



Metodologías Ágiles

Metodologías ágiles

Por Nicolás López (TecLab)

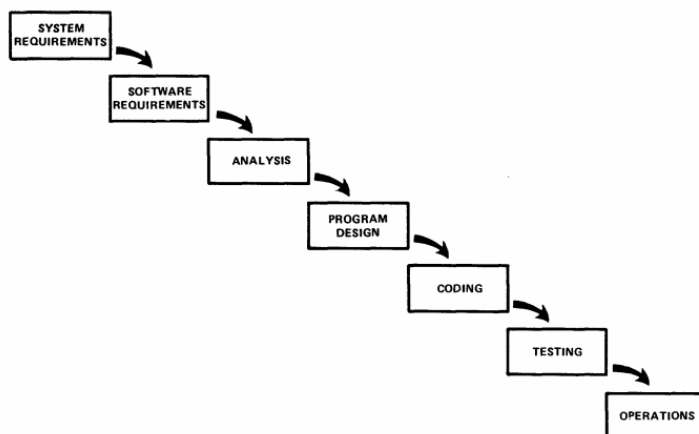
Historia

Antes de seguir avanzando, vamos a tomarnos un tiempo para explorar una manera distinta de trabajar y organizarse. ¿Escucharon hablar de las metodologías ágiles?

El enfoque o marco de trabajo ágil nace desde el desarrollo de *software* para hacer frente a las metodologías clásicas, en la cuales los proyectos eran planificados en fases lineales y secuenciales de tipo cascada y gestionados con métodos como Gantt, PERT, entre otros, que decantaron en exhaustivos controles sobre las tareas y los procesos.

En 1970, el Dr. Winston W. Royce presenta un modelo secuencial para el desarrollo de *software*. Más adelante, a este modelo se lo conocerá como “modelo en cascada” o “*waterfall*” y se convertirá en la metodología más utilizada en la industria.

Figura 1: Modelo original de fases secuenciales propuesto por Winston W. Royce



Fuente: Royce, 1970, <https://bit.ly/2ZGzd0l>

Este proceso de fases sugiere una evolución lineal y secuencial, con un orden que no se puede modificar. El proceso consiste en las siguientes fases:

- Especificación y análisis de los requerimientos
- Diseño
- Construcción o codificación
- Integración
- *Testing*
- Instalación y operación
- Mantenimiento

El proceso comienza con una fase de análisis que incluye un estudio de viabilidad y una definición de los requisitos. En el **estudio de viabilidad** se evalúan los costos, la rentabilidad y la factibilidad del proyecto de *software*.

A continuación, se realiza una **definición detallada de los requisitos**, incluyendo un análisis de la situación de salida y un concepto.

Por último, se incluye un análisis de la definición de los requisitos en el que los problemas complejos se dividen en pequeñas tareas secundarias y se elaboran las correspondientes estrategias de resolución.

En la fase de diseño se formula una solución específica en base a las exigencias, tareas y estrategias definidas. En esta fase, los desarrolladores de software se encargan de diseñar la **arquitectura de software** (...) centrándose en componentes concretos, como interfaces, entornos de trabajo o bibliotecas. La fase de diseño da como resultado un borrador preliminar con el plan de diseño del software.

(...)

La arquitectura definida en la fase de diseño, se ejecuta en la **fase de implementación**, en la que se incluye la **programación del software**, la **búsqueda de errores** y las **pruebas unitarias**. En la fase de implementación, el proyecto de software se traduce al correspondiente lenguaje de programación. Los diversos componentes se desarrollan por separado, se comprueban a través de las pruebas unitarias y se integran poco a poco en el producto final.

(...)

La fase de prueba incluye la integración del software en el entorno (...) Los productos de software se envían en primer lugar a los usuarios [o las usuarias] finales seleccionados en **versión beta**. Las **pruebas de aceptación** desarrolladas en la fase de análisis permiten determinar si el software cumple con las exigencias definidas con anterioridad.

(...)

Una vez que la fase de prueba ha concluido con éxito, se autoriza la **aplicación productiva** del software. La última fase del modelo en cascada incluye la **entrega**, el **mantenimiento** y la **mejora del software**. (IONOS, 2019, <https://bit.ly/3e9zvSO>)

Los mayores inconvenientes de la aplicación en el *software* de modelos de tipo *waterfall* se centran en el contexto cambiante de la industria y en el proceso mismo de desarrollo,

donde el resultado depende de las habilidades, talentos y actitud de las personas, más que de procesos ejecutados.

