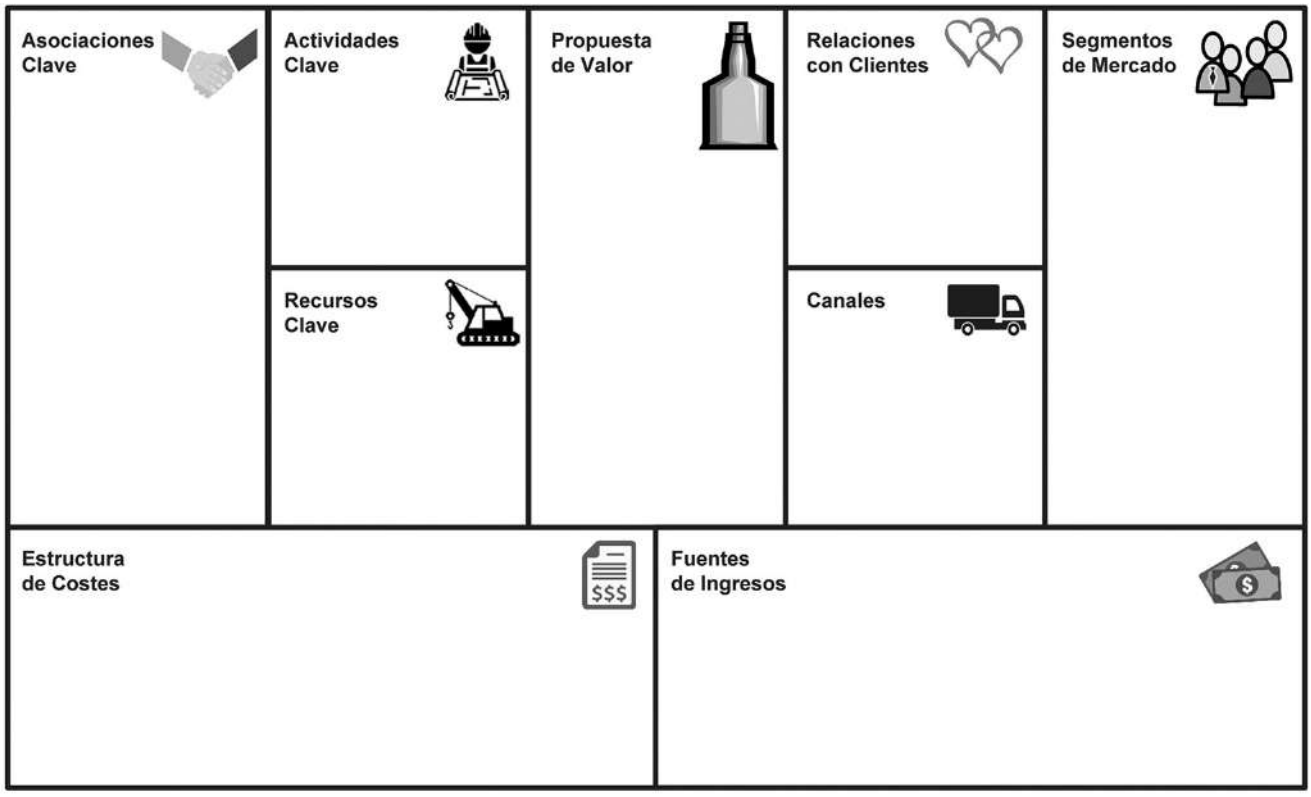


Figura 16.3. Plantilla del lienzo del modelo de negocio.

Existen numerosos patrones de modelos de negocio basados en el conocimiento de grandes expertos empresariales. También hay múltiples técnicas para su diseño que pueden

ser de gran ayuda en la fase de ideación de un producto, para la creación de prototipos o en la narración de historias, pero estos temas salen fuera del propósito de este manual. Si se quisiera profundizar en alguna de estas disciplinas, se recomienda en la bibliografía algunas lecturas muy interesantes y especializadas en estas materias.

En resumen

Las *startups* o empresas emergentes son un desafío para las empresas que ya están en el mercado que, o se ponen al día, o se quedarán obsoletas más pronto que tarde. Para mantenerse en el terreno de juego del mercado, es fundamental que se mantenga actualizado el modelo de negocio. Este capítulo pretende ser una introducción para ayudar a descubrir las áreas que lo componen, la relación entre ellas y cómo trabajar en equipo con todos los interesados para mantener vivo ese modelo.

El *Business Model Canvas* o lienzo del modelo de negocio nos ayudará a conseguir plasmar de forma sencilla, que nunca simplista, nuestro modelo, con sus puntos fuertes, riesgos, amenazas y apuestas.

---

68 *Business Model Canvas*, Alexander Osterwalder & Yves Pigneur. Published by John Wiley & Sons, Inc. Hoboen, New Yersey. 2010.

17 Especificaciones ejecutables

En este capítulo aprenderá:

- Qué son las especificaciones ejecutables.
- Cómo aplicar las especificaciones ejecutables en *Scrum*.
- Riesgos y ventajas de las especificaciones ejecutables.

No hay especificaciones buenas o especificaciones malas. Existen especificaciones y otro conjunto de cosas que se pueden denominar de muchas maneras, menos especificaciones. Para poder ostentar dignamente el título de especificación, solo hay que hacer una cosa: ser fiel a su nombre. Especificar significa concretar, dar detalle. No es suficiente hablar de manera generalista, vaga o ambigua de la funcionalidad que se quiere construir o alcanzar. Lamentablemente, muchas veces llegar a ese nivel de detalle es complicado. No tanto porque se desconozca lo que se quiere hacer, sino porque no se encuentra la manera adecuada de poder reflejarlo en algún artefacto. Es entonces cuando las especificaciones ejecutables aparecen en escena. Son una solución para reflejar el nivel de detalle necesario en la implementación de un proyecto o producto.

¿Qué son las especificaciones ejecutables?

Antes de entrar en materia, revisemos algunos conceptos del *Backlog* y sus elementos, que se han visto en anteriores secciones del libro. Este repaso será de utilidad en este capítulo, ya que los elementos que se revisarán son la base de las especificaciones en *Scrum*.

Es importante recordar que el *Backlog* del producto o proyecto recoge la lista de los requisitos funcionales y no funcionales que se tienen que llevar acabo. Para que nuestro *Backlog* se encuentre en buena forma, necesita cumplir los preceptos DEEP. Estos preceptos se desgranan en que éste se encuentre detallado suficientemente según vamos avanzando por él, que esté estimado, que sea emergente y esté priorizado. Nuestro *Backlog* contendrá diversos elementos o ítems que podremos organizar en temas o temáticas. En función de su tamaño, los consideraremos épicas o historias de usuario. Estos elementos que conforman el *Backlog*, para estar correctamente concebidos, deberán seguir el principio INVEST. Esto implica que cada elemento debe de ser independiente, negociable, que aporte valor al conjunto del proyecto o producto, que sea estimable, con un tamaño (**size**) adecuado a su posición en el *Backlog* y, finalmente, que se pueda probar (**test**). Estos elementos del *Backlog* tienen que estar correctamente definidos y, para esto, seguiremos la recomendación CCC. En esta recomendación, se sugiere que cada elemento se debería poder concretar de tal manera que entrara en una tarjeta de resumen. Que su definición fuera el resultado de las conversaciones tenidas dentro del equipo. Y, por último, que éste necesitará de una confirmación que indique claramente que ese elemento se ha completado.

Si cumplimos todas estas recomendaciones, ¿tendremos unas especificaciones para poder trabajar de forma correcta? Desafortunadamente, la respuesta es no. Existen una serie de problemas que suelen ocurrir y nos muestran la necesidad de complementar estos pasos, de por sí necesarios, con algo más.

