Gestión ágil en proyectos de *business intelligence*

Caso práctico. Comercio electrónico

Xavier Gumara Rigol

PID_00237500

Tiempo mínimo previsto de lectura y comprensión: **4 horas**

Universitat Oberta de Catalunya

Este material docente ha sido coordinado y revisado por la profesora Isabel Guitart Hormigo (septiembre 2016).

Índice

Introducción

En un mercado que evoluciona rápidamente, algunas organizaciones requieren estrategias ágiles para el lanzamiento de sus productos, donde, frecuentemente, a medida que se va desarrollando el producto, se van incorporando nuevos requisitos. Adicionalmente, demasiado a menudo los proyectos tienen fechas de entrega imposibles, el patrocinador está ausente durante la fase de desarrollo, el alcance del proyecto es inasequible y los requisitos del negocio están incorrectamente establecidos. Estas son algunas de las razones del fracaso de muchos proyectos de *business intelligence*.

Según Gartner, a principios de la década de 2010, entre un 70 % y un 80 % de las implementaciones de proyectos de *business intelligence* fracasaron, entendiendo fracaso como retrasos en la entrega, proyectos que nunca se terminaron o que no hacían lo que se había pedido inicialmente (falta de alineación en el alcance).

¿Pueden las metodologías ágiles reducir la tasa de fracaso de los proyectos de *business intelligence*? Seguramente sí.

Una característica general de la metodología de gestión ágil es la siguiente.

> La gestión de proyectos ágil no se formula sobre la necesidad de anticipación, sino sobre la de adaptación continua.

En un proyecto de *business intelligence*, definiremos la metodología de gestión ágil como sigue.

> La aplicación de diferentes estilos de desarrollo iterativo e incremental a los retos específicos de integración y presentación de datos para la toma de decisiones.

Existen muchos tipos distintos de estilos de metodología ágil, entre los que destacan *Scrum* y *Kanban*. En este documento, nos centraremos en *Scrum* para aplicar la metodología de gestión ágil de proyectos *business intelligence*.

Objetivos

El objetivo general de estos materiales es conocer la metodología *Scrum* como estilo de gestión de proyectos y describir los retos cuando se aplica a proyectos de *business intelligence* y *data warehousing*.

Los objetivos específicos de estos materiales son:

1. Presentar el paradigma de la gestión ágil de proyectos.
2. Entender las principales diferencias entre la gestión de proyectos tradicional y LA ágil.
3. Entender las principales características y ventajas en una gestión de proyectos ágil.
4. Dar a conocer una metodología ágil concreta, *Scrum*.
5. Conocer la adaptación de *Scrum* en la gestión de proyectos en *business intelligence*.
6. Saber aplicar *Scrum* en la gestión de proyectos en *business intelligence*.
7. Dar a conocer casos de uso y ejemplos.

1. Contexto

Antes de presentar las características y ventajas que aporta la gestión ágil de proyectos y las principales diferencias entre la gestión de proyectos tradicional y la ágil, indicamos cómo nació el concepto de metodología ágil.

1.1. Manifiesto ágil

En 2001, diecisiete profesionales del software, críticos de los modelos de producción basados en procesos, se reunieron en Utah para discutir sobre los procesos empleados por los equipos de programación. En la reunión se acuñó el término ***métodos ágiles*** como alternativa a las metodologías formales (PMI, SPICE...) que consideraban excesivamente rígidas y dependientes de planificaciones detalladas previas al desarrollo.

El resultado de esa reunión se publicó como el "Manifiesto ágil" con los cuatro pilares básicos que todo equipo u organización ágil debe aprender a valorar:

- Individuos e interacciones sobre procesos y herramientas.
- Software que funciona sobre documentación exhaustiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre el seguimiento de un plan.

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

Los cuatro pilares del Manifiesto ágil se desglosan en estos doce principios:

1) Nuestra principal prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2) Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3) Entregamos frecuentemente software, en periodos de entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4) Las personas del negocio y el equipo de desarrollo trabajamos juntos de forma cotidiana durante todo el proyecto.

5) 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.

6) El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7) El software que funciona es la principal medida de progreso.

8) Los procesos ágiles promueven el desarrollo sostenible. Los promotores, el equipo de desarrollo y los usuarios debemos ser capaces de mantener un ritmo constante de forma iterativa.

9) La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10) La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11) Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

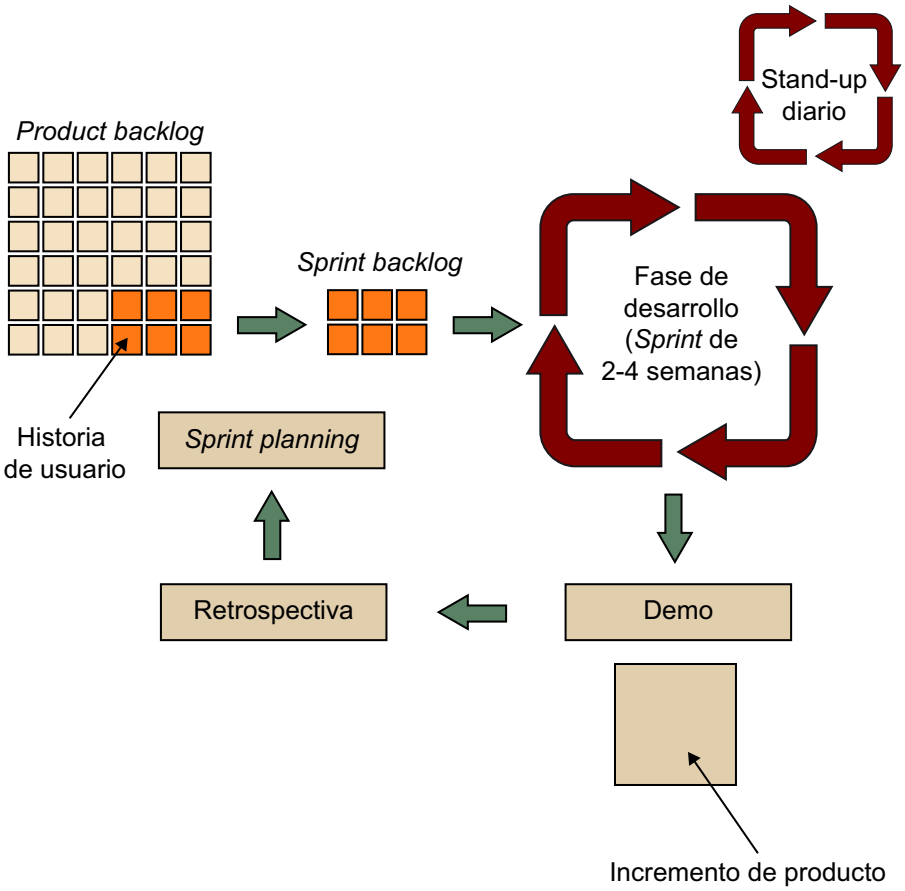
12) A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

En muchas ocasiones, adoptar los principios del Manifiesto ágil supone un reto para el equipo e incluso para la empresa; sobre todo si la organización está acostumbrada a una gestión de proyectos más tradicional. Si se desea implantar un estilo de gestión de proyectos ágil, hay que tener en cuenta que todos los principios citados anteriormente deben ser adoptados y aceptados por todos los niveles de la organización.

1.2. Enfoque ágil

En este subapartado, vamos a introducir los principales conceptos de la gestión ágil e iterativa de proyectos. En la figura 1, se puede observar la simple estructura de una iteración con la que los equipos que utilizan gestión ágil de proyectos construyen sus aplicaciones. Un equipo de aproximadamente 6-8 personas repite este ciclo cada 2, 3 o 4 semanas.

Figura 1. Estructura de una iteración en *Scrum*



Fuente: Xavier Gumara

La estructura y funcionamiento de la iteración es la siguiente:

1) Un listado de requerimientos arranca el proyecto. Este listado de requerimientos está formado por **historias de usuario**, que son simples frases escritas por el negocio que detallan una necesidad funcional. La representante del negocio en el proyecto es la responsable de esta lista y la mantiene ordenada según la importancia de cada historia para el negocio. Obviamente, las historias más prioritarias se colocarán en la parte superior de la cartera.

2) Una vez que la **cartera de historias** esté documentada, el equipo de desarrollo irá tomando historias de la parte superior de la cartera en función de cuántas de ellas podrán ser implementadas en una iteración. Normalmente durará de 2 a 4 semanas, y la duración de la iteración aconsejable será de 2 semanas.

3) El proceso tiene lugar durante la ceremonia llamada **reunión de planificación**, donde los desarrolladores usarán **puntos de historia** para identificar aquellas historias de la parte superior de la **cartera del producto** que pueden convertir en código funcional para la aplicación durante la iteración. Normalmente durante esta ceremonia se descomponen las historias de usuario

en tareas más específicas. Al conjunto de historias de usuario que se van a desarrollar durante una iteración se les llama **cartera de la iteración**.

4) Después de confirmar que el equipo ha definido la cantidad de trabajo adecuado para la duración de la iteración, se pasa a la fase de desarrollo, donde el equipo se **autoorganiza** para entregar el valor añadido al cual se han comprometido durante la reunión de planificación. La principal ceremonia durante esta fase del desarrollo es la **reunión diaria de seguimiento**. Esta reunión se realiza de pie y no debe durar más de 15 minutos.

5) Al final del ciclo de desarrollo, el equipo realiza una **demostración** del valor añadido al producto durante la iteración, en la cual la representante del negocio en el proyecto es presente en representación de los *stakeholders* del proyecto, que pueden asistir también si la parte demostrada es clave para su día a día y si su agenda se lo permite.

6) Finalmente, y antes de la planificación del siguiente ciclo de desarrollo, el equipo realiza una **reunión de retrospectiva** donde se discuten tanto los aspectos buenos como los aspectos que hay que mejorar del último ciclo de desarrollo que acaban de completar. El equipo está listo para empezar un nuevo ciclo.

Se realizan iteraciones siempre y cuando haya historias de usuario en la cartera del proyecto. Cabe destacar que durante la ejecución de una iteración, el negocio podría haber cambiado la prioridad de las historias de usuario que todavía están por empezar o incluso haber añadido o quitado historias. Esto no debería preocupar al equipo de desarrollo, ya que mientras este se centre en la parte superior de la cartera, el negocio puede cambiar el rumbo del equipo cada pocas semanas, llevando el proyecto hacia donde es preciso que se dirija. Esta flexibilidad suele gustar al negocio y es una de las principales ventajas de la gestión ágil de proyectos.