Con el avance de la globalización e internet, los sistemas pasaron de ser estables a transformarse en un entorno volátil, donde los requerimientos definidos en un comienzo rara vez son válidos meses más tarde. En este nuevo contexto, las metodologías *waterfall* resultaron muy “pesadas” para responder satisfactoriamente a los cambios de negocio.

Esto significó que muchos proyectos terminaran siendo cancelados a medio camino, y muchos de los que se completaron no satisfacían todas las necesidades actuales de la empresa, incluso si se cumplían los objetivos originales del proyecto.

Surgimiento de metodologías ágiles

En los años 90 surgieron varios movimientos identificados como Metodologías Livianas (*Lightweight Methodologies*). Entre estos se encuentran Extreme Programming (XP), Scrum, Pragmatic Programming, Lean Software Development, etc.

Más tarde, en febrero de 2001, se reunieron en Utah (EE. UU.) diecisiete líderes reconocidos del desarrollo de software (...) con el objetivo de determinar los valores y principios que les permitirían a los equipos desarrollar software de forma más acertada con las necesidades del cliente y responder mejor a los cambios que pudieran surgir a lo largo de un proyecto de desarrollo. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por la rigidez y dominados por la documentación.

En esta reunión se creó la Agile Alliance, una organización sin fines de lucro cuyo objetivo es el de promover los valores y principios de la filosofía ágil y ayudar a las organizaciones en su adopción. También se declaró la piedra angular del movimiento ágil, conocida como Manifiesto Ágil (Alaimo y Salías, 2015, p. 7).

Manifiesto ágil

El manifiesto ágil es una declaración de intención y está compuesto por cuatro valores y doce principios.

Estamos descubriendo mejores maneras de desarrollar software haciéndolo y ayudando a otros a hacerlo. Este trabajo nos ha llevado a valorar:

Individuos e interacciones sobre procesos y herramientas

Software funcionando sobre documentación extensiva

Colaboración con el cliente sobre negociación contractual

Respuesta ante el cambio sobre seguir un plan. (Beck, 2001, <https://bit.ly/3f7eBFk>)

Valores:

- **Personas e interacciones** sobre procesos y herramientas
 - Las organizaciones están compuestas por personas, por lo cual es necesario que estas tengan actitud, talento, compromiso y se comuniquen e interactúen de manera eficiente y eficaz. “Es más importante construir un buen equipo que construir el contexto” (Alaimo y Salías, 2015, p. 8).
- **Software en funcionamiento** sobre documentación exhaustiva
 - Un producto o servicio que funciona es una de las grandes diferencias que aporta el desarrollo ágil. La documentación es el resultado intermedio y “su finalidad no es dar valor en forma directa al cliente [o clienta]” (Alaimo y Salías, 2015, p. 8).
- **Colaboración con la clientela** sobre negociación de contratos
 - Conseguir la implicación del cliente o la clienta en el proceso de desarrollo de “producto o servicio” es esencial para el éxito del proyecto.
- **Respuesta ante el cambio** sobre seguimiento de un plan
 - Poder responder a los cambios que surgen a lo largo del proyecto determina su éxito o fracaso. “Por lo tanto, la planificación no debe ser estricta sino flexible” (Alaimo y Salías, 2015, p. 8).

“Esto es, aunque los elementos a la derecha tienen valor, nosotros valoramos más los que están a la izquierda” (Beck, 2001, <https://bit.ly/3f7eBFk>).

Principios del desarrollo ágil:

Además de los cuatro valores del manifiesto, existen doce principios que lo completan y que nos ayudan a entender aún más la filosofía de la agilidad. Alaimo y Salías (2015) nos hablan de estos principios:

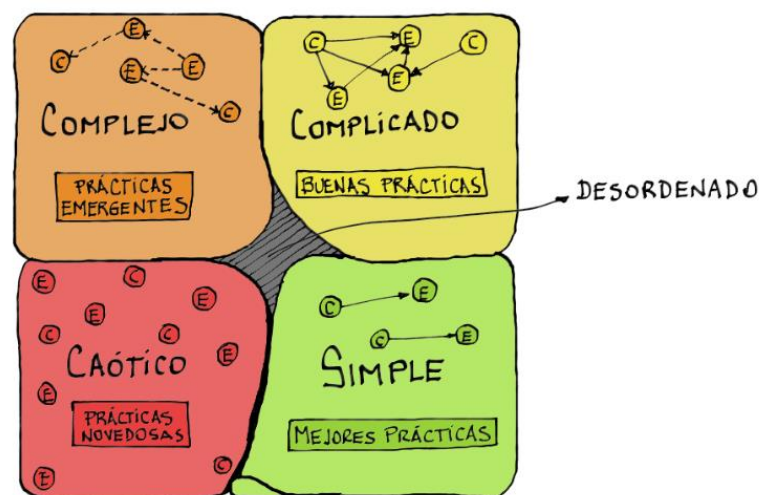
- La prioridad fundamental consiste en satisfacer a la clientela entregando de manera pronta y continua *software* que aporte valor.
 - Admitimos cambios en los requisitos, incluso aquellos casos en los que llegan tarde al desarrollo. Los procesos ágiles sacan provecho de estos cambios y se usan para la ventaja competitiva de la clientela.
 - A menudo, hacemos entregas de *software* que funcione, desde algunas semanas hasta algunos meses, aunque preferimos la escala de tiempo más corta.
 - La gente de negocio y quienes son profesionales del desarrollo deben trabajar en conjunto todos los días durante todo el proyecto.
 - Construimos proyectos con personas que tienen motivación. Les brindamos los entornos y el apoyo necesarios, y confiamos en que harán bien el trabajo.
 - Creemos que para llevar información a un Equipo de Desarrollo y hacerla circular dentro de este, la conversación cara a cara es la forma más eficiente y efectiva de hacerlo.
 - Pensamos que la primordial medida de progreso es el *software*.
 - Los procesos ágiles promueven el desarrollo sostenido. Es necesario que especialistas en promoción, especialistas en desarrollo y usuarios y usuarias puedan mantener un ritmo constante de manera indeterminada.
 - Atender en todo momento el buen diseño y la excelencia técnica mejora la agilidad.
 - La simplicidad, es decir, el arte de lograr la mayor cantidad de trabajo que no se hace, es esencial.
 - Los mejores diseños, arquitecturas y requisitos son el resultado de equipos autoorganizados.
- En intervalos que se dan de manera regular, el equipo debe reflexionar y buscar la manera de ser más efectivo, y ajustar su comportamiento a esto.

Marco Cynefin

¿En qué entorno Scrum nos es más útil?

A medida que nos adentramos en el mundo de la agilidad, se hace más importante conocer y analizar el contexto en el que nos hallamos, para poder entender cuál es la diferencia entre este y otros, y cómo debemos actuar en cada uno de ellos. Para esto, vamos a introducirnos en un marco creado por David Snowden, llamado “Cynefin”. Su nombre viene del galés, y si bien no tiene una traducción exacta al castellano, la más cercana sería “hábitat”. Cynefin es un *framework* que representa las cinco situaciones en las que una organización puede encontrarse y la manera en que deberíamos actuar en cada una de ellas. A continuación, analizaremos cada una.

Figura 2: Marco Cynefin



Fuente: Alaimo y Salías, 2015, p. 16

Dominio simple u obvio

En este escenario todo el equipo conoce cómo hacer y resolver las cosas, casi no existe incertidumbre y se conoce todo el proceso. Su estructura es fácil de comprender, y el comportamiento de los elementos es muy predecible. Aquí existen las mejores prácticas, y las soluciones son conocidas para problemas conocidos. Los procesos en este dominio “especifican una serie lógica de pasos y se ejecutan de manera repetitiva” (Alaimo y Salías, 2015, p. 16).

Dominio complicado

En este dominio encontramos problemas complicados, que no hay que confundir con los complejos. Estos problemas tienen una relación directa de causa y efecto, y cuentan con cantidades de variables y buenas prácticas, pero se requiere de personas expertas para poder identificar estas prácticas y llegar a una solución.