Estos posibles problemas son:

- **Especificaciones generalistas:** Las especificaciones no bajan al suficiente nivel de detalle deseado y solo dan un esbozo de lo que se está buscando. Esto genera bloqueos en el equipo por dudas u otros problemas de entre los listados a continuación.
- **Malentendidos en la interpretación:** Debido a la falta de detalle en las especificaciones, los actores involucrados en el proceso de creación del producto o proyecto pueden interpretar de manera distinta los mensajes y ejecutar la especificación de manera diferente.
- **Especificaciones incompletas:** Aun siendo especificaciones detalladas, pueden no estar completas, lo cual genera igualmente bloqueos en el equipo al no poder finalizar de forma adecuada la implementación.
- **Falta de un lenguaje común:** Así como las especificaciones generalistas pueden llevar a malentendidos, la falta de un lenguaje común puede llevar a denominar un mismo concepto de manera distinta, generando igualmente bloqueos de comunicación. También puede dar lugar a problemas de interpretación.
- **Falta de trazabilidad y dificultad de mantenimiento:** El no conocer la fuente de los requisitos, su estado continuo y adaptación a las necesidades del proyecto complica su manejo y gestión durante su ciclo de vida.
- **Indefinición a la hora de saber si se han completado:** Las especificaciones están para implementarse, por lo que es vital saber si se han completado. En muchos casos, no existe un criterio claro que ayude a validar cuándo se ha terminado una funcionalidad y está lista para ser entregada.

¿Cuál sería la solución para completar las bases de la especificación de requisitos en *Scrum* para evitar estos problemas?

Para evitar unas especificaciones pobres, malentendidos o partes incompletas, la solución consiste en pasar de especificaciones abstractas, *ad hoc*, anárquicas y aburridas a una forma más humana y sistemática de materializar la definición de requisitos. Una forma que las acerque a la realidad y haga tangible el proyecto o producto que se está implementando, que ayude a visualizalo a todas las personas involucradas en el proyecto en curso. Esta forma implica definir lo que se busca según ejemplos concretos y reales. Sería como convertir las especificaciones en algo parecido al manual de instrucciones del montaje de una maqueta. En este manual, se detallaría paso a paso de la forma más explícita y concreta posible, siguiendo ejemplos de cómo colocar las piezas, el proceso que se debe seguir para llegar a buen puerto. En resumen, crear especificaciones basadas en ejemplos.

Para evitar la falta de un lenguaje común, la solución es aplicar la hipótesis de Sapir-Whorf en su principio de relatividad lingüística. La hipótesis de Sapir-Whorf establece que existe una cierta relación entre las categorías gramaticales del lenguaje que una persona habla y la forma en que la persona entiende y conceptualiza el mundo. Con lo que, estableciendo un lenguaje común que haga una relación directa a cómo se están entendiendo las cosas, podremos mejorar el entendimiento en el proyecto. En otras palabras, que los ejemplos antes

citados compartan expresiones y palabras entendidas igualmente en su contexto por todo el equipo. La solución es crear un nuevo lenguaje común y estructurado, sin ambigüedades, que manejen todas las personas del proyecto y que sirva para documentar este.

Para evitar los problemas de trazabilidad y mantenimiento, se debe reducir el salto entre las personas que saben lo que se quiere hacer y las que tienen que hacerlo. Así pues, se debe crear un artefacto común de definición que compartan todas las personas involucradas en el equipo para que no existan diferencias de criterio entre ambos mundos.

Por último, para saber cuándo se han completado los elementos de las especificaciones, debería existir un mecanismo fiable que reprodujera unas condiciones concretas de finalización y fuera capaz de discernir si algo está completado o no de manera repetible, autónoma y unívocamente. Una verificación automática de las DoD (**Definition of Done**) de la especificación.

Sintetizando, para evitar los problemas comunes a la hora de definir el contenido del *Backlog*, deberíamos usar ejemplos, bajo un lenguaje común, como elementos compartidos por todo el equipo y sobre los que se pudieran ejecutar automatismos en su confirmación de completado. A esta forma de definir los requisitos se le denomina especificaciones ejecutables y a la forma de incluir esta práctica en los ciclos de *Scrum* para validar su implementación se le denomina BDD (*Behavior Driven Development*) o desarrollo dirigido o guiado por comportamiento. Esta forma de desarrollo fue definida por Dan North[69] en respuesta a todos los problemas que identificó a la hora de intentar transmitir las bases del desarrollo dirigido por pruebas (TDD).

¿Por dónde empiezo?

Aplicar BDD en un equipo implica extender el concepto del *Backlog* que se esté manejando y, en concreto, extender la forma en la que las historias de usuario se gestionan. Implica pasar del criterio de las 3 C al criterio de las 4 C. La última C que se está añadiendo a nuestras historias de usuario es la de codificar o programar. Si se quiere que nuestras especificaciones sean ejecutables, se tendrán que convertir en código ejecutable. Pero, para saber qué es lo que hay que programar, antes se tiene que haber sido capaz de expresar las historias de usuario en un formato similar al de los lenguajes de programación. Para esto, hay que fijarse en la anterior C, la que hace referencia a los criterios de aceptación. Hay que encontrar la forma de redactar los criterios de aceptación en una forma estructurada y predecible que facilite el trabajo de codificación. Con este objetivo aparece en acción el concepto de los escenarios.

Los escenarios, el núcleo de las funcionalidades

Los escenarios son parecidos a las escenas en una película. De forma individual dan una visión de detalle de algo que está ocurriendo, algo muy concreto. Cuando se suman todas las escenas (o escenarios en nuestro caso), se obtiene el guion de lo que ocurre en la película. Este principio se puede aplicar a la forma de describir las historias de usuario o características (*features*) del producto. Se puede descomponer cada una de ellas en estas escenas o escenarios parciales. Y su suma dará como resultado una muy detallada descripción de la historia de usuario completa. Como se comentaba en el párrafo anterior, esta definición basada en escenas sería la última de las C, la de los criterios de aceptación o *Definition of Done*. Una historia de usuario estará completa cuando todos sus escenarios puedan ser confirmados. Para construir los escenarios de forma programática, un primer paso es definir una forma estructurada y común a la hora de construirlos. Una buena aproximación es estructurarlos según las siguientes secciones:

- **Título del escenario:** Descripción resumida que identifica al escenario que se describirá.
- **Introducción o prerrequisitos:** En esta parte de la descripción del escenario, se describe la condición de contexto particular en la que va a ocurrir la escena. Se detalla, para todos los elementos implicados en el escenario, la situación en la que se encuentran al comienzo de la escena.
- **Cuerpo o procedimiento:** En esta parte de la descripción del escenario es donde se relatan las acciones que tienen lugar en la escena. Es donde se desarrolla la actividad principal del escenario.
- **Desenlace o validación:** En esta parte de la descripción del escenario es donde se explica el resultado de las acciones que han ocurrido en la escena y se define su validación.

Para entender mejor esta estructura, veamos un ejemplo de una funcionalidad y su descomposición en escenarios.

Podemos imaginar un producto en el que los usuarios acceden con un identificador y una clave. Se tendría entonces una historia de usuario o *feature* de identificación de usuario. Para poder describir la funcionalidad y aplicar BDD, hay que ser capaces de descomponer esta *feature* en escenarios o escenas. Las escenas, como si de una película se tratara, deberían describir posibles situaciones en las que se encontraría un usuario a la hora de identificarse. Una posible lista sería:

- El usuario introduce mal su identificador.
- El usuario introduce mal su clave.
- El usuario deja alguno de los campos vacío.