En el siguiente apartado, vamos a repasar este y otros problemas de la gestión tradicional de proyectos y cómo *Scrum* da respuesta a las desventajas de la gestión de proyectos en cascada.

1.3. Diferencias entre gestión de proyectos tradicional y gestión de proyectos ágil

En la gestión de proyectos tradicional, los requisitos se recogen al inicio del proyecto para evitar errores conceptuales. Siguiendo este modelo, todos los requisitos se deberían recoger antes de empezar la fase de diseño y todos los diseños se deberían completar antes de la fase de desarrollo. Finalmente, las pruebas del software deberían realizarse después de finalizar una gran parte del código. En este proceso, cada fase de trabajo debe completarse antes de empezar la fase siguiente.

Los proyectos que utilizan una gestión tradicional pueden verse como una apuesta segura para aplicaciones grandes que requieren integración de datos desde diferentes aplicaciones, porque los ingenieros pueden pensar con antelación todos los aspectos del proyecto. Una metodología ágil, en cambio, solo recoge unos pocos requisitos de la parte superior de una cartera y los convierte en código antes de que se tenga en cuenta el siguiente conjunto de características.

En un mercado que está continuamente cambiando, hay razón para pensar que los proyectos gestionados con una metodología tradicional seguirán siendo muy arriesgados para la construcción de grandes sistemas. Las especificaciones de cada una de las fases van a contener defectos, ya que las personas encargadas de prepararlos no tienen la visión suficiente, especialmente en sectores donde el mercado cambia día tras día, donde hay más competencia entre empresas o donde cada vez se premia más ser el primero en lanzar un nuevo producto o una nueva funcionalidad.

Lamentablemente, con un método tradicional, se hace difícil al equipo de desarrollo hacer una revisión profunda de los requerimientos si encuentran errores en el código, arquitectura o diseño. La fase de prueba, confinada a la última fase del proyecto, no puede usarse como un replanteamiento del sistema desde su diseño hasta su puesta en producción.

En este caso, el proyecto entero será un éxito o un fracaso basándose en la calidad del paquete completo de requisitos, la calidad del paquete completo del diseño y, finalmente, la calidad del código como un todo.

Ejemplos del desarrollo del *data warehouse* en proyectos de *business intelligence*

Existen algunos ejemplos de proyectos de *business intelligence*, en concreto del desarrollo del *data warehouse*, que al utilizar una metodología tradicional han observado lo siguiente:

- Los requisitos del proyecto no se recogieron correctamente.
- El cliente no sabía lo que quería porque desconocía las capacidades y las limitaciones de una base de datos analítica (*data warehouse*).
- El tiempo empleado en la fase de diseño se dilata más de lo planificado, reduciéndose el tiempo en la fase de programación (lo que provoca programar con prisas) y la fase de pruebas del software, que serán menos exhaustivas.
- El equipo de desarrollo construye un software con aspectos diferentes a los requisitos especificados por el cliente.
- Los datos son incorrectos, incompletos y duplicados, es decir, de baja calidad, lo que comporta problemas en la fase de integración de datos y en el desánimo del equipo.
- En el momento de entregar la aplicación, el negocio había cambiado y el cliente quería incorporar nuevas funcionalidades.

1.4. Ventajas del desarrollo ágil

En una metodología de gestión de proyectos ágil, se destacan las siguientes ventajas:

- **Incrementos de pequeño alcance**. La restricción en el tiempo en cada iteración reduce el conjunto de características del producto que pueden implementarse a la vez.

- **Centrado en el negocio**. El hecho de incluir una interlocutora del negocio en el equipo del proyecto sirve de guía para el equipo y ofrece una validación constante del trabajo realizado.

  Esta interlocutora ve un flujo constante de mejoras en la aplicación que el negocio puede entender y visualiza claramente que el equipo de desarrollo está aportando valor. El negocio también está validando el trabajo cuando este es completado, procurando que el equipo cometa el mínimo número de errores posible y manteniendo los riesgos bajo mínimos. Descubrir errores pronto y a menudo hace que el equipo no pierda mucho tiempo programando malentendidos y, en general, acelera la entrega de valor.

- **Misma ubicación**. El hecho de que los equipos se junten en un mismo espacio para construir cada iteración permite que los equipos compartan especificaciones de forma verbal, cara a cara, lo que permite eximir la escritura de la mayoría de especificaciones que un método en cascada requeriría.

- **Equipos autoorganizados**. Los ingenieros en el equipo son libres de organizar su trabajo como mejor les convenga, ya que el objetivo de la iteración ha quedado definido tras la reunión de planificación. El equipo tiene la habilidad y la autoridad de tomar las decisiones necesarias y adaptarse fácilmente a los posibles cambios. Se habla de equipos **autoorganizados** por ser cada uno de los miembros del equipo el que escoge las tareas en las que trabajar sin esperar a que un gestor del proyecto se las asigne.

- ***Just-in-time***. El equipo de desarrollo clarifica los requisitos cuando empieza a trabajar en una historia de usuario. Cuando estas tareas suceden lo más juntas posible al hecho de programar, la calidad y la velocidad de desarrollo se maximizan, porque el contexto de negocio y sus restricciones están todavía frescos en la mente del equipo de desarrollo.

- **Especificaciones 80-20**. Los equipos revisan los requerimientos por anticipado y diseñan la arquitectura de datos de la aplicación antes de que la implementación empiece. Sin embargo, estas revisiones y diseños se hacen solo hasta que la solución es suficientemente clara como para empezar a programar. Se entiende que mientras los puntos principales de integra-

ción de la aplicación estén claros, los detalles de cada uno de los módulos pueden esperar a ser tratados cuando se empiecen a implementar dichos módulos.

Conviene nombrar los resultados de este enfoque como "especificaciones 80-20". Usando este tipo de especificaciones, los programadores empiezan a programar cuando tienen claros el 80 % de los detalles más importantes de un módulo, ya que el hecho de trabajar en la misma ubicación y los incrementos pequeños de alcance rellenarán los detalles restantes en el momento en que se necesiten.

Bibliografía recomendada

Ralph Hughes (2013). *Agile Data Warehousing Project Management*. Waltham, MA: Elsevier, Inc.

Definir el 80 % de la parte más importante de un módulo requiere típicamente solo el 20 % del tiempo que un método *waterfall* hubiera invertido en construir las especificaciones completas; por este motivo el enfoque 80-20 acelera los equipos ágiles.

- **Falla rápido y soluciona con rapidez**. Las metodologías ágiles no consideran los fallos en desarrollo como malos, siempre que estos tengan un alcance pequeño y ocurran suficientemente pronto como para ser detectados y corregidos.

  Los equipos ágiles quieren detectar fallos antes de que haya pasado demasiado tiempo, ya que, si hay que solucionar un problema, siempre es mejor hacerlo lo antes posible. Con el enfoque incremental, solo una pequeña parte de la solución global es conducida por el equipo desde la fase inicial de toma de requisitos hasta la puesta en producción. La naturaleza incremental de la metodología ágil permite a sus equipos descubrir errores pronto en el proceso y en las entregas parciales, poniendo solo una pequeña parte de la aplicación en riesgo.

- **Seguro de calidad integrado**. Se construye software de calidad y testado durante la implementación de todas las historias de usuario, en lugar de dejar la calidad como una fase más al final del proyecto. En cada iteración, los programadores escriben las pruebas que los módulos deben pasar antes de que la programación haya empezado. El equipo de desarrollo también revisa el trabajo de los compañeros, y la persona asignada del negocio valida que cada entrega cumpla con su propósito.

  Aparte de los errores de código, las buenas prácticas en temas de calidad de las metodologías ágiles también permiten identificar y resolver pensamientos equivocados y aquellos patrones de trabajo que han llevado a cometer los errores. De este modo, en cada iteración se mejora la calidad del código producido, la calidad del proceso y del trabajo en equipo.

- **Mejores resultados**. Todos los aspectos listados anteriormente permiten al equipo de desarrollo ahorrar tiempo y esfuerzo, propiciando uno de los máximos exponentes de la metodología ágil: entregar valor al negocio constantemente.

1.5. Gestión ágil de proyectos *business intelligence*

Hasta ahora hemos comentado las características de la metodología ágil en comparación con las más tradicionales, pero lo hemos hecho desde una perspectiva general, hablando de "aplicaciones" o "productos", sin centrarnos en los proyectos de *business intelligence*, que forman parte del conjunto de aplicaciones más complejas que una organización posee. Aplicar las metodologías ágiles "tal cual" a la gestión de proyectos de *business intelligence* se hace más complicado cuando la envergadura del proyecto comprende diversidad de áreas, que pueden ir desde la construcción de cuadros de mando hasta la integración de datos. Estas áreas están en las antípodas en cuanto al tipo de trabajo realizado y las herramientas usadas.

- **Integración de datos.** La integración de datos requiere la implementación sucesiva de las siguientes cuatro fases:

  – Extraer los datos necesarios de los sistemas origen.
  – Limpiar los datos y estandarizar su formato.
  – Integrar los datos desde las múltiples fuentes de datos origen.
  – Transformar los datos integrados en estructuras dimensionales para la correcta exploración de los datos.

- **Cuadros de mando**. Actualmente, las herramientas de construcción de cuadros de mando son bastante maduras y fáciles de usar, por lo que la implementación de los cuadros de mando es relativamente sencilla. Sin embargo, antes de que los datos se puedan visualizar, deben ser preparados por un sistema de integración de datos. Los datos deben extraerse del origen, ser transformados para cumplir con las definiciones que el negocio propone y cargados de vuelta a una base de datos de destino que va a crecer en volumen carga tras carga.

  Añadir un nuevo elemento en un cuadro de mando ya existente puede resultar un trabajo simple, como hacer un poco de espacio en el cuadro de mando para añadir el nuevo elemento que se ha de mostrar. En cambio, implicará un mayor trabajo para modificar el proceso ETL asociado, y habrá que construir y probar cada una de las cuatro fases por separado, para que el nuevo elemento del cuadro de mando presente los datos requeridos.

2. Metodología ágil: *Scrum*

Scrum es un modelo ágil caracterizado por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos autoorganizados que en la calidad de los procesos empleados.
- Solapar las diferentes fases del desarrollo, en lugar de realizar una fase tras otra en un ciclo secuencial.