Dominio complejo

Cuando estamos inmersos en escenarios complejos, la estructura no es fácil de entender para el equipo. Muchas cosas no se han realizado antes, la incertidumbre es alta, los resultados son impredecibles, y no existen buenas prácticas ni personas expertas que puedan apoyar a quienes se encuentran en esta situación. Frente a esto, “solo podemos examinar los resultados y adaptarnos” (Alaimo y Salías, 2015, p. 16). Este es el dominio de las prácticas emergentes y donde mejor funcionan las metodologías ágiles, dado que generalmente en estos escenarios se requiere de experimentación, innovación, altos niveles de comunicación e interacción. En este contexto Scrum se utiliza para operar, inspeccionar y adaptar las prácticas emergentes de un equipo.

Dominio caótico

Los escenarios caóticos requieren de una rápida respuesta. No existe la organización y reina el desconcierto. Estamos en emergencia, en crisis, hay muchas incógnitas, y necesitamos actuar de inmediato para restituir el control, establecer un orden y sacar la situación del caos. “Este es el dominio de la improvisación” (Alaimo y Salías, 2015, p. 17).

Dominio desordenado

Este escenario es sumamente peligroso, puesto que no sabemos en qué dominio estamos, hacia dónde tenemos que avanzar ni cómo debemos actuar. De hecho, en la mayoría de los casos, las personas actúan de acuerdo con su interpretación y modelos conocidos. Si nos encontramos en este escenario, debemos usar todos nuestros recursos para identificar el dominio correcto y movernos de inmediato hacia él, para actuar de manera afín.

Figura 3: Marco Scrum



Fuente: [imagen sin título sobre marco Scrum], 2020, <https://bit.ly/2O2CWJE>

Nacimiento de Scrum

El concepto de Scrum tiene su origen en un *paper* escrito por Hirotaka Takeuchi y Ikujiro Nonaka en 1986, titulado “The New Product Development Game”. Aquí, los autores hablan acerca de los nuevos procesos de desarrollo que se utilizan en productos que gozan de éxito en Estados Unidos y Japón. En este trabajo de investigación comenzaron a aparecer los primeros conceptos de equipos multidisciplinarios, procesos iterativos e incrementales en la construcción de nuevos productos. Además, se comparó la nueva forma de trabajo en equipo con el avance en formación de *scrum* de los jugadores de *rugby*; de ahí, el término “*scrum*” para referirse a esta forma de trabajo.

En el año 1995, **Ken Schwaber** presentó “Scrum Development Process”. Se trataba de un marco de reglas para desarrollar *software*, que tenía sus bases en los principios de Scrum. El mismo Schwaber había empleado este marco en el desarrollo de Delphi, y luego **Jeff Sutherland** lo aplicó en su empresa Easel Corporation. Aquí es cuando nace formalmente el marco Scrum.

¿Qué es Scrum?

Según la guía Scrum creada por Ken Schwaber y Jeff Sutherland (2016), es un marco de trabajo por el cual las personas pueden abordar problemas complejos adaptativos y al mismo tiempo entregar productos con el máximo valor posible, productiva y creativamente.

Para Alaimo y Salías (2015) Scrum se define de la siguiente manera:

un marco de trabajo que nos permite encontrar prácticas emergentes en dominios complejos. No es (...) una metodología, [dado que] en lugar de proporcionar una descripción detallada de cómo deben realizarse las tareas de un proyecto/producto, genera un contexto (...) iterativo, de transparencia, inspección y adaptación para que los involucrados [e involucradas] vayan ajustando y mejorando su propio proceso. Esto se debe a que no existen buenas prácticas en contextos complejos. Es el equipo (...) quien encontrará la mejor manera de resolver sus problemáticas (p. 2).

En Scrum podemos identificar a un equipo denominado “Equipo Scrum”, que está compuesto por el o la *product owner*, el o la *Scrum master* y el Equipo de Desarrollo.

En lo que refiere al rol de *Scrum master*, según Alaimo y Salías (2015), esta persona se responsabiliza de que el Equipo Scrum comprenda el marco Scrum y lo ponga en práctica. Además, hace de líder y de coach, pues acompaña al equipo y a las personas externas con el objetivo de producir interacciones que maximicen el valor entregado.

Alaimo y Salías (2015) también definen los otros roles:

El [o la] *product owner* representa al negocio, clientes, *stakeholders*, y usuarios [y usuarias] finales. Tiene su foco en maximizar el valor del producto y del trabajo del Equipo de Desarrollo.