- El usuario falla varias veces al introducir su identificador y contraseña.

Aunque esta lista podría extenderse mucho más, se puede observar cómo, con cada uno de los escenarios, se va enriqueciendo más el abanico de posibilidades que deben tenerse en cuenta a la hora de implementar la funcionalidad de identificación de usuarios. Tómese como referencia una de ellas para analizar su estructura, por ejemplo, “El usuario introduce mal su clave”.

Siguiendo la referencia que se ha expuesto anteriormente, se tendría un título claramente identificado: “El usuario introduce mal su clave”.

Como inicio o sumario, se podría explicar que el sistema de identificación de usuarios está iniciado y operativo.

Como introducción, se podría plantear en qué situación tendría que estar el usuario para poder describir esta escena. Rápidamente se podría concluir que se necesitará que el usuario esté dado de alta en el sistema.

En la sección de cuerpo o procedimiento, se puede describir que el usuario introduce su nombre de usuario correctamente y su clave de manera incorrecta. Finalmente, en el desenlace o validación, se puede concluir que, dado que la clave es incorrecta, se le avisará al usuario que ha introducido incorrectamente alguno de los valores.

Si se pone en forma de lista esta estructura, se llegaría a algo similar a esto:

- El usuario introduce correctamente su identificador y clave.
- El usuario introduce mal su clave.
- El usuario está dado de alta.
- El usuario introduce su identificador correctamente y su clave de manera incorrecta.
- El sistema avisará al usuario de que no ha introducido correctamente alguno de los valores.

El proceso de creación de escenarios es un proceso iterativo. Cuando se va intentando resolver o detallar cuestiones de los escenarios individuales, aparecen nuevas pistas que pueden llevar a nuevos escenarios. En el caso que se analizaba previamente, a la hora de decidir sobre los prerrequisitos de un escenario, se planteaba en qué estado debería estar el usuario antes de realizar el procedimiento de identificación. La conclusión fue que debería estar previamente dado de alta en el sistema. ¿Qué hubiera ocurrido si el usuario no hubiera estado dado de alta? Pues que, aunque el procedimiento hubiera sido el mismo, el resultado o desenlace hubiera variado, lo cual indica que se tendrá otro escenario. Así que el escenario original tendrá que desdoblarse a su vez en dos nuevos escenarios.

Al igual que ocurre cuando se itera sobre escenarios pudiendo aparecer otros nuevos escenarios, también puede suceder lo contrario. Puede ocurrir que varios escenarios que, a priori, pudieran parecer diferentes, acaben formando parte de un mismo escenario. Esto ocurre cuando prerrequisitos que pueden parecer distintos lleguen a idéntico desenlace con

un mismo procedimiento. En ese caso, se puede crear lo que se conoce como “juego de datos” para un escenario. Un juego de datos, o *dataset*, es una colección de datos o prerrequisitos que se pueden utilizar sobre un mismo escenario obteniendo resultados similares. Recupérese el ejemplo anterior para visualizar la aplicación de un *dataset*. En el escenario de identificación de usuarios, la acción “introduciendo incorrectamente la clave” admite diversas situaciones. Se podría decir que la clave podría ser totalmente distinta, podría ser la clave correcta difiriendo en un carácter al final y la clave correcta difiriendo al inicio. En todos los casos, obtendríamos el mismo resultado y esos valores de claves erróneas serían nuestro *dataset*.

¿Cómo tienen que ser los escenarios ideales? Como buen elemento dentro de las metodologías ágiles, también se tendrá un acrónimo que ayudará a recordar cómo deberíamos detallar estos escenarios. En este caso, sería FIRST (*Fast* - *Isolated* - *Repeatable* - *Self validating* - *Timely*). Explicaremos con detalle lo que implica este acrónimo cuando veamos más sobre la implementación y codificación de los escenarios. También aparece otro elemento recurrente de las metodologías ágiles en la creación de escenarios: la prioridad. Estos escenarios, en relación a una historia de usuario, estarán ordenados por prioridad, lo que implica que los primeros escenarios que enunciemos serán más prioritarios que los últimos.

En resumen, detallar las especificaciones de nuestro producto implica iterar para descomponer cada una de las historias de usuarios en una agregación de escenarios o escenas. Con los escenarios ya se tiene uno de los primeros elementos que se estaban buscando: la definición basada en ejemplos. A continuación, se verá cómo se puede aplicar un lenguaje común sobre esta definición.

Gherkin, un lenguaje fresco

Al hablar de escenarios, hemos descrito uno de los aspectos básicos de las especificaciones ejecutables: crear ejemplos detallados que concreten lo que queremos especificar. Con la estructura propuesta se tiene un marco de referencia para organizar la información, pero es necesario concretar la forma en la que se expresará. Como ya se vio, manejar un lenguaje de dominio o DSL es garantía para evitar malentendidos en las especificaciones.

Gherkin (Pepinillo) es un lenguaje creado específicamente para BDD. Su nombre procede de una de las herramientas que popularizó el concepto de BDD, *Cucumber* (Pepino). *Gherkin* es un lenguaje específico de dominio con las siguientes características:

- Legible por cualquier persona del equipo.
- Solo contiene texto, sin necesidad de herramientas específicas.

- Útil para documentar el producto y automatizar pruebas.

Este lenguaje tiene una sintaxis orientada a línea. Esto implica que la redacción de los escenarios se realiza en líneas separadas. Cada una de estas líneas se denomina paso o *step*. Cada una de las secciones que se describieron anteriormente en la estructura de un escenario podrá estar compuesta de uno o más pasos. En consecuencia, cada parte del escenario tendrá una o varias líneas o pasos. Para mostrar visualmente la separación de los bloques del escenario, se usa el sangrado. En *Gherkin* se pueden describir los escenarios con cualquier idioma.

En la figura 17.1 se puede observar la estructura de una historia de usuario o *feature* tal y como se describía previamente, detallando los pasos o *steps* de cada una de las secciones.

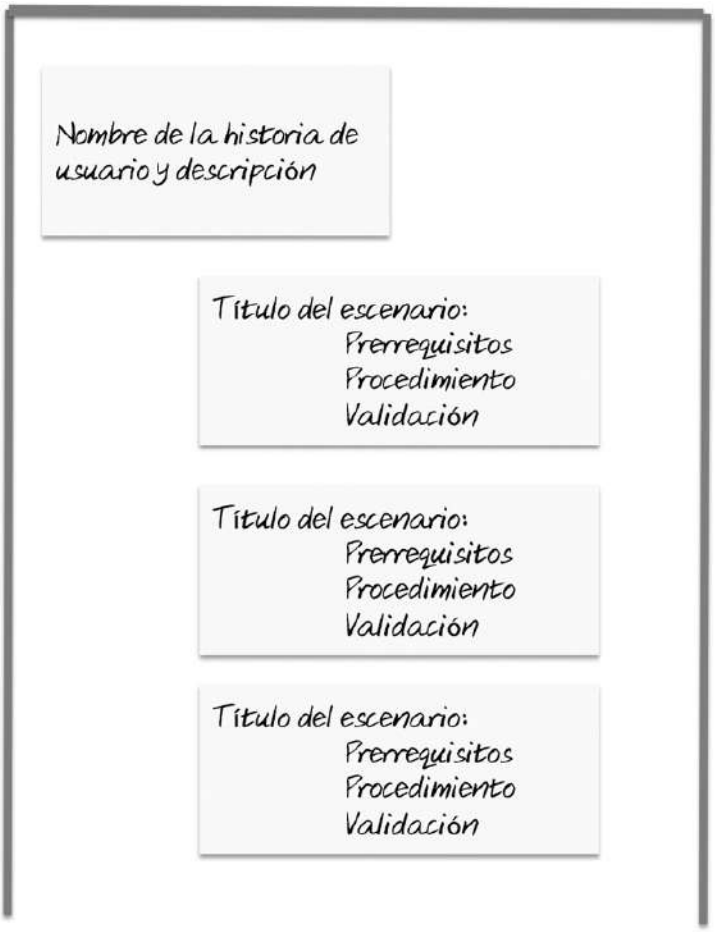


Figura 17.1. Estructura de una feature en BDD.