Actualmente, *Scrum* es el método ágil más popular por su facilidad de aprendizaje y puesta en práctica. De hecho, *Scrum* ha conseguido tal repercusión en el mundo del desarrollo de software que, si alguien no especifica otro enfoque, se asume *Scrum* como el método utilizado cuando alguien hace referencia a los métodos ágiles.

Aplicación de *Scrum*

En los proyectos de *business intelligence*, como el desarrollo de *data warehouse* y la construcción de informes y cuadros de mando, se puede aplicar *Scrum* sin modificar, a diferencia de los proyectos de integración de datos, donde habrá que modificar algunos aspectos en la aplicación de *Scrum*.

2.1. Los roles en *Scrum*

En una metodología ágil los miembros del equipo son: la propietaria del producto, el *Scrum master* y el equipo de desarrollo. A continuación, se describen las características y las responsabilidades de cada uno.

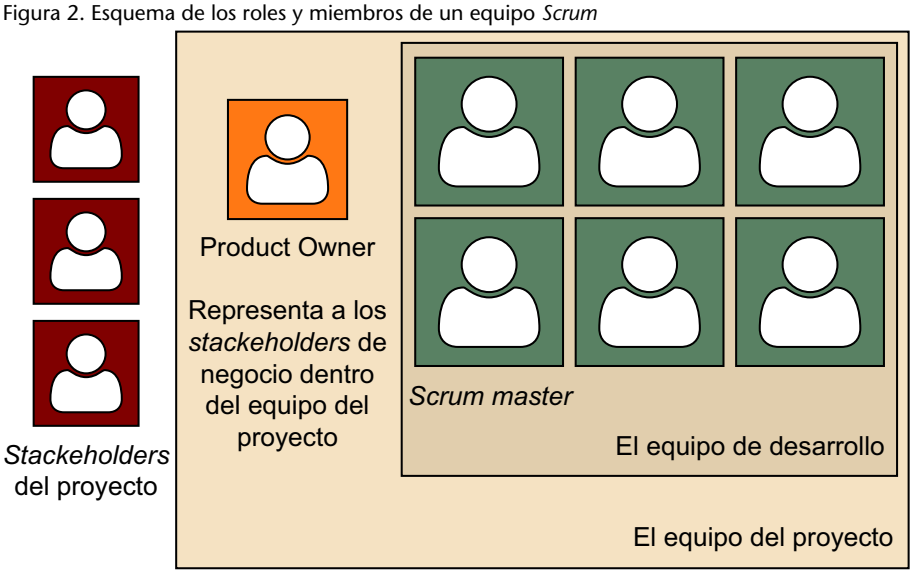
2.1.1. Propietario del producto

El propietario del producto es el representante del negocio y quien toma las decisiones del cliente.

Para simplificar la comunicación y toma de decisiones, es necesario que este rol recaiga en una única persona (figura 2). El propietario del producto debe tener el conocimiento suficiente del producto y de las funcionalidades que se han de implementar, ya que proporcionará al equipo de desarrollo las directrices que habrá que seguir durante el proyecto. Además, esta persona debe tener las atribuciones necesarias para tomar decisiones.

El propietario del producto es también conocido como *product owner*.

Figura 2. Esquema de los roles y miembros de un equipo *Scrum*



Fuente: Iconos diseñados por Freepik

Entre sus responsabilidades destacamos las siguientes:

- Dirigir el proyecto como persona que representa el negocio. Es la persona que representa directamente a la organización que financia el proyecto; por este motivo, está muy focalizado en maximizar los resultados.

- Decidir las características que hay que construir en la aplicación, en qué orden añadirlas, y cuáles no.

- Descartar las implementaciones que sobrepasan la capacidad del equipo en una determinada iteración, entrega o incluso proyecto.

- Responder a las preguntas del equipo de desarrollo sobre los requerimientos y las historias de usuario en tiempo real y cara a cara. El propietario del producto tendrá que trabajar junto al equipo de desarrollo durante la creación del software, en lugar de participar en remoto desde otra oficina.

  El propietario del producto tiene que involucrarse continuamente con el equipo de desarrollo. Su participación es más elevada en las etapas de planificación y demostración del producto.

- Validar y aceptar los distintos componentes desarrollados tan pronto como estos sean finalizados.

  Si en el momento de la entrega del proyecto la aplicación no cumple con las expectativas de la organización que ha pagado por su desarrollo, todos en el equipo habrán fallado, pero estos fallos habrán pivotado sobre las directrices y revisiones del propietario del producto.

Ejemplo de principios ágiles de los líderes de proyecto

En algunos proyectos *business intelligence*, como el desarrollo del *data warehouse*, esta revisión instantánea de los nuevos módulos es difícil de conseguir debido a que los datos necesarios para validar una ETL no siempre están disponibles al instante. Esta situación es otro ejemplo donde los líderes del proyecto deben tener los principios ágiles en mente para improvisar por encima de las técnicas que se encuentran en los libros de texto sobre el *Scrum* genérico.

2.1.2. *Scrum master*

Normalmente, una persona del equipo de desarrollo se encarga del rol de *Scrum master*, gestionando su tiempo entre el desarrollo del proyecto y el facilitar las buenas prácticas de *Scrum* (ver figura 2).

Entre sus responsabilidades destacamos las siguientes:

- Actúa como persona que facilita cuando sea necesario, asegurando que se entienden en la organización y se trabaja conforme a ellas.
- Asesora y proporciona la formación necesaria al propietario del producto y al equipo de desarrollo.
- Revisa y valida la cartera del producto.
- Modera las reuniones.
- Resuelve los impedimentos que en la iteración pueden entorpecer la ejecución de las tareas.
- Gestiona las dificultades de dinámica de grupo que se puedan generar en el equipo.

Los gestores de proyecto en *Scrum* no hacen las funciones de controlar y dirigir al equipo de desarrollo, sino que son los encargados de organizar los recursos y la información del departamento y del proyecto padre dentro de la sala de desarrollo. Por ejemplo, si una persona del equipo de desarrollo deja la empresa, es el gestor del proyecto el encargado de asegurar un reemplazo para el equipo.

Equipo de desarrollo

El resto de miembros del equipo son “desarrolladores”, tal y como puede observarse en la figura 2.

2.2. Definición de requisitos

Las metodologías ágiles han cambiado la definición de requisitos a piezas mucho más pequeñas que son escritas y definidas a lo largo del proyecto. Muchos de estos métodos encargan al propietario del producto que exprese las características para ser añadidas a la aplicación escribiendo un liviano artefacto llamado “historia de usuario”.

Las historias de usuario son el componente principal de la cartera del proyecto, y cada historia define en una o dos frases una pequeña función que el negocio necesita que la aplicación cumpla.

Definición de requisitos

La definición de requisitos es conocida en *Scrum* como historia de usuario o *user story*.

El propietario del producto es el encargado de escribir las historias de usuario, que están formadas por tres componentes clave:

Como `<ROL>`
Quiero `<FUNCIONALIDAD>`
Para `<BENEFICIO>`

El `<ROL>` corresponde a un usuario particular del producto que el propietario del producto visualiza utilizando la aplicación en esta historia en concreto. Responde a la pregunta "¿quién?".

La `<FUNCIONALIDAD>` corresponde al uso que este actor querrá conseguir usando la aplicación. Responde a la pregunta "¿qué?".

Y por último, el `<BENEFICIO>` refleja el valor de negocio que el actor obtendrá del uso de la aplicación. Responde a la pregunta "¿por qué?".

Ejemplos de historia de usuario para una aplicación de BI

"**Como** analista, **quiero** poder diferenciar las compras que se hacen mediante la plataforma web de las que se hacen mediante aplicaciones móviles **para** conocer el éxito que tienen nuestras aplicaciones móviles recién lanzadas."

"**Como** analista financiero, **quiero** incluir la comparación con el mes anterior y el mismo mes del año pasado en el gráfico de líneas del cuadro de mando de ventas por mes **para** poder comparar el volumen de ventas del mes actual con los meses adecuados."

"**Como** *data scientist*, **quiero** una tabla agregada por usuario que indique el volumen total de compra de cada usuario en euros, número de productos, número de categorías distintas en las que ha comprado y fecha de la última compra **para** poder generar un modelo de predicción de hábitos de compra de nuestros usuarios."

La **carpeta de la iteración** es el conjunto de historias de usuario que el equipo se compromete a convertir en código durante la siguiente reunión.

La carpeta de la iteración es también llamada *sprint backlong*.

La **cartera del producto** es el conjunto de historias de usuario que forman la definición del producto que la propietaria del producto ha previsto.

La carpeta del producto es también llamada *product backlong*.

La **cartera de lanzamiento** es el subconjunto de historias que la propietaria del producto prevé que formarán parte del siguiente conjunto de características disponibles para el usuario final en la siguiente puesta en producción.

La carpeta de lanzamiento es también llamada *release backlong*.

Cuando las historias de usuario son cortas (tal y como se recomienda), ponerlas todas juntas por escrito en tarjetas o en un soporte digital da al equipo una forma muy efectiva de visualizar y gestionar el proyecto.

Colección de historias

Un proyecto de *business intelligence* como el desarrollo del *data mart* puede implementarse con una colección de entre 10 y 20 historias de usuario. Un *data warehouse* para una parte concreta del negocio puede especificarse con entre 40 y 80 historias de usuario.

Si las historias de usuario se imprimen en tarjetas o se escriben en notas adhesivas, se pueden esparcir en una superficie grande o pegar en una pizarra y la propietaria del producto podrá ordenarlas por prioridad; mientras, el equipo puede buscar también el orden que minimiza tanto las dependencias entre historias, como el riesgo general del proyecto. Este tipo de planificación, tan visual y colaborativa, da como resultado un plan de proyecto que tanto el equipo de desarrollo como el negocio pueden entender.

Usar historias de usuario breves y con la información imprescindible hace que los requisitos del proyecto se puedan ir alineando con las condiciones cambiantes del negocio. Si nuevas ideas aparecen durante el proyecto, la propietaria del producto puede crear historias de usuario adicionales de forma muy rápida. De la misma forma, aquellas historias que ya no sean relevantes para el proyecto pueden ser descartadas fácilmente. Nadie va a sufrir por haber eliminado una historia ya que solo se ha invertido unos minutos en escribirla.

En resumen, el uso de historias de usuario ayuda a capturar mejor las necesidades de negocio y permite adaptarse al cambio más fácilmente, lo que da como resultado una noción más ajustada de lo que la aplicación debería ser.