El progreso de los proyectos Scrum se realiza (...) en iteraciones llamadas *sprints*. Estos *sprints* tienen una duración fija, preestablecida no mayor que un mes. Al comienzo de cada *sprint*, el Equipo de Desarrollo realiza un compromiso de entrega de una serie de funcionalidades o características del producto.

Al finalizar el *sprint*, se espera que estas características comprometidas estén terminadas, [y se genera] un incremento de producto. En este momento es cuando (...) el Equipo Scrum y *stakeholders* hacen una reunión de revisión del producto construido durante el *sprint*. [Las oportunidades] de mejoras obtenidas en esta reunión pueden ser incluidas en futuros *sprints*. (Alaimo y Salías, 2015, p. 18)

Valores Scrum

En julio de 2016, los valores de Scrum se agregaron oficialmente a *La Guía de Scrum*. Estos valores incluyen coraje, foco, compromiso, respeto y apertura, y son la base sobre la que se construye Scrum:

Cuando el Equipo Scrum encarna y vivencia los valores de compromiso, coraje, enfoque, apertura y respeto, los pilares de transparencia,

inspección y adaptación de Scrum cobran vida y crean confianza para todos. Los miembros del Equipo Scrum aprenden y exploran esos valores mientras trabajan con los eventos, roles y artefactos de Scrum. (Schwaber y Sutherland, 2016, <https://bit.ly/3gwCoif>)

A continuación, describiremos cada uno de ellos:

Compromiso

Cada una de las personas que forme parte del Equipo Scrum hará el máximo esfuerzo posible y será completamente transparente sobre el progreso del *sprint*, es decir, se comprometerá con los objetivos del equipo. “Compromiso” significa dedicación, y se refiere a las acciones y el esfuerzo, no al resultado final.

Foco

Quienes formen parte del equipo, sin excepción, deben enfocarse en el trabajo planificado en cada *sprint* (ítems del *sprint backlog*), dado que esto permite cumplir los objetivos propuestos de dicho *sprint*.

Respeto

“Los [y las] miembros de un Equipo Scrum respetan el conocimiento y la experiencia profesional no solo del resto (...) del equipo, sino también de aquellas personas con las que se relacionan” (CertMind, 2019, <https://bit.ly/3guL4FC>).

Coraje

El equipo debe tener coraje para hacer lo correcto a pesar de que esto implique deshacer caminos ya recorridos. El coraje es necesario para resolver los impedimentos que puedan surgir y para mejorar la aplicación de Scrum, identificando oportunidades de mejora.

Apertura

En los equipos “se privilegia la transparencia y la discusión abierta de los problemas” (Alaimo y Salías, 2015, p. 2). Por otra parte, el equipo debe estar abierto a interactuar otros equipos o *stakeholders*, y debe tener la apertura para aprender nuevas habilidades o adquirir nuevos conocimientos que lo convierta en personas multifuncionales.

Pilares Scrum

Los pilares de Scrum están basados en el empirismo, el cual consiste en aprender haciendo y mejorar sobre la base de nuestras experiencias. Es por esto que Scrum emplea un enfoque iterativo e incremental, que permite mejorar la previsibilidad y el control del riesgo. Para esto, siguiendo a Schwaber y Sutherland (2016), Scrum se basa en tres pilares, que detallaremos a continuación:

Transparencia

Implica dar visibilidad a todo lo que sucede, “ya que los aspectos significativos del proceso deben ser visibles para [aquellas personas] responsables del resultado” (Schwaber y Sutherland, 2016, <https://bit.ly/3gwCoif>).

Inspección

Se debe realizar una inspección frecuente de los artefactos y del progreso para identificar y corregir las variaciones indeseadas. En otras palabras, revisamos el proceso de trabajo y el producto, y en caso de ser necesario, corregimos.

Adaptación

Consiste en realizar los ajustes necesarios en los procesos y en el producto para minimizar la desviación y maximizar la entrega de valor.

Roles dentro de Scrum

Figura 4: Roles dentro de Scrum



Fuente: [imagen sin título sobre roles dentro de Scrum], 2019, <https://bit.ly/2O28c27>

Los Equipos Scrum cuentan con tres roles fundamentales. Estos son *product owner*, *Scrum master* y el Equipo de Desarrollo.