Notación en Gherkin

Aplicar *Gherkin* a una *feature* implica seguir unos elementos básicos de notación. Lo primero que se encuentra en una *feature* en *Gherkin* es el nombre de la *feature*. Después, se encontrará el valor desde el punto de vista de negocio de la historia de usuario con la

maquetación correspondiente. En otras palabras, se encontrará la definición tal y como se vio en capítulos anteriores. *Como usuario [rol], me gustaría [funcionalidad], de tal manera que [beneficio]*. A continuación, se encuentra el primer encabezado del primer escenario, con el título, también denominado *outline*, del escenario. La siguiente imagen da un ejemplo de una apertura de la *feature* en *Gherkin*.

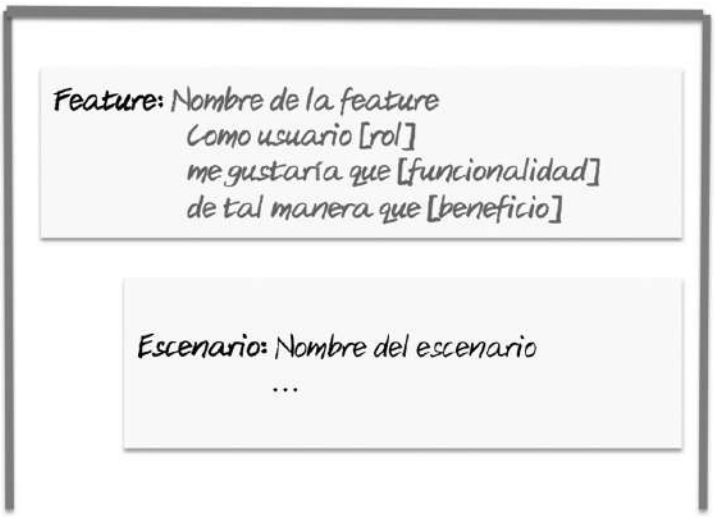


Figura 17.2. Encabezado de feature en Gherkin.

Lo siguiente que se encuentra en una *feature* son los prerrequisitos del escenario. A esta sección en *Gherkin* también se la conoce como los *Givens*. Se le denomina de esta manera porque el primer paso o *step* de esta sección se inicia en inglés con la palabra "*Given*". En castellano, iniciaríamos el paso con un "dado que". Si se retoma el ejemplo de la identificación de los usuarios y el escenario en concreto que manejábamos, la redacción del prerrequisito quedaría como:

Dentro de cada sección, puede haber tantos pasos o *steps* como sean necesarios, aunque estos ya no suelen iniciarse repitiendo la palabra *Given*: en su lugar, se suele usar "y" () cuando se quiere añadir algo o "pero" () cuando se quiere incluir una excepción. Así pues, se pueden ampliar los prerrequisitos de la siguiente forma:

La siguiente sección que se encuentra es la del procedimiento. Esta sección se denomina el "*When*" o "Cuando" en castellano. En esta sección, tal y como se comentaba con anterioridad, se describe la acción que tiene lugar en la escena. A diferencia de en otras secciones, aquí se recomienda que exista un único paso o *step* que defina la acción. Retomando el ejemplo se tendría un procedimiento de la siguiente forma:

Finalmente, se tendrá la sección del desenlace o validación, también conocida como los "*Thens*" o "entonces". En esta sección, se observan los resultados del escenario y se pueden validar para certificar que son correctos. Al igual en los prerrequisitos, pueden existir varios pasos o *steps* pero solo el primero llevará un inicio característico, mientras que los otros *steps* serán o , según sea el caso.

En resumen, juntándolo todo tendríamos un escenario definido de la siguiente manera en *Gherkin*:

Usando los datos

Continuando con el ejemplo, se puede ver cómo se utilizan los datos en *Gherkin*. Ya se vio en el ejemplo inicial cómo se podían agrupar varios escenarios en uno solo: creando un *dataset*. Se trata de un conjunto de datos aplicados al escenario. En este conjunto de datos, tendríamos varias claves erróneas para aplicar al escenario. Con un *dataset* tendríamos algo similar a:

Modos imperativo y declarativo

Una vez está clara la estructura de los escenarios y la notación que se utiliza para narrar los escenarios en un DSL común, queda definir el estilo que se quiere usar. Existen dos estilos de narración de los escenarios: el modo imperativo y el modo declarativo.

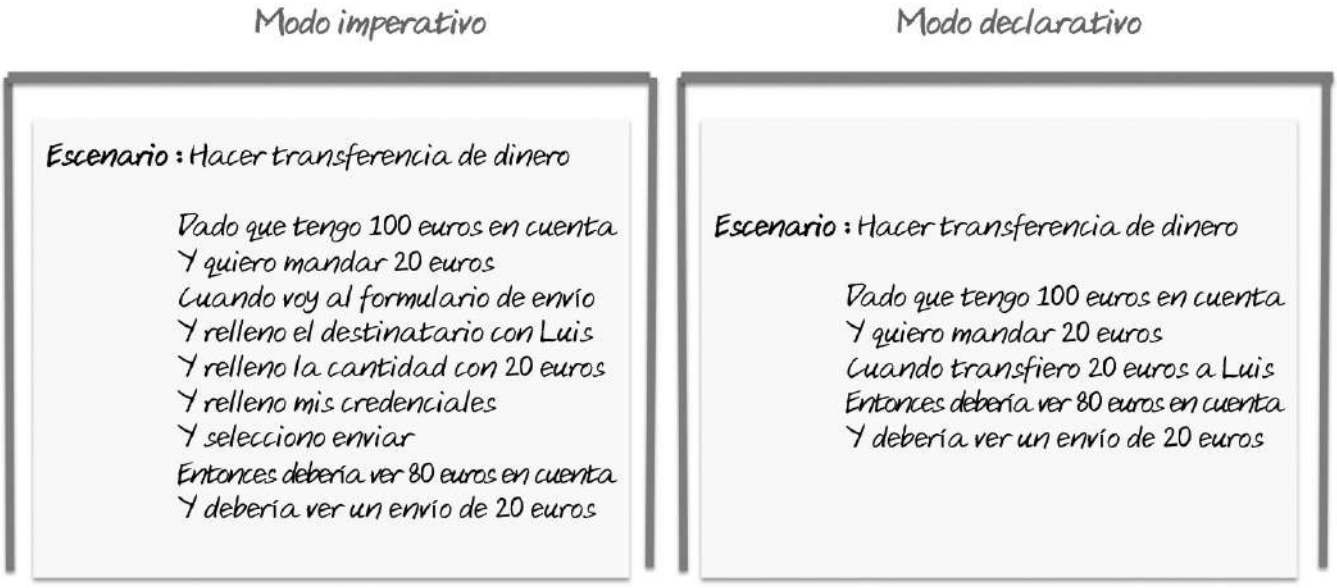


Figura 17.3. Ejemplos de estilo de Gherkin.