2.3. Fases de las iteraciones

Una vez que el equipo dispone de una colección de historias de usuario priorizadas por la propietaria del producto, los equipos de desarrollo, empezando por la parte superior de la lista, van a codificar las historias de usuario de forma iterativa. Estas iteraciones van a tener siempre la misma duración; normalmente entre 2 y 4 semanas dependiendo del equipo y del proyecto, pero manteniendo la duración durante todo el desarrollo.

La cartera del producto es también conocida como *product backlog*.

Durante la fase de iteración:

Las iteraciones son llamadas *sprints*.

- El primer día de la iteración (o medio día, dependiendo de la duración) se dedica a planificar la iteración en la reunión de planificación.

- El resto de días de la iteración, exceptuando el último (o el último medio día), se dedican al desarrollo puro de la aplicación.

La reunión de planificación es conocida como *sprint planning*.

- La última parte de la iteración se dedica a la revisión de las nuevas funcio-

nalidades añadidas durante la iteración, y se dedican también unas horas a reflexionar sobre cómo el equipo puede mejorar sus prácticas y técnicas de desarrollo durante una reunión de retrospectiva.

El esquema de todas las ceremonias de *Scrum* puede verse en la figura 1. En los siguientes subapartados vamos a comentar cada una de estas ceremonias en más detalle.

2.3.1. Reunión de planificación

El objetivo principal de la reunión de planificación es obtener el conjunto de historias de usuario que el equipo de desarrollo se compromete a implementar durante la iteración.

Duración de la reunión de planificación

La duración de la reunión de planificación varía según la duración de la iteración, la cantidad de personas que forman parte del proyecto y de lo bien o mal definidas que estén las historias de usuario. Por regla general, se suele dedicar una mañana para la reunión de planificación en iteraciones de dos semanas de duración.

Si el equipo es nuevo, es probable que en las primeras iteraciones se requiera más tiempo, ya que probablemente las historias de usuario deberán ser formuladas en un lenguaje que tanto el equipo técnico como la propietaria del producto entiendan y con el que estén cómodos.

Cuando tanto el equipo de desarrollo como la propietaria del producto tengan más rodaje en la metodología *Scrum*, los distintos miembros del equipo empiecen a compartir el mismo lenguaje y la aplicación esté en una fase avanzada de desarrollo, las reuniones de planificación van a durar menos tiempo, de manera que pueden llegar a durar unas 2 horas para una iteración de dos semanas.

El reto más importante en las reuniones de planificación es conocer cuántas historias de usuario van a caber en la siguiente iteración. Para cumplir con este objetivo, el equipo necesita estimar el trabajo en la implementación de cada historia de usuario mediante los puntos de historia.

Los puntos de historia son también llamados *story points*.

La unidad de la métrica de los puntos de historia (es decir, el equivalente a un punto de historia) es definida por el equipo de desarrollo la primera vez que empiezan a trabajar juntos como equipo y nunca se traduce en horas o líneas de código. Esta característica permite que los puntos de historia sean usados solamente para comparar el tamaño de una historia respecto a otra y para conocer rápidamente cuántas historias van a caber en una iteración.

El número de los puntos de historia de una historia se especifica comparando el trabajo hecho en iteraciones anteriores y por el tiempo que podrán dedicar al desarrollo en la próxima iteración. En el caso de no tener todavía estas referencias, se puede coger la mínima unidad de desarrollo y asignar un punto de historia.

Número de puntos de historia

En proyectos de *business intelligence* se puede usar la construcción de una dimensión sin jerarquías como la unidad básica de puntos de historia. Si hay que estimar el tiempo que se tardará en construir una tabla de hechos, el equipo deberá evaluar la complejidad de la misma para decidir a cuántos múltiplos de la unidad equivale dicha implementación.

Para estimar una historia en puntos de usuario, se pueden utilizar varias técnicas; a continuación, se describe una de las más usuales: El ***planning poker*** busca llegar al consenso de las estimaciones en puntos de una historia de usuario del equipo de desarrollo. Una vez que la propietaria del producto ha explicado la historia de usuario que se ha de estimar y se ha discutido a alto nivel, cada participante realiza una estimación en privado mediante un juego de cartas especiales (figura 3). La propietaria del producto no participa en estas estimaciones.

Figura 3. Un mazo de cartas para *planning poker*

Fuente: Xavier Gumara

Planning poker

Cada participante recibe un juego de cartas. Una vez que la propietaria del producto ha explicado la historia de usuario que se ha de estimar y se ha discutido a alto nivel, cada participante selecciona una carta con su estimación en privado y la pone boca abajo encima de la mesa.

Se puede utilizar el 0 para cosas que ya estén hechas, el infinito para cosas imposibles y el signo de interrogación si no se tiene ninguna pista sobre cuántos puntos puede costar implementar la historia en cuestión.

Una vez que todos los desarrolladores han elegido su carta, se da la vuelta a todas las cartas al mismo tiempo, y si las estimaciones no coinciden, el equipo debe discutirlas, haciendo especial hincapié en los valores extremos. Es importante que el equipo tome el tiempo necesario para escuchar a los desarrolladores que han elegido los valores extremos, ya que pueden aportar conocimiento sobre la aplicación que puede hacer cambiar la estimación de los demás participantes. Normalmente, es habitual llegar al consenso calculando la media de puntos de historia que ha sugerido cada miembro del equipo si estos no distan mucho entre ellos.

En general, si la estimación de una historia de usuario es consensuada pero es un valor muy alto, se debería intentar partir la historia de usuario en otras más pequeñas para minimizar los riesgos de implementación. Esto se puede hacer durante esta misma sesión de la reunión de planificación con la propietaria del producto presente, ya que también le servirá a ella para conocer la complejidad de la aplicación y ser más realista durante la autoría de historias de usuario futuras.

Una vez elegida y estimada una historia de usuario, es el momento de descomponerla en las tareas necesarias para dotar la aplicación de la nueva funcionalidad especificada en la historia. Del mismo modo que con las historias, las tareas se pueden plasmar en tarjetas (físicas o digitales) para poderlas mover y cambiar de orden. Las tareas pueden incluir trabajo de análisis, diseño, desarrollo, pruebas o documentación, ya que en *Scrum* todas las fases de la producción de software se hacen en paralelo.

Finalmente, hay dos aspectos importantes que se deben tener en cuenta antes de terminar una reunión de planificación:

1) Decidir un objetivo de la iteración. Junto con la propietaria del producto se puede escoger un tema para la iteración que ayude al equipo de desarrollo a mantener el foco y centrarse en aquellas historias realmente importantes. El objetivo puede ser la historia de usuario más grande o un conjunto de historias relacionadas entre ellas.

2) Establecer nuevas historias de usuario. La propietaria del producto debe dejar algunas historias de usuario bien definidas en la parte superior de la nueva y reducida cartera del producto para que los desarrolladores puedan asignarse historias nuevas si algunas de las actuales se bloquean durante la iteración (por cualquier motivo imprevisible) o por si finalizan antes de lo previsto las tareas de la cartera de la iteración.

2.3.2. Fase de desarrollo

La **fase de desarrollo** se caracteriza por las reuniones de seguimiento diarias y la autoorganización del equipo.

Las **reuniones de seguimiento diario** sirven para mantener al equipo coordinado entre la reunión de planificación y la demostración de la iteración para tomar el pulso del equipo en 15 minutos.

La fase de desarrollo es el corazón de un ciclo ágil, ya que, en una iteración de dos semanas de duración, la fase de desarrollo dura unos nueve días.

Reuniones de seguimiento

Las reuniones de seguimiento son también conocidas como ***stand-ups***. La reunión se realiza de pie, no alejados del espacio habitual de trabajo, sin salas cerradas, ni diapositivas, ni reuniones eternas.

Durante las reuniones de seguimiento, cada miembro del equipo de desarrollo, por turnos, hace tres anuncios distintos que responden a las siguientes preguntas:

- ¿Qué hice ayer?
- ¿Qué voy a hacer hoy?
- ¿En qué estoy bloqueado?

Compartir esta información es esencial, ya que ayuda a conocer si hay dependencias entre las personas del equipo que ya han sido resueltas, si hay duplicidad de esfuerzos por parte de dos miembros del equipo o qué desarrolladores están atascados en sus tareas diarias y necesitarían un poco de ayuda.

Tan importante es conocer el contenido de las reuniones diarias como conocer qué no debe incluirse en estas reuniones. El hecho de mantener la reunión guiada por estas tres preguntas hace que el equipo de desarrollo hable solo de las historias que tienen en mano, que han finalizado o van a empezar; por lo tanto, cualquier otro detalle o entrar muy en detalle en alguna de las tareas en curso no es el objetivo de la reunión y se deberá comentar aparte solo con aquellas personas implicadas en el desarrollo de la tarea. En este caso, el rol del *Scrum master* es muy importante para detectar las reuniones de seguimiento que se desvían del foco y anunciarlo en el momento al equipo para volver a centrar la reunión y asegurar una reunión efectiva y de corta duración.

Técnica del reloj de arena

Cuando el *Scrum master* detecte que las reuniones de seguimiento se están alargando demasiado, se pueden aplicar varias técnicas para reducir el tiempo de la reunión. Una de estas consiste en usar un pequeño reloj de arena de 1-2 minutos que el desarrollador pueda ver y que le indique el tiempo restante que le queda para realizar su exposición.

Aunque las reuniones sean cortas, es importante ser metódicos en su realización sea cual sea la situación del equipo, y también hacerlas cada día a la misma hora. Aunque se recomienda hacerlas por la mañana para responder literalmente a las preguntas propuestas anteriormente, se pueden hacer a cualquier otra hora del día; en este caso, comunicando el progreso desde la última reunión de seguimiento. Al finalizar el día, el *Scrum master* también actualiza un gráfico que muestra el número total de puntos de historia de usuario que quedan de las tareas incompletas en la iteración actual.

El gráfico diario es conocido como *burn down chart*.

Otro concepto clave de la fase de desarrollo es la **autoorganización**:

1) El equipo decide en todo momento qué miembros trabajarán en cada tarea. En lugar de tener a un gestor de proyecto asignando tareas, es el equipo el que decide la mejor forma de emparejar tareas con desarrolladores. Cada miembro del equipo va a ir moviendo las tarjetas de tarea en una pizarra (física o digital) que permite visualizar el avance del desarrollo. Como todos comparten la misma fecha de entrega, si un miembro del equipo empieza a quedarse atrás,

el resto lo va a notar. El hecho de que se vaya a tener éxito o fracasar como equipo en la iteración es un factor motivador que hará saltar las alarmas del equipo entero, el cual buscará conjuntamente una solución para el problema.