En el modo imperativo, los pasos o *steps* se relacionan con la actividad concreta que hace el usuario. Esta actividad está muy relacionada con la forma en la que interactúa con el sistema del que se está definiendo la funcionalidad. Cada acción se describe con un paso o *step* y estos tienen un grado de detalle muy elevado. Esto permite crear un DSL de muy bajo nivel y que, en muchos escenarios, estos pasos o *steps* se reutilicen para describir las mismas acciones.

Por otro lado, existe el modo declarativo. Este modo se centra más en el valor para el usuario, en vez de en las acciones que este realiza, y describe cada uno de los *steps* o pasos desde un punto de vista más amplio.

¿Cuál de estas aproximaciones es mejor? No hay una respuesta fácil. Depende del contexto. El lenguaje imperativo da mucha más información sobre la interfaz de usuario, además permite crear elementos reutilizables en los que más gente puede participar en la construcción de los escenarios simplemente reaprovechando estos elementos del lenguaje. Sin embargo, el lenguaje imperativo hace escenarios mucho más largos perdiéndose demasiado en los detalles y perdiendo información sobre el marco general. El lenguaje declarativo tiene la ventaja de que es más fácil de mantener y está más alineado con una estrategia de criterios de aceptación basados en el dominio que se está tratando.

En resumen, con *Gherkin* ya se tienen dos de los elementos que se estaban buscando para mejorar las especificaciones: un lenguaje y un artefacto comunes en el equipo. A continuación, se verá cómo se puede convertir este lenguaje en un elemento ejecutable.

Volviendo a las 4 C

Cuando definíamos las historias de usuario, comentábamos que para ser de calidad deberían cumplir la regla de las 3 C. Con el objetivo de tener una validación automática de requisitos, la regla de las 3 C se convierte en las 4 C, tal y como se ha comentado previamente. La cuarta C significará codificar. “Codificar” quiere decir convertir los escenarios definidos en *Gherkin* anteriormente en código que se pueda ejecutar.

Gracias a la estructuración y normalización del lenguaje que se ha hecho al aplicar *Gherkin*, el paso a codificación es mucho más sencillo. Para esto, existen *frameworks* de automatización en los que se utilizan los ficheros de definición en *Gherkin*. En estas herramientas se trabaja en programar acciones para cada uno de los *steps* definidos. De esta manera, se obtienen piezas de código con las acciones que representa cada *step*. Así, cuando se ejecuta el escenario en cuestión, se realizarán cada una de las acciones que se han previsto: los prerrequisitos para configurar el sistema y llevarlo al punto deseado en el escenario, el procedimiento para accionarlo y los resultados para tomar la salida del sistema y verificar su resultado. De esta manera, si ponemos estas especificaciones ejecutables frente a un sistema real, después de su ejecución, sabremos cuáles están completadas y cuáles no se comportan de la manera definida o están pendientes por completar.

Tener esta capacidad de ejecución es extremadamente útil, ya que sirve para introducirla dentro de los ciclos de implementación, ayudando a resolver muchos de los problemas definidos al inicio de la sección.

En resumen, utilizando *frameworks* específicos de desarrollo se puede asociar a las definiciones en *Gherkin* piezas de código ejecutable. Así, se conseguiría el último punto que

se estaba buscando para mejorar las especificaciones: la capacidad automática de saber si algo está terminado o no.

BDD en acción

BDD no es una herramienta, es una forma de trabajar. ¿Cómo se aplica BDD dentro de *Scrum*? Muy sencillo: los elementos del *Backlog* estarán definidos en función a escenarios, tal y como se ha descrito en este capítulo. Siguiendo las recomendaciones de *Scrum*, los elementos que estén en la parte superior del *Backlog* estarán mucho más trabajados y tendrán completos estos escenarios. En este caso, se habrán trabajado y definido con notación *Gherkin.* Además, irán acompañados de una codificación pertinente de tal manera que podrán ser ejecutados.

Una vez se decida que ese elemento del *Backlog* es parte del *Sprint* y este comience, el desarrollador que quiera implementar la historia de usuario tomará la especificación y la ejecutará. Inicialmente fallará, es lo que normalmente se puede esperar. A partir de ese momento, el desarrollador trabajará en ciclos añadiendo la funcionalidad hasta que la ejecución de los escenarios de la historia de usuario se verifique de manera satisfactoria. En ese momento, el desarrollador podrá dar por concluida la historia de usuario. Esta forma de trabajar reducirá sus posibles dudas y ambigüedades en la interpretación de la historia de usuario. Le ayudará a saber con claridad cuándo ha concluido su trabajo y, además, velará por la calidad del producto que se está desarrollando.

La siguiente imagen resume el ciclo BDD aplicado a los ciclos de desarrollo en *Scrum*.

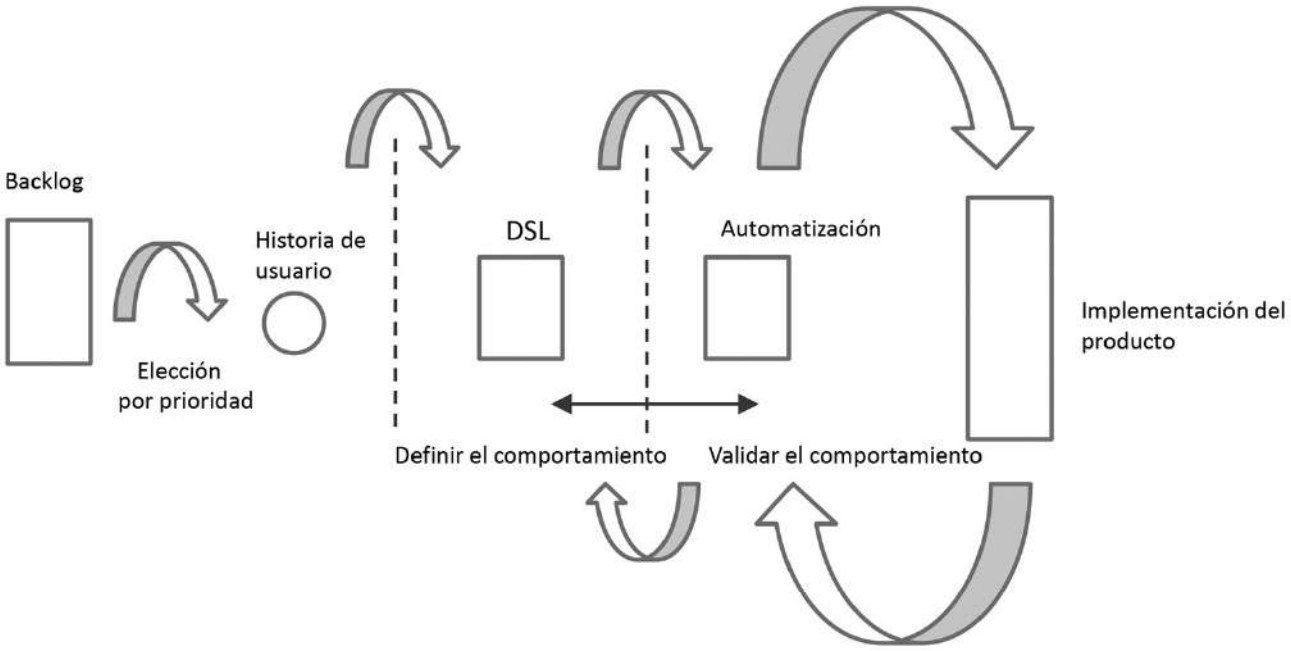


Figura 17.4. Ciclo BDD.

Ahora que ya se conoce como se aplica BDD, se puede retomar la guía para definir buenos escenarios y su principio FIRST (*Fast* - *Isolated* - *Repeatable* - *Self Validating* - *Timely*). Rápidos (*Fast*) porque, para utilizarlos como herramienta de desarrollo con validación, deben serlo para no ralentizar los ciclos de desarrollo. Aislados (*Isolated*) para que la ejecución de cada uno de ellos no influya en los otros. Repetibles (*Repeateable*) para obtener los mismos resultados cada vez que se ejecuten. Autoevaluables (*Self Validating*) para que pueda el propio escenario informar si el resultado de la implementación es o no correcto. Y, por último, Oportuno (*Timely*) para que se defina y construya en el momento necesario.

Recomendaciones

BDD es un arma de doble filo. Manejada de manera correcta puede atacar y resolver muchos de los problemas que existen hoy en día con las especificaciones, pero también tiene un lado oscuro. Uno de los mayores inconvenientes es que, independientemente de que se utilice en formato declarativo o imperativo, se generan unas especificaciones bastante largas: si se imaginan unas especificaciones en las que se tienen 10 historias de usuario, si se tienen unos 10 escenarios por cada historia de usuario y 10 pasos por cada escenario, se tendrían 1.000 líneas de especificaciones a las que se tendría que prestar atención y mantener actualizadas correctamente. Esto es un problema para su mantenimiento, pero también para su rendimiento. Si se manejan especificaciones muy complejas, el tiempo de ejecución puede incrementarse demasiado. En este punto, el utilizar la especificación ejecutable como una herramienta rápida de construcción quedaría descartado. La especificación quedaría únicamente como una validación puntual de los requisitos.

Otro de los riesgos que se sufre en este tipo de planteamientos es el riego del “corta-pega”. Se abusa mucho de esta práctica y hace que, con mucha frecuencia, los errores se extiendan por toda la especificación.

Usar BDD como herramienta para pruebas también puede llevar a malas prácticas. Forzar su uso para *testing* hace que el lenguaje con el que se representan los casos de uso sea retorcido y se lleve al límite, restando frescura a la redacción de la especificación.

¿Cuáles serían entonces las buenas prácticas que hay que seguir? La primera y fundamental es la de escribir las especificaciones antes de la implementación. Parece una obviedad, pero en muchos equipos se inicia el desarrollo con una simple conversación o correo y las especificaciones se definen posteriormente. Se requiere un esfuerzo extra, pero mantener las especificaciones actualizadas durante la creación de un producto o en un proyecto es clave para evitar problemas. Mucho más en el caso de BDD, dado que, de no

tener la definición actualizada, aparecerán errores en la validación de las historias de usuario y, por lo tanto, perderá valor el trabajo realizado.

Otra recomendación es intentar reducir la complejidad y longitud de cada escenario. Un indicador de escenarios largos es que las historias de usuario no están suficientemente desgranadas y merecen un ejercicio de desglose.

Por último, es muy importante que las especificaciones de BDD no sean un artefacto de unos pocos, sino que se conviertan en un artefacto de todo el equipo, que sea ese elemento vertebrador que una las conversaciones y debates de todo el equipo.

---

69 Introducing BDD by Dan North. https://dannorth.net/introducing-bdd/.

Glosario

Este anexo contiene los términos más importantes mencionados a lo largo del libro, con una breve descripción.

BDD: *Behavior Driven Development* o desarrollo dirigido o guiado por comportamiento. Forma de desarrollo definida por Dan North, en la que el desarrollo es dirigido por unas especificaciones ejecutables que validan y aprueba los desarrollos que se van creando de forma incremental.

Burn-out chart: Uno de los artefactos *Scrum*. Es una gráfica que muestra la evolución del trabajo en un *Sprint*. En el eje vertical, se muestra la cantidad de trabajo en puntos de historia y, en el horizontal, el tiempo. Una línea señala la evolución ideal y cada día se actualiza con la real. Permite detectar de manera temprana posibles problemas en el desarrollo del *Sprint*.

Business Model Canvas: Ver "Lienzo de modelo de negocio".

Contratos ágiles: Una forma de relación contractual entre un cliente, una empresa para el desarrollo de un proyecto o producto bajo metodologías ágiles. En lugar de cerrar de antemano la definición del trabajo que se va a realizar, los contratos ágiles incorporan mecanismos para poder trabajar con la incertidumbre y requisitos abiertos, como ocurre en los métodos ágiles.

Daily meeting, ***Scrum meeting*** **o** ***Scrum*** diario: Reunión que, con frecuencia idealmente diaria, realiza el equipo *Scrum* para anunciar los avances realizados desde la última, las actividades que se planean realizar hasta la siguiente y los impedimentos encontrados. Esos impedimentos van a parar al *Impediment Backlog*, al tiempo que se les asigna un encargado de resolverlos.

Desperdicio: Es todo aquello que no aporta un beneficio al cliente, que no es valioso para él. En un sentido amplio, esto implica también todo aquello que dificulta o entorpece el proceso para llegar al producto. Eliminar el desperdicio es seguramente uno de los principios de más alcance e impacto en el conjunto de los métodos ágiles.

Épica: Requisito de usuario de gran alcance, imposible de realizar en un único *Sprint*, por lo que debe ser dividido en varias historias de usuario.

Equipo de trabajo: Equipo multidisciplinar formado por todos los perfiles necesarios para la creación del producto o ejecución del proyecto. Junto con el *Product Owner* y el *Scrum Master*, forman el equipo *Scrum*.

Equipo *Scrum*: Es el formado por el *Product Owner*, el *Scrum Master* y el propio equipo de trabajo.

Feedback: Realimentación, comentario o reacción. Se trata de la información que el cliente, usuario, o *stakeholder* traslada al equipo tras cada ciclo de trabajo y ayuda a corregir el trabajo de cara a próximas iteraciones.

Gherkin: Es un lenguaje de dominio creado específicamente para BDD que permite especificar el comportamiento en las historias de usuario.

Historia de usuario: Requisito de producto o proyecto, escrito en lenguaje de negocio, que puede llevarse a cabo en el transcurso de un *Sprint*. Las historias se complementan con unos criterios de aceptación que determinen de forma clara qué es lo que debe ofrecerse para

considerar que se ha completado el resultado.

Impedimentos: Todo aquello que impide completar una tarea o historia de usuario en un *Sprint*. Cuando se identifica uno de estos impedimentos, se declaran en el curso de la reunión diaria, se incluyen en el *Backlog* de impedimentos y se asigna un responsable para resolverlo.

Kanban: Método ágil para la organización de proyectos que permite mostrar permanentemente y de forma muy visual el estado del proyecto a todos los implicados. Kanban es un método muy valioso para la gestión de proyectos con requisitos cambiantes o poco predecibles, aunque puede aplicarse en numerosas situaciones más. Este método basa su funcionamiento en la limitación del trabajo permitido (WIP) en cada etapa del proceso sobre el que se aplica.

LEAN: Filosofía de trabajo basada en la reducción de los denominados "desperdicios" o aspectos que no aportan valor. Estos desperdicios son, entre otros, la producción en exceso, los tiempos de espera entre paso y paso de un proceso o los defectos. LEAN se fundamenta en la búsqueda de todo aquello que no aporta valor al producto para eliminarlo o disminuirlo y, de esta forma, aumentar la calidad del producto y disminuir el tiempo y coste de su producción.

Lean Software Development: Método ágil centrado en la estrategia y su origen está en la empresa de la manufacturación y posterior adaptación al desarrollo de software. Este método tiene tres objetivos principales: reducir drásticamente el tiempo de entrega de un producto, reducir su precio y reducir también el número de defectos o *bugs*.