2) La autoorganización mejora la visibilidad de la calidad del software desarrollado. Genera que el trabajo de cada desarrollador cumpla con los estándares impuestos por el equipo. Si un compañero de equipo empieza a dejar trabajo incompleto de forma continuada, el resto del equipo lo va a notar y se deberá poner solución al problema.

3) El equipo es libre de inventar o adaptar herramientas y técnicas de otras metodologías ágiles de gestión de proyectos que mejoren la velocidad del equipo sin dañar la calidad del código producido. Algunas de las herramientas más usadas pueden ser las técnicas de calidad.

2.3.3. Demostración para el usuario

La reunión de demostración es la reunión donde los programadores enseñan a la propietaria del producto las nuevas características de la aplicación que acaban de desarrollar en la iteración que concluye.

Al final de la fase de desarrollo, la propietaria del producto ya está familiarizada con la mayoría de las entregas en la iteración, ya que las ha ido aceptando o rechazando a medida que avanzaba la iteración. En este caso, en el último día de la iteración, la propietaria del producto será capaz de demostrar las nuevas características a los *stakeholders* del proyecto, mostrándoles la nueva versión de la aplicación que los programadores han logrado construir bajo su dirección. Esta presentación dará al equipo una retroalimentación muy valiosa de su trabajo. El hecho de que se muestre la aplicación al usuario final permite llamar a este evento **demostración para el usuario**.

Demostración para el usuario

La definición de demostración para el usuario se aplica perfectamente para equipos que se encargan de construir aplicaciones de software general o interfaces de *business intelligence* como informes y cuadros de mando. Sin embargo, los proyectos de integración de datos entran en conflicto con este paradigma. Los datos para una simple demostración no van a estar disponibles hasta que el equipo finalice las tareas de transformación de datos y pasen un día o dos rellenando las tablas destino. En este escenario, el último día de la iteración es el primer momento en el que la propietaria del producto podrá dar una opinión realista sobre las nuevas características implementadas. Los ingenieros de datos todavía requerirán de comentarios y retroalimentación de los *stakeholders*, así que el equipo todavía tendrá que realizar otra demostración de las nuevas funcionalidades.

Demostración en proyectos de almacenes de datos

En los proyectos de almacenes de datos del *business intelligence*, la propietaria del producto también va a revisar la calidad de los datos cargados en las tablas destino como parte de la demostración.

Es importante que la propietaria del producto pruebe las nuevas características por sí sola, porque hay más posibilidades de que el software sea usado de maneras más inesperadas. La propietaria del producto irá historia por historia (en orden de prioridad), asegurando que cada una de ellas se ha implementado correcta y completamente.

Aquellas historias de usuario que la propietaria del producto acepte son candidatas a ser subidas a producción en la siguiente subida a producción. Los puntos de estas historias se van a tener en cuenta para calcular la velocidad del equipo en esa iteración.

Si la propietaria del producto rechaza una historia, esta es puesta de vuelta a la cartera del producto. Si la historia sigue siendo relevante, los desarrolladores la verán en la parte superior de la carpeta durante la siguiente reunión de planificación. En caso contrario, si las condiciones han cambiado y la historia deja de ser relevante o ya no se necesita, la propietaria del producto puede desplazarla al final de la carpeta o incluso eliminarla.

Muchos equipos de *Scrum* utilizan una variante de la demostración a la explicada anteriormente. Durante la iteración, y mientras las historias de usuario se van completando, los desarrolladores piden a la propietaria del producto que examine las historias completadas y decida su aceptación o rechazo en ese momento. De este modo, al final de la iteración, la mayoría de los módulos habrán sido aceptados, lo que permitirá a los equipos ampliar la demostración a más personas interesadas. Sin necesitar ya el tiempo de una aceptación/rechazo formal, la propietaria del producto puede invitar al grupo de *stakeholders* para revisar el software construido.

Demostración en los proyectos de integración de datos

En proyectos de integración de datos de un *business intelligence* probablemente se va a dedicar una sesión separada de la demostración e independiente de las iteraciones en la que el equipo pueda recibir retroalimentación y comentarios del grupo completo de *stakeholders* del negocio una vez que los datos estén cargados en el *data warehouse* y la propietaria del producto haya validado la calidad de los mismos.

2.3.4. Reunión de retrospectiva

La reunión de retrospectiva se organiza después de la demostración para reflexionar sobre la efectividad como equipo y para identificar nuevos comportamientos que permitirán al equipo entregar más rápidamente y con mayor calidad durante la siguiente iteración.

También es considerada por muchos como la fase más importante, ya que permite que el equipo optimice sus propias prácticas en muchos ámbitos. Es importante recalcar que la propietaria del producto debe estar invitada en el proceso de retrospectiva, ya que no solo se trata de reflexionar sobre la efectividad del equipo, sino también sobre la relación con el negocio. La persona que tenga el rol de *Scrum master* es quien liderará la reunión.

El objetivo de la reunión de retrospectiva es que la siguiente iteración sea menos estresante y más efectiva.

Los pasos realizados durante la reunión de retrospectiva son los siguientes:

1) El *Scrum master* pide a todos los asistentes a la retrospectiva que comenten en voz alta y escriban aquellos aspectos de la iteración recién finalizada que han ido bien. Cada comentario se va a escribir en una nota adhesiva distinta para poder moverlos y reagruparlos posteriormente. Sin este paso los equipos pasarían por alto el avance conseguido en el proyecto y cómo han mejorado desde la última iteración.

2) El *Scrum master* pide a todos pensar sobre qué no ha ido tan bien durante la iteración. Los asistentes a la retrospectiva escriben en silencio cada tema en una nota por separado, y cuando todo el mundo ha terminado, se va leyendo por turnos cada una de las notas que se han escrito.

3) Hay que hacer la identificación de los temas fundamentales que han surgido durante la iteración, teniendo en cuenta tanto las cosas que han ido bien como las cosas que han ido mal. Temas recurrentes en las retrospectivas son "trabajo en equipo", "historias definidas vagamente" o "problemas con la calidad de los datos"; por poner ejemplos. El *Scrum master* creará etiquetas en una pizarra para cada uno de estos temas y pedirá a todos (incluido él mismo) que coloquen las notas adhesivas debajo del tema que mejor les pertenece (verd figura 4).

En la figura 4 se puede observar una clasificación de tres tipos de aspectos (que hay que mantener, quitar y añadir) diferenciados por tres colores distintos de notas adhesivas y agrupados en distintos temas.

Figura 4. Identificando áreas de mejora durante una retrospectiva

Fuente: Xavier Gumara

4) El *Scrum master* debe facilitar una sesión de tormenta de ideas preguntando a sus compañeros cómo creen que pueden mejorar los aspectos problemáticos detectados más urgentes. Estos temas urgentes se discuten hasta que el equipo ha generado una lista de acciones que habrá que realizar para prevenir que los problemas identificados vuelvan a surgir otra vez.

5) Finalmente, se debe dedicar unos minutos a repasar las acciones de la última reunión de retrospectiva para revisar si las acciones han sido desarrolladas correctamente durante esta iteración o hace falta volver a incluirlas en la retrospectiva de la iteración actual, con o sin alguna modificación.

2.4. Iteración especial: iteración 0

La **iteración 0** es donde el equipo se prepara para tenerlo todo listo para el desarrollo: la adquisición y configuración de recursos técnicos como servidores, licencias, la búsqueda y selección de estándares de código o la negociación de los requisitos no funcionales con los equipos de operaciones, requisitos que permitirían empezar el proyecto de forma tranquila. En esta iteración no se llega a construir nada respecto al producto.

La **iteración 1** es donde empieza el desarrollo del proyecto; la **iteración 0** es la iteración previa a la iteración 1.

En proyectos de integración de datos, la iteración 0 permite a los arquitectos de datos trabajar en el modelo de datos del proyecto. También se puede dedicar este tiempo a revisar la calidad de los orígenes de datos para empezar a definir las reglas de transformación que habrá que emplear durante la primera iteración. Debido a que la iteración 0 permite que un arquitecto de datos empiece el trabajo antes que el resto del equipo, también permite que este trabajo esté por delante del resto del equipo para el resto de las iteraciones.

3. Adaptar *Scrum* para proyectos de *business intelligence*

Escribir historias de usuario para proyectos de *business intelligence* no es una tarea fácil, ya que frecuentemente estas resultan muy grandes como para que sean entregadas en un periodo corto de tiempo. Esta dificultad hace que los métodos ágiles requieran de algunas adaptaciones para entregar con éxito proyectos de *business intelligence*.

Entre los cambios y adaptaciones que se deben tener en cuenta vamos a detallar los siguientes:

- Las historias de desarrollador como división de las historias de usuario.
- Los roles de desarrollador adecuados para un proyecto ágil de *business intelligence*.
- La entrega en cadena: la *iteración* -1 y las demostraciones en dos fases.

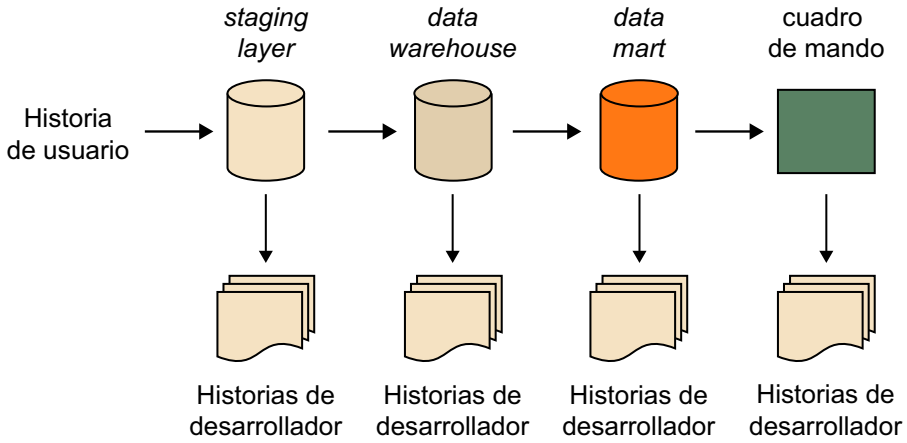
3.1. Las historias de desarrollador

Las historias de usuario en proyectos de integración de datos suelen definirse como cambios en la capa semántica de las aplicaciones de *business intelligence* (allá donde residen los metadatos y el conjunto de "universos" de datos de las aplicaciones de creación de informes y cuadros de mando), ya que es donde la propietaria del producto y los *stakeholders* del proyecto concentran el uso de las aplicaciones de *business intelligence*. Lo que probablemente desconocen es que añadir una nueva dimensión, una nueva tabla de hechos o un nuevo cuadro de mando en la capa de presentación requiere trabajo en las múltiples capas ocultas de la aplicación.

Las historias de desarrollador son el resultado de la intersección del concepto de historias de usuario de *Scrum* con las distintas capas de una arquitectura de *data warehouse*.

La forma más fácil de identificar las historias de desarrollador es, durante la sesión de la reunión de planificación, pensar en los cambios que hay que hacer en cada una de las distintas capas de la arquitectura de *data warehouse*. En cada capa, los desarrolladores deberán listar las tablas que tendrán que ser modificadas y recargadas para que la siguiente capa pueda cumplir con los nuevos requisitos (figura 5).

Figura 5. Esquema de la derivación de historias de desarrollador partiendo de una historia de usuario



Fuente: Xavier Gumara

Serán los mismos miembros del equipo los que van a decidir caso por caso si quieren escribir una historia de desarrollador por cada una de las tablas que requieren ser cargadas en una capa específica o pueden implementar y cargar varias tablas de la misma capa a la vez en una misma historia de desarrollador.

Por lo tanto, podemos resumir diciendo que una historia de desarrollador es una unidad de trabajo que:

- Puede realizarse en una iteración.
- Contribuye a entregar valor dentro de una historia de usuario.
- Es comprensible para la propietaria del producto.

Las historias de desarrollador siguen una plantilla similar a las historias de usuario:

El `<MÓDULO/CAPA>`
recibirá `<CONJUNTO DE CARACTERÍSTICAS>`
para que la propietaria del producto pueda validar los datos cargados en las `<TABLAS DESTINO>`
que complementan la `<HISTORIA DE USUARIO>`.

El <MÓDULO> o <CAPA> detalla qué parte de la arquitectura del *data warehouse* se verá afectada por la historia de desarrollador.

El <CONJUNTO DE CARACTERÍSTICAS> corresponde a la nueva habilidad que se le dará a la propietaria del producto en esta historia de desarrollador.

Las <TABLAS DESTINO> corresponde al conjunto de tablas y atributos que se van a generar en esta historia de desarrollador.

La <HISTORIA DE USUARIO> es la historia de usuario a la cual pertenece la historia de desarrollador que se está especificando.

A continuación, partiendo de una historia de usuario como la siguiente:

"**Como** analista, **quiero** poder diferenciar las compras que se hacen mediante la plataforma web de las que se hacen mediante aplicaciones móviles **para** conocer el éxito que tienen nuestras aplicaciones móviles recién lanzadas." Se ofrecen varias (no todas) historias de desarrollador válidas para la correcta consecución de la historia:

La capa de *stage* del módulo de Compras **recibirá** el atributo plataforma y dispositivo **para que la propietaria del producto pueda validar los datos cargados en** *stg_order* y *stg_order_detail* **que complementan la historia de usuario**. "Como analista, quiero poder diferenciar...".

La capa de *data warehouse* del módulo de Compras **recibirá** el conjunto completo de valores posibles de los diccionarios de plataforma y dispositivo **para que la propietaria del producto pueda validar los datos cargados en** *dim_plataform* y *dim_device* **que complementan la historia de usuario**. "Como analista, quiero poder diferenciar...".

La capa de *data warehouse* del módulo de Compras **recibirá el atributo plataforma y dispositivo para que la propietaria del producto pueda validar los datos cargados en** *dwh_order* y *dwh_order_detail* **que complementan la historia de usuario**. "Como analista, quiero poder diferenciar...".

Una vez que se han detallado todas las historias de desarrollador que forman una historia de usuario, el equipo de desarrollo está capacitado para estimar correctamente una historia de usuario y separar el trabajo que hay que realizar en iteraciones.

Historias de desarrollador en proyectos de *business intelligence*

Las historias de desarrollador en proyectos de *business intelligence* solo se necesitan para trabajos de integración de datos. Cuando se realizan cambios en la capa de presentación, se pueden seguir usando las historias de usuario ya que no requieren de trabajo en múltiples capas.

Descomponer historias de usuario en historias de desarrollador hace que las historias de usuario ya no requieran ser terminadas en una misma iteración. Tampoco es necesario que una misma historia de usuario sea lanzada completa a producción, y lanzar a producción historias de desarrollador individualmente puede ser beneficioso para el equipo porque, si hay algún error con la carga

de datos, se va a detectar antes. También hará que, en el momento de entregar la historia de usuario, este trabajo no sea tan pesado por solo tener que entregar las historias de desarrollador finales.

3.2. Nuevos roles en un proyecto ágil de *business intelligence*

El hecho de que cada tipo de trabajo a realizar en un proyecto de *business intelligence* tenga su propia herramienta hace que los desarrolladores de proyectos de *business intelligence* tiendan a especializarse considerablemente.

Herramientas para un proyecto de *business intelligence*

La definición del modelo de datos, por ejemplo, requiere conocimientos de herramientas de modelaje. Existen en el mercado numerosos paquetes de limpieza de datos para asegurar la calidad de los datos en cada capa de la arquitectura del *data warehouse*. Existen también grandes *suites* especializadas para la construcción de ETL. Y finalmente, hay herramientas específicas para la construcción de informes y cuadros de mando.

Los equipos ágiles de *data warehouse* acaban teniendo más roles específicos que los equipos ágiles en general. Eso no significa que el equipo tenga que crecer considerablemente, ya que ciertas personas pueden llegar a tener más de un rol a la vez, sino que el proyecto de *business intelligence* se va a beneficiar de repartir responsabilidades en el equipo en función de la experiencia y los conocimientos aportados por cada uno de sus miembros.

Típicamente, un equipo ágil de *business intelligence* necesitará los siguientes roles:

- Project architect (arquitecto del proyecto)
- Data architect (arquitecto de datos)
- System analyst (analista del sistema)
- System tester (comprobador del sistema)
- Proxy product owner (Representante del propietario del producto)
- *Scrum master*

Bibliografía recomendada

Ralph Hughes (2013). *Agile Data Warehousing Project Management*. Waltham, MA: Elsevier, Inc.

A continuación, detallamos una breve explicación de las nuevas obligaciones de estos roles.

3.2.1. Arquitecto del proyecto

El objetivo del arquitecto del proyecto es entregar una solución en el tiempo y presupuesto acordados.

Los *data warehouses* son aplicaciones tan complejas que el negocio debería pedir que exista una persona en la parte técnica del equipo que sea capaz de

articular, cuando sea necesario, las razones por las cuales la construcción del almacén de datos es correcta.

El arquitecto del proyecto debe:

- Tener una visión global de los esfuerzos hechos por el equipo de desarrollo. Ha de tener la habilidad y capacidad de reconducir la aplicación para dar mejor respuesta a las necesidades funcionales y de rendimiento del proyecto.

- Entender las necesidades del proyecto, el diseño de la solución propuesta y las técnicas de mantenimiento de calidad. No es necesario que el arquitecto esté 100 % implicado en cada una de las tareas, pero debe conducirlas y asegurar, durante todo el proyecto, que alguien las está llevando a cabo eficazmente.

3.2.2. Arquitecto de datos

El primer consumidor de la visión del arquitecto del proyecto es el arquitecto de datos.

Principal deber del arquitecto de datos

Debido a la complejidad de repositorios de datos con distintas arquitecturas y capas, el equipo va a necesitar diagramas lógicos y físicos para cada una de las capas. También se va a necesitar un diccionario de los datos con documentación de las tablas, columnas y relaciones que existen entre las distintas entidades que forman el *data warehouse*. Este trabajo es el principal deber del arquitecto de datos *data architect* del equipo. La combinación de capas y áreas de negocio que puede contener un almacén de datos supone un trabajo colosal para el arquitecto de datos.

El arquitecto de datos define las reglas de nomenclatura para cada una de las distintas capas y, debido a su conocimiento del modelo de datos y del contenido específico de cada una de las columnas de las capas de *data warehouse* y *data mart*, el arquitecto de datos tomará un rol importante en el análisis de la calidad de los datos.

3.2.3. Analista del sistema

El analista del sistema se encarga de mantener las transformaciones de datos entre los distintos puntos origen y destino de cada una de las capas. Esta visión le permite estandarizar, mediante patrones de diseño, los módulos ETL de la aplicación.

Otra tarea importante es la de tener una idea general de los orígenes de datos y, por lo tanto, realizar los mapeos de valores origen-destino por cada columna del *data warehouse*.

3.2.4. Comprobador del sistema

En un proyecto de *business intelligence* es esperable que todos los desarrolladores realicen pruebas unitarias y por componentes. Aun así, asegurar que todos los componentes funcionen como es esperado es una responsabilidad difícil de conseguir de forma segura cuando recae en un conjunto de personas.

Las responsabilidades del comprobador del sistema son:

- Crear pruebas de integración.
- Preparar a los desarrolladores para que hagan buenas pruebas unitarias y de integración y que validen los resultados de las mismas.
- Certificar, mediante pruebas automatizadas, que la aplicación está lista para ser lanzada a producción.
- Certificar que los datos en el almacén de datos son completos y correctos.

3.2.5. Representante del propietario del producto

El representante del propietario del producto es una persona del mundo técnico que entiende suficientemente el negocio para asegurar que el propietario del producto está desempeñando correctamente su trabajo y que es capaz de proveer al equipo ce la información que falta y las decisiones que este necesita para seguir adelante.

4. Monitorización de proyectos *Scrum*

La gestión ágil de proyectos de *business intelligence* requiere pocos artefactos de monitorización. En concreto, son una pizarra de tareas, los gráficos *burn down* para la monitorización de iteraciones individuales y el gráfico *burn up* para la monitorización del proyecto entero. En este apartado vamos a hablar de estos tres conceptos.

4.1. La pizarra de tareas

La pizarra de tareas permite a todo el mundo ver el progreso del equipo en lo que respecta a las tareas definidas durante la reunión de planificación.