Lienzo de modelo de negocio: Herramienta para describir, visualizar, evaluar y modificar modelos de negocio.

Metodologías ágiles: Las metodologías ágiles son procesos que dan soporte a la filosofía *Agile*, es decir, son la manera de llevar a la práctica los valores y principios ágiles. Son una serie de reglas que proponen un cambio de paradigma en el desarrollo de proyectos y productos en contraposición a las metodologías tradicionales por fases o de tipo cascada.

Pivotar: En *Lean Startup*, es la acción de cambiar la orientación del servicio o producto, bien por tecnología, público, modelo de negocio, forma de comercialización... Tras un ciclo de Crear-Medir-Aprender, el equipo debe tomar la decisión de perseverar en la línea seguida o pivotar (cambiar).

Product Backlog: Artefacto de *Scrum* que recoge todas historias de usuario y requisitos no funcionales que se quieren implementar para un producto. Es una lista dinámica que cambiará durante todo el ciclo de creación del producto con la aparición o descarte de ítems, orden de priorización de estos y su nivel de detalle.

***Product Owner* (PO o Dueño de Producto)**: El *Product Owner* es un rol en *Scrum*, perteneciente al equipo *Scrum*. Su papel consiste en velar por el producto y su éxito definiendo la visión adecuada de producto, manteniendo el *Product Backlog* actualizado y priorizado, así como creando un plan de entregas acertado.

Producto Mínimo Viable: En *Lean Startup*, es una versión del producto que contiene las características básicas para acceder al mercado y que permite recolectar el máximo de información. También conocido por sus siglas "PMV", "Producto Viable Mínimo" o, en

inglés, "*Minimum Viable Product*" (MVP).

Propuesta de valor: Es aquello que te hace diferente de la competencia y por lo que los clientes querrán pagar. La propuesta de valor, junto con los segmentos de mercado, son el núcleo del negocio.

Puntos de historia: Medida numérica que se asigna a los elementos del *Product Backlog*, que nos ayuda a compararlos entre ellos en términos de esfuerzo para su realización. Esta medida ayuda a crear la velocidad de un equipo en una iteración como la suma de los puntos de cada historia completada dentro de un *Sprint.*

Refactorizar: Consiste en modificar el código sin que cambie su comportamiento para mejorar la claridad del mismo. Esta forma de mantener el código no añade funcionalidad nueva al producto, pero sí que debe simplificar la estructura del código para que su comprensión y mantenimiento sean más sencillos.

Refinamiento: Reunión en la que se prepara el *Product Backlog* para la reunión de planificación del *Sprint*. Se introducen nuevos elementos, se reprioriza, se dividen los elementos existentes y se les añaden criterios de aceptación. Tiene lugar un par de días antes del fin de *Sprint*. Anteriormente, se conocía como "*grooming*".

Retrospectiva: Reunión que tiene lugar al finalizar cada iteración del ciclo de *Scrum*. Estas reuniones tienen como objetivo analizar la manera en que el equipo está trabajando para detectar todo aquello que no es útil para eliminarlo o modificarlo, así como para potenciar y maximizar aquello que sí lo es. Este mecanismo de búsqueda de la mejora constante aumenta la calidad de lo que se construye y es una oportunidad extraordinaria para revisar los posibles riesgos del proyecto y para mejorar la comunicación y la relación entre las personas del equipo.

Scrum: Una de los más populares metodologías o métodos ágiles. Se trata de un marco de trabajo iterativo e incremental, de propósito general, aunque muy utilizado en el desarrollo software. Presentado en 1995 por Ken Schwaber y Jeff Sutherland, y definido en 2001 por Ken Schwaber y Mike Beedle en el libro *Agile Software Development with Scrum*.

Scrum but: Término que se asocia a malas prácticas o modificaciones que se adoptan para justificar por qué no se está siguiendo alguna de las reglas de *Scrum*. Tienen una estructura que normalmente se suele expresar como: "*Usamos Scrum pero + (práctica de Scrum no seguida) + (excusa lógica) + (adaptación de la práctica)*".

Scrum Master: El *Scrum Master* es un rol en *Scrum* perteneciente al equipo *Scrum*. Su papel consiste en responsabilizarse del proceso que se está siguiendo y de garantizar el seguimiento correcto de *Scrum*. Debe garantizar que el equipo puede trabajar sin distracciones o problemas con lo que debe resolver cualquier impedimento que el equipo pueda tener.

Scrum of Scrums: Reuniones de sincronización entre los diferentes equipos de *Scrum* que están trabajando en la construcción de un mismo producto. Estas reuniones tendrán lugar cuando sea necesario dividir a un equipo de trabajo en varios equipos más pequeños y su objetivo es mantener la coordinación entre ellos para no perder el foco en el producto común y detectar de manera temprana sus posibles dependencias.

Sprint: Es el nombre que recibe en *Scrum* cada una de las iteraciones que tienen lugar durante la realización de un producto. El *Sprint* es el periodo de tiempo comprendido entre la planificación del mismo y la *Review* y Retrospectiva.

Sprint 0: Periodo previo al primer *Sprint* de creación del producto en el que se prepara todos los elementos clave para la ejecución de las iteraciones de *Scrum*, la Visión y el *Product Backlog*.

Sprint Backlog: Pila de historias de usuario y tareas que definen el trabajo que se va a realizar durante una iteración o *Sprint*. El contenido está ordenado y se selecciona de acuerdo con el criterio de prioridad que establece el *Product Owner*, pero el alcance (la cantidad de historias incluidas) lo define el equipo según la velocidad estimada para el *Sprint*. Las historias que componen *Sprint Backlog* se seleccionan durante la reunión de Planificación y el equipo las subdivide en tareas durante la Planificación detallada.

Sprint Planning: Reunión dividida en dos fases que tiene lugar antes del inicio de un *Sprint* para definir el trabajo que se va a realizar en él. En la primera fase, se define el alcance del *Sprint* seleccionando qué parte del *Product Backlog* se va a trabajar durante el *Sprint*. Durante la segunda fase, se definen las tareas que son necesarias ejecutar para la realización de cada elemento del *Backlog* elegido.

Sprint Review: Reunión que tiene lugar el último día del *Sprint*, cuyo objetivo principal es la recogida de información o *feedback* sobre el estado del proyecto o producto en desarrollo después del trabajo realizado durante el *Sprint.*

Stakeholder: Cualquier persona interesada o afectada por el producto que se está construyendo. Pueden ser desde los usuarios del proyecto hasta cualquier cliente final, pasando por administradores o gestores. Es importante contar con los *Stakeholders* al menos durante las reuniones de *Sprint Review*, ya que sus impresiones y comentarios sobre el proyecto en el que se está trabajando son fundamentales para conseguir el objetivo con éxito.

Tarea: Definición de trabajos expresados en el lenguaje del dominio técnico del proyecto. El equipo de trabajo define las tareas en el curso de la Planificación detallada a partir de las historias de usuario definidas por el *Product Owner*. Cada tarea debe estar limitada temporalmente en un rango de entre medio y tres días para su realización. Cada tarea cuenta con una "Definición de Hecho" (DoD, *Definition of Done*), que describe los pasos, en el dominio técnico, necesarios para verificar qué se ha completado.

TDD: *Test Driven Development* o desarrollo dirigido por pruebas es una técnica de desarrollo de software cuyo fundamento es escribir la prueba unitaria que debe cumplir el código y, a continuación, escribir el código de manera que pase dicha prueba. Cuando la prueba se pasa con éxito, se debe refactorizar el código para simplificarlo y mejorarlo. Este proceso se repetirá tantas veces como sea necesario. El efecto de programar así es que todo el código habrá pasado pruebas unitarias de forma automática. TDD permite realizar de manera simultánea el diseño, las pruebas, la arquitectura y la codificación.