En su forma más simple, la pizarra es un gran mural localizado en una de las paredes de la oficina, cerca de donde trabaja el equipo. La base del mural es una tabla con cinco columnas. La primera servirá para colocar las tarjetas adhesivas de las historias de usuario, que no se moverán de esta columna en toda la iteración. Las siguientes tendrán los títulos "Por hacer", "En desarrollo", "Pendiente de validar" y "Hecho" (figura 6).

Figura 6. Composición básica de una pizarra de tareas

Fuente: Xavier Gumara

Después de la reunión de planificación, todas las historias de usuario o desarrollador que hayan entrado en el *sprint* se ordenan en la columna de la izquierda por orden de prioridad. Asimismo, todas las tarjetas adhesivas que corresponden a las tareas de las historias de usuario de esta iteración se van a colocar bajo la columna "Por hacer", ya que todavía no se ha empezado el desarrollo de ninguna historia de usuario (figura 7).

Figura 7. Composición de la pizarra después de la reunión de planificación

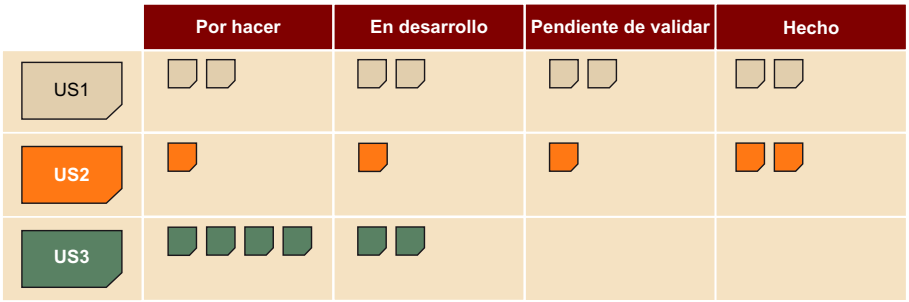
Fuente: Xavier Gumara

A medida que avanza la iteración, los desarrolladores irán cogiendo tareas de la columna "Por hacer", una cada vez, y las colocarán en la columna "En desarrollo". Normalmente, se va a escribir el nombre de la persona que está trabajando en la tarea para que el resto del equipo y también los *stakeholders* puedan dirigirse a esa persona si tienen alguna duda sobre la implementación.

Una vez que se considere que la tarea está finalizada, el desarrollador colocará la nota adhesiva de la tarea en la columna "Pendiente de validar", y entonces cualquier otro desarrollador del equipo recogerá la tarea y validará que realmente cumple con las especificaciones de la misma. Si es así, se moverá la nota adhesiva de la tarea a la columna "Hecho".

Cuando todas las notas adhesivas de una historia de usuario se han colocado en la columna "Hecho", significa que la historia de usuario está ya terminada y es candidata a ser subida a producción en la siguiente *release*. Anteriormente, la propietaria del producto habrá validado la historia para dar su conformidad. En caso de que no cumpla con las especificaciones, las tareas que haya que revisar volverán a la columna "Por hacer" para que sean recogidas por un desarrollador otra vez. En la figura 8 se puede observar el posible estado de una pizarra de tareas a mitad de una iteración.

Figura 8. Estado de una pizarra de tareas a mitad de una iteración



Fuente: Xavier Gumara

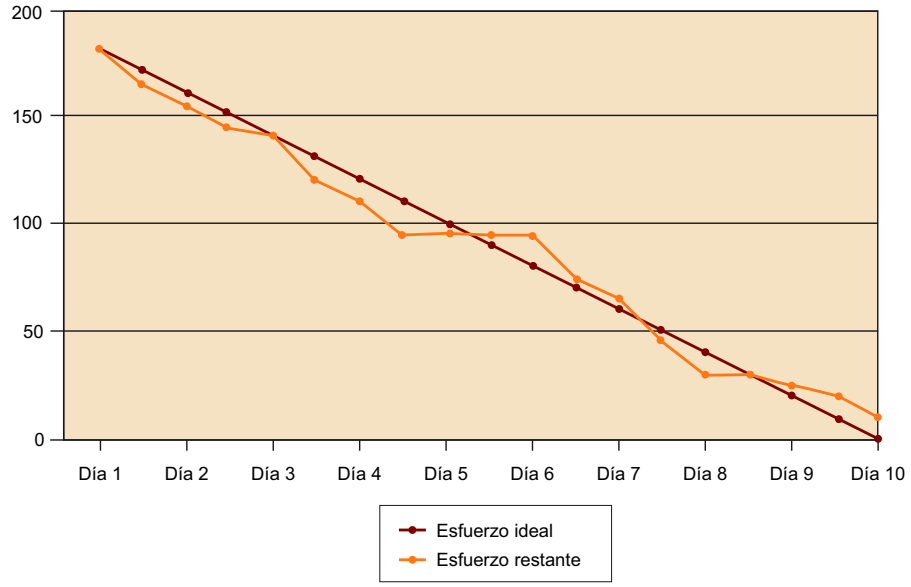
El objetivo de la iteración es que todas las tareas lleguen a la columna "Hecho"; esto significa que se ha cumplido con las estimaciones dadas durante la reunión de planificación.

4.2. Gráfico *burn down*

Otro instrumento de monitorización de las iteraciones son los gráficos *burn down*. Estos gráficos ayudan al *Scrum master* y a la propietaria del producto a conocer el avance de la iteración en puntos de historia.

En el eje horizontal (eje X) se muestra el tiempo, y dicho el eje alcanzará la duración de la iteración en días. El eje vertical (eje Y) se destina a los puntos de historia (figura 9).

Figura 9. Gráfico *burn down* para una iteración de 10 días



Fuente: Xavier Gumara

Cuando durante la reunión de planificación ya se ha acordado el número de puntos de historia que será posible entregar en la iteración (teniendo en cuenta vacaciones, reuniones, etc., del equipo), el *Scrum master* puede dibujar la línea de esfuerzo ideal. Esta línea se traza desde la intersección del eje Y con el número de puntos totales que se han de entregar en la iteración hasta la intersección del eje X con el último día de la iteración. Suponiendo un trabajo constante del equipo durante toda la iteración, esta línea representa el esfuerzo que el equipo deberá realizar (en número de puntos "quemados", de aquí el nombre de *burn down*) para conseguir entregar todas las historias de usuario y tareas al final de la iteración.

Hay que tener en cuenta que en cada iteración el número de puntos de historia será distinto, ya que depende de la capacidad del equipo que en ese momento se disponga. El objetivo del gráfico no es el de comparar el número de puntos de historia estimados que se han de entregar en una iteración u otra, sino el de monitorizar el avance de la iteración.

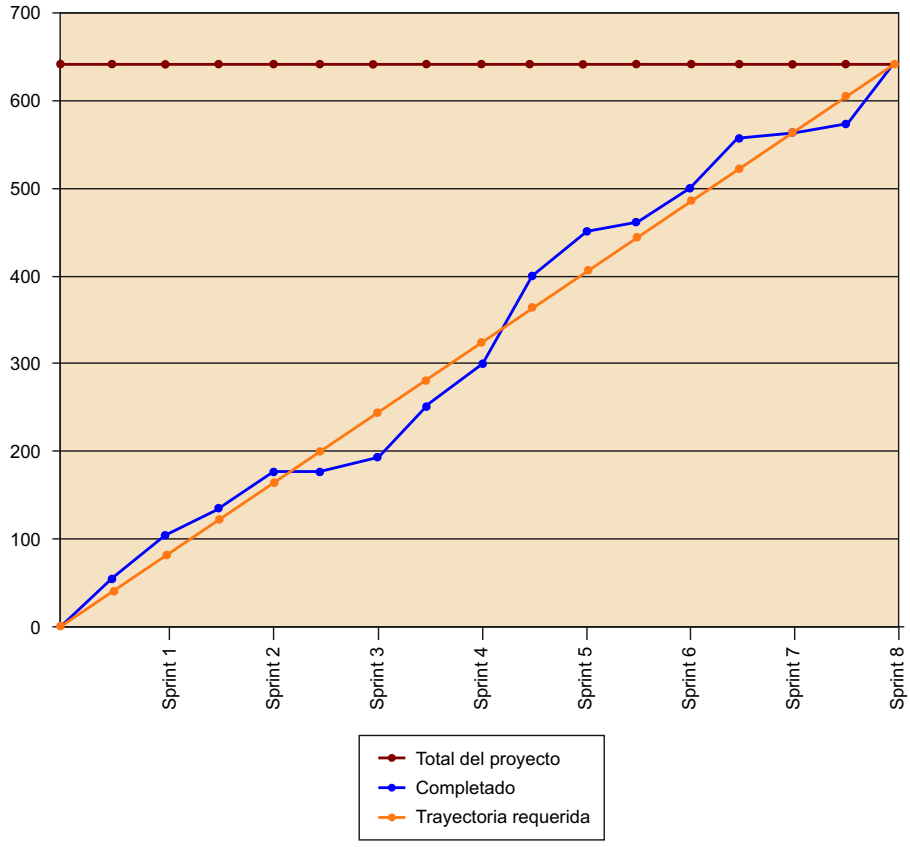
Con este funcionamiento, los gráficos *burn down* dan una buena visión del estado de la iteración y pueden ayudar a identificar varios problemas en el equipo, la planificación y el desarrollo de la iteración.

La primera función, y la más clara, de un gráfico *burn down* consiste en saber si el equipo está entregando ("quemando" puntos) de acuerdo con su velocidad.

4.3. Gráfico *burn up*

Si el gráfico *burn down* sirve para controlar el desarrollo de una interacción, el gráfico *burn up* sirve para visualizar la evolución del proyecto entero. Este tipo de gráfico se utiliza cuando el equipo ha especificado la totalidad de las historias de usuario que forman el proyecto y, por lo tanto, se tiene una idea del número de puntos de historia totales que se van a necesitar para desarrollar el proyecto.

Figura 10. Ejemplo de un gráfico *burn up*



Fuente: Xavier Gumara

En el eje horizontal (eje X) se colocan las iteraciones, y en el eje vertical (eje Y) se colocan los puntos de historia. Para obtener la versión inicial del gráfico, se debe trazar una línea entre el punto que cruza los puntos de historia 0 y la iteración 0, y el punto que cruza los puntos de historia totales del proyecto y

la última iteración. Esta línea se corresponde con la "trayectoria requerida" en el gráfico de la figura 10.

Por otra parte, se debe trazar otra línea, paralela al eje horizontal y a la altura del número de puntos de historia totales del proyecto. Esta línea se corresponde con la línea "total del proyecto" de la figura 10.

Al final de cada iteración, el *Scrum master* debe actualizar la línea correspondiente al esfuerzo completado marcando el número de puntos de historia que se han logrado entregar en la iteración que acaba de finalizar. Una línea de trabajo completado por encima de la trayectoria requerida significa que se está entregando valor por encima de lo estimado. Una línea de trabajo completado por debajo significa que el proyecto se está retrasando respecto al esfuerzo estimado.

Para la creación de un gráfico *burn up*, es necesario que todas las historias de usuario sean definidas al principio del proyecto y que sean estimadas por el equipo desarrollador. Por distintos motivos, es probable que no en todos los proyectos se pueda realizar esta actividad.

5. Enunciado del caso práctico: comercio electrónico

Una empresa consolidada del sector del comercio electrónico en España dispone desde hace dos años de un almacén de datos formado por una arquitectura de tres capas: el *staging area*, el *data warehouse* y varios *data marts*. Actualmente, se almacena la información sobre los clientes, los productos y el historial de ventas de la tienda electrónica, pero se tiene poca visión del volumen de clientes y ventas en comparación con la población de cada región.

El gestor del área de *business intelligence* tiene a su cargo un equipo de cuatro desarrolladores de *business intelligence*. En otras partes de la compañía, se están empezando a utilizar métodos ágiles para la gestión de proyectos y se le pide al gestor del área de *business intelligence* que los aplique también para la construcción de una nueva parte del *data warehouse*, dedicada a la ampliación del mismo con datos del *reach* de la tienda en línea. También se quiere conocer el *reach* en función de la población usuaria de Internet en España. Esta información se quiere conocer por ubicación del usuario, género y edad, y también cómo ha evolucionado a lo largo de los últimos dos años y mensualmente.

El objetivo del proyecto es ampliar el *data warehouse* y el *data mart* con las dimensiones y hechos necesarios para dar respuesta a la petición anterior, más la construcción de un cuadro de mando donde poder visualizar la métrica *reach* por comunidad autónoma, género, edad y mes del año.

Definición de *reach*

Definiremos *reach* como el porcentaje de usuarios que compran en nuestra tienda en línea respecto a la población total.

El equipo de desarrollo está formado por cuatro personas. Durante los últimos dos años, cada desarrollador del equipo ha ido evolucionando hacia un perfil concreto:

- Un desarrollador ETL con experiencia en arquitectura de proyectos de *business intelligence*.
- Un desarrollador ETL y de cuadros de mando con experiencia en arquitectura de datos.
- Dos desarrolladores de ETL y cuadros de mando.

En la realización del caso práctico, deberéis aplicar la metodología *Scrum* y tendrás asignado el rol de propietario del producto.

5.1. Ejercicios

5.1.1. Ejercicio 1A. Planificación inicial

Después de haber leído las necesidades de información externa de la empresa, describid las historias de usuario y las historias de desarrollador que definen

la ampliación del *data warehouse* y ordenadlas por prioridad y dependencias para formar la cartera del producto inicial del proyecto. En caso de necesitar un *sprint* 0, detallad las historias de usuario que formarían parte de este.

Detallad en qué iteraciones va a ser necesaria la dedicación de cada uno de los roles del proyecto. ¿En qué se podría especializar alguno de los dos desarrolladores puros?

El director de la compañía está especialmente interesado en conocer los riesgos de la gestión ágil de proyectos en el ámbito de *business intelligence*. Durante la entrega de la cartera, hay que preparar una presentación que los explique y que incluya propuestas para mitigarlos.

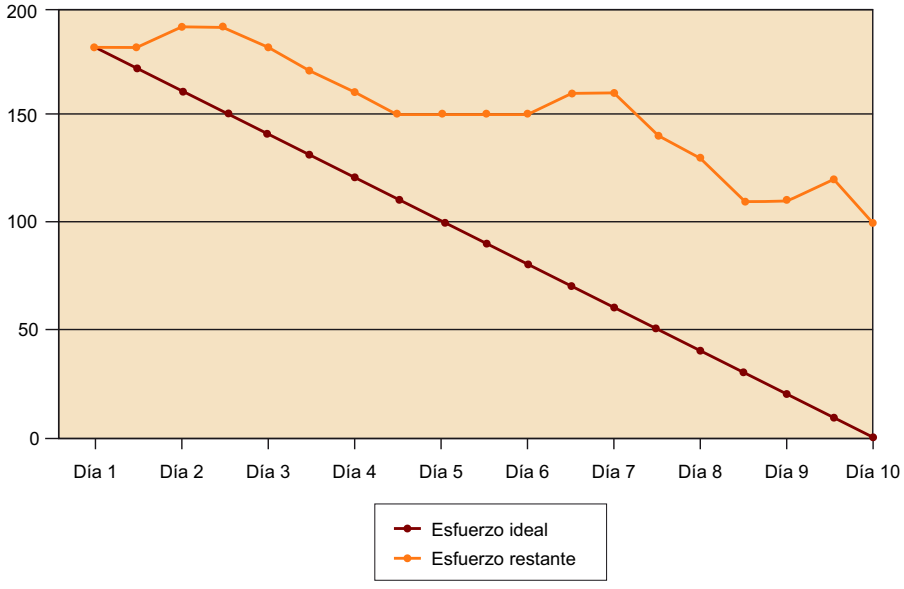
5.1.2. Ejercicio 1B. La primera cartera de la iteración

Teniendo en cuenta la cartera del producto del apartado anterior, ¿qué esfuerzo de trabajo debería considerar el equipo como el equivalente a un punto de historia? Teniendo en cuenta esta estimación, asignad los puntos de historia necesarios a todas las historias de usuario citadas en el apartado anterior.

5.1.3. Ejercicio 2A. Primera reunión de retrospectiva

Durante la primera iteración se ha calculado el gráfico *burn down* de la figura 11.

Figura 11. Gráfico *burn down*



Fuente: Xavier Gumara

Figura 11

En el gráfico se incluye la información tanto a mitad como a final del día, y los puntos de historia totales de la iteración son inventados y no guardan relación con los que se piden en el ejercicio 1.

Durante la reunión de retrospectiva al final de la primera iteración, como propietarios del producto, haced un diagnóstico de lo ocurrido durante la iteración e identificad las acciones que se habrán de realizar y el rol responsable de realizarlas.

5.1.4. Ejercicio 2B. Segunda reunión de retrospectiva

Durante la reunión de retrospectiva de la segunda iteración, se recogen los siguientes comentarios del equipo de desarrollo (no se incluyen todos):

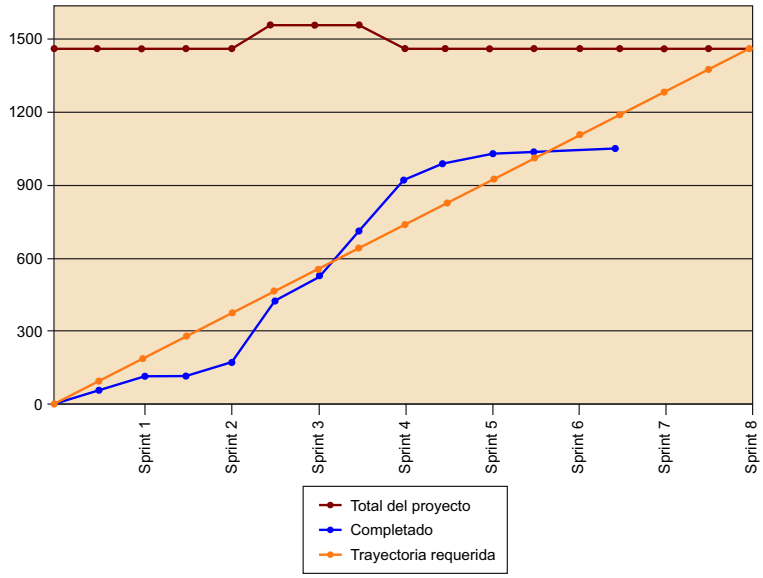
- solo el 90 % de los usuarios de nuestra base de datos tiene informado el género.
- El proyecto es motivante y aportará valor a la empresa.
- Los *data marts* no contienen datos en todo momento, ya que se borran enteros durante la carga.
- Se han añadido tareas a la iteración sin pasar por la cartera.
- Se ha tardado mucho en terminar pocas tareas.
- Algunas tareas entraron en la iteración sin estar bien definidas.
- La demostración fue muy caótica y con problemas.

Definid cuáles son las acciones que habrá que realizar y cuándo, además de quién es el responsable de que estas acciones se lleven a cabo. Justificad las respuestas.

5.1.5. Ejercicio 3. Gráfico *burn up*

A dos iteraciones de la entrega del proyecto, el director de la compañía pide reunirse con vosotros para revisar el cierre del proyecto. Como propietario del producto, le vais a presentar el gráfico *burn up* hasta el momento de la figura 12.

Figura 12. Gráfico *burn up*



Fuente: Xavier Gumara

Figura 12

En el gráfico, además de los puntos completados al final de la iteración, también se incluyen los puntos completados a la mitad de cada iteración. El total de puntos de historia del proyecto y el número de iteraciones es orientativo y puede no guardar relación con la respuesta dada en el ejercicio 1.

¿Está el proyecto en riesgo? ¿Qué puntos deberán ser expuestos al director? Recordad que debéis hacer énfasis en todas las fases del proyecto, no solo en los *sprints* finales, para justificar vuestro discurso.

Resumen

En este módulo didáctico, hemos presentado los conceptos generales de la gestión ágil de los proyectos de *business intelligence*; una de las metodologías ágiles más usadas, como es *Scrum*; la adaptación de *Scrum* en proyectos de *business intelligence*, y finalmente, las herramientas usadas para la monitorización en los proyectos *Scrum*.

Bibliografía

Collier, Ken (2012). *Agile Analytics*. Boston, MA: Pearson Education, Inc.

Corr, Lawrence; Stagnitto, Jim (2012). *Agile Data Warehouse Design*. Leeds, Reino Unido: DecisionOne Press, Burwood House

Eckerson, Wayne (2012). *Secrets of Analytical Leaders: Insights from Information Insiders*. Denville, NJ: Technics Publications.

Hughes, Ralph (2013). *Agile Data Warehousing Project Management*. Waltham, MA: Elsevier, Inc.

Hughes, Ralph (2016). *Agile Data Warehousing for the Enterprise*. Waltham, MA: Elsevier, Inc.