Tema: Una de las grandes divisiones de la funcionalidad de un producto o proyecto. Cada tema describe un gran bloque funcional ("motor" en el diseño de un coche, "catálogo" en una Web de compras...), que, a su vez, se divide en épicas, bloques menores más manejables que

recogen grandes requisitos de usuario.

Velocidad: Es la cantidad de trabajo, expresada en puntos de historia, que un equipo puede realizar durante un *Sprint*. La velocidad la estimará el equipo y puede calcularse en función del trabajo realizado en *Sprints* anteriores, siempre y cuando se mantenga constante la duración de los *Sprints* y el equipo de trabajo.

WIP: El *Work In Progress* o trabajo en progreso es un concepto utilizado en Kanban e indica el número máximo de requisitos permitidos en un estado determinado del flujo de un proyecto. El objetivo de fijar este WIP es detectar de manera temprana en qué puntos del proceso se producen cuellos de botella para ponerles solución.

XP: *eXtreme Programimng* o Programación extrema es el método ágil de programación más extendido. Este método se basa en la aceptación de que los requisitos de un proyecto suelen cambiar con frecuencia. Por este motivo, propone seguir una serie de prácticas de desarrollo software que permitan construir de manera que los cambios tengan el menor impacto posible.

Referencias

Lecturas recomendadas para ampliar información:

Abrahamsson, Pekka; Salo, Outi; Ronkainen, Jussi. *Agile Software development methods. Review and Analysis.* University of Oulu. 2002.

Anderson, David J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press. 2010. (Inglés). Steve Blank. K&S Ranch. 2005.

Beck, Kent. *eXtreme Programming Explained: Embrace Change*. Ed. Addison Wesley. 1999.

Beck; Kent; Andres, Cynthia. *eXtreme Programming Explained: Embrace Change* (2nd Edition). Ed. Addison Wesley. 2004.

Blank, Steve. *The Four Steps to the Epiphany: Successful Strategies for Products That Win*. K&S Ranch. 2005.

Blank, Steve; Dorf, Bob. *The Startup's Owner Manual*. K&S Ranch. 2012.

Cohn, Mike. "*Leader of the Band*" [en inglés]. http://www.scrumalliance.org/articles/36-leader-of-the-band. Un artículo muy citado que hace un repaso a los principales atributos de un buen *Scrum Master*.

Cohn, Mike. *Succeeding with Agile - Software development using Scrum*. Ed. Addison Wesley. 2009.

Derby, Esther; Larsen, Diana. *Agile Retrospectives: Making Good Teams Great*. Ed. The Pragmatic Programmers. 2006.

Elatta, S. *Top Eight Reasons Why Organizations Are Making the Switch*. Scrum Alliance, 2008.

Hunt, Andrew; Thomas, David. *The Pragmatic Programmer*. Ed. Addison Wesley. 2000.

Jurado, Susana; Olano, María. *Lean Elephants*. 2013.

Kaplan, Robert; Norton, David P. *Strategy Maps: Converting Intangible Assets into Tangible Outcomes*. Ed. Harvard Business Press Books. 2003.

Kniberg, Henrik. *Scrum and XP from the trenches*. http://www.infoq.com/minibooks/scrum-xp-from-the-trenches. Un libro escrito desde la perspectiva del uso diario de *Scrum* y no desde la metodología. Destaca, sobre todo, por su descripción de la experiencia y las dificultades del día a día. La versión española es *Scrum desde las trincheras*. http://www.proyectalis.com/wp-content/uploads/2008/02/scrum-y-xp-desde-las-trincheras.pdf.

Kniberg, Henrik; Skarin, Mattias. *KANBAN and SCRUM. Making the most of both.* C4Media Inc. 2010.

Kniberg, Henrik; Skarin, Mattias. "*Kanban y Scrum obteniendo lo mejor de ambos*". http://www.proyectalis.com/documentos/KanbanVsScrum_Castellano_FINAL-printed.pdf.

Kniberg, Henrik. *Lean from the Trenches.* Pragmatic Bookshelf. 2011.

Leffingwell, Dean. *Scaling Software Agility*. Addison Wesley. 2007.

Leffingwell, Dean; Cockburnm Alistair; Highsmith, Jim. *Scaling Software Agility. Best Practices for Large Enterprises*. Series Editors. 2007.

López de Ávila, Mario; de Miguel, Jose Antonio. *España Lean Startup 2013*. 2013.
Mar, Kane. *An Enterprise Strategy for Introducing Agile*. 2006.
Moore, Geoffrey. *Crossing the Chasm*. Ed. Harper Business Essentials. 1991.
North, Dan. "*Introducing BDD*". https://dannorth.net/introducing-bdd/.
Osterwalder, Alexander; Pigneur, Yves. *Business Model Canvas*. Published by John Wiley & Sons, Inc. Hoboken, New Jersey. 2010.
Palmer, S.R.; Felsing, J.M. *A Practical Guide to Feature-Driven Development.* Upper Saddle River, NJ, Prentice Hall. 2002.
Pichler, Roman. *Agile Product Management with Scrum: Creating Products that Customers Love*. Ed. Addison Wesley. 2010.
Poppendieck. Mary and Tom; Cockburn, Alistair; Highsmith, Jim. *Lean Software Development. An Agile Toolkit*. Series Editors. 2003.
Ries, Eric. *El método de Lean Startup*. Deusto Ediciones. 2012.
Shore, James; Warden, Shane. *The Art of Agile Development*. Ed. O´Reilly. 2008.
Shwaber, Ken; Beedle, Mike. *Agile Software Development with SCRUM*. Publisher: Prentice Hall. 2001. El primer libro publicado sobre *Scrum*.
Shwaber, Ken; Sutherland, Jeff. *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game*. 2011.
Takeduchi, Hirotaka; Nonaka, Ikujiro. "*The New New Product Development Game*". *Harvard Business Review*. 1986.

Estas son algunas direcciones interesantes en Internet:

Página de Mike Cohn: www.mountaingoatsoftware.com.
Página de Roman Pichler: www.romanpichler.com.
Página de Henrik Kniberg: www.crisp.se/kanban.
Blog de Ken Schwaber: kenschwaber.wordpress.com.
Página de Mary y Tom Poppendieck: www.poppendieck.com.
Página de Esther Derby: www.estherderby.com.
Página de Alex Osterwalder: www.alexosterwalder.com/.
Blog de Mario López de Ávila: http://nodosenlared.com/.
Página en español sobre temas relacionados con *Scrum*. Una iniciativa de Xavier Albaladejo: www.proyectosagiles.org.
Página de la herramienta para gestión de proyectos ágiles VersionOne: www.versionone.com.
Página de la Scrum Alliance: www.Scrumalliance.org.
Agile Spain. Comunidad sobre métodos ágiles en español: www.agile-spain.com.
Historia de eXtreme Programming: www.c2.com/cgi/wiki?HistoryOfExtremeProgramming.

InfoQ es una comunidad *on-line* de desarrolladores para desarrollo de software empresarial que tiene una sección específica sobre desarrollo de software con metodologías *agile*: www.infoq.com/agile_techniques.

Retromat. Una colección de técnicas para enriquecer y dinamizar retrospectivas: https://plans-for-retrospectives.com/es/.

Edición en formato digital: 2018

Calle Juan Ignacio Luca de Tena, 15
28027 Madrid

ISBN ebook: 978-84-415-3771-2

Conversión a formato digital: REGA

www.anayamultimedia.es