Product owner:

Como mencionamos anteriormente, quien sea *product owner* es “responsable de maximizar el valor del producto y del trabajo del Equipo de Desarrollo” (Alaimo y Salías, 2015, p. 18).

Es responsable de la gestión y administración del *product backlog*. De acuerdo con Alaimo y Salías (2015), esto incluye lo siguiente:

- Cerciorarse de que los elementos del *product backlog* estén expresados claramente.
- Asegurar la visibilidad, transparencia y comprensión de todos los elementos del *product backlog* por parte del equipo.
- Ordenar los ítems del *product backlog* para lograr los objetivos y maximizar la entrega de valor.
- Asegurar que el Equipo de Desarrollo entiende los ítems del *product backlog* tanto como sea necesario (Alaimo y Salías, 2015).

Es cierto que quien cumpla el papel de *product owner* puede delegar este trabajo al Equipo de Desarrollo, pero la responsabilidad será siempre suya.

Otras actividades habituales que le corresponden a este rol son las siguientes:

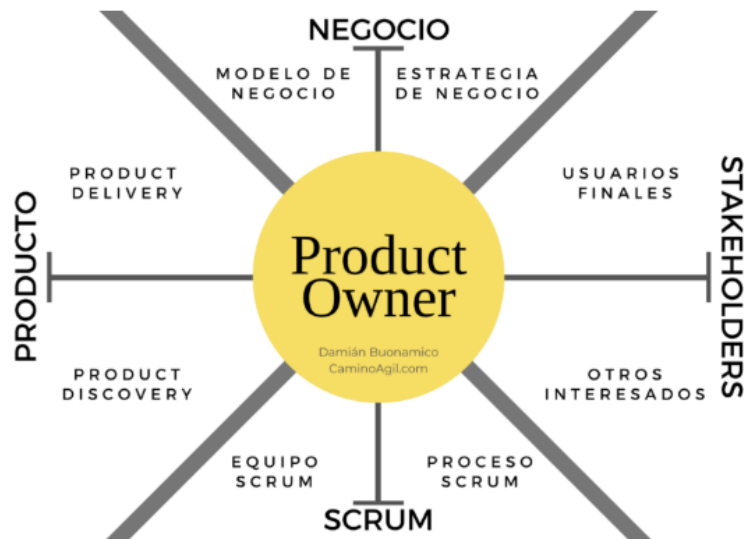
- cocrear la visión del producto;
- gestionar las expectativas de los y las *stakeholders*;
- determinar las prioridades de las características del producto;
- generar un plan de entregas (*release plan*);
- maximizar la rentabilidad del producto; y
- cambiar las prioridades de las características según avanza el proyecto, adaptándose a los nuevos contextos (Alaimo y Salías, 2015).

Esto es lo que dicen Alaimo y Salías (2015) acerca de la persona que cumple el papel de *product owner*:

se focaliza en maximizar la rentabilidad del producto. La principal herramienta con la que cuenta para poder realizar esta tarea es la priorización. Otra responsabilidad importante (...) es la gestión de las expectativas de los [y las] *stakeholders* mediante la comprensión total de la problemática de negocio y su descomposición. (p. 21)

Además, si bien quien tiene este rol puede ser representante de los deseos y decisiones de un comité, sigue habiendo una sola persona responsable, y la organización debe respetar y apoyar lo que esa persona decida (Alaimo y Salías, 2015).

Figura 5: *Product owner:* responsabilidades



Fuente: Buonomico, 2019, <https://bit.ly/2Z4s9M0>

Scrum master:

Según Schwaber y Sutherland (2016), la tarea principal de quien cumple el papel de Scrum *master* es asegurar que Scrum es entendido y adoptado. Para ello deben asegurarse que el Equipo Scrum se ajuste a la teoría, prácticas y reglas de Scrum. Estos autores agregan, además, lo siguiente acerca del rol en cuestión:

(...) líder al servicio del Equipo Scrum. También ayuda a las personas externas al Equipo Scrum a entender qué interacciones con el Equipo Scrum pueden ser de ayuda y cuáles no. (...) Ayuda a mejorar estas interacciones para maximizar el valor creado por el Equipo Scrum. (Schwaber y Sutherland, 2016, <https://bit.ly/3gwCoif>)

Scrum master al servicio de la organización

Existen varias maneras en las que la persona que cumple el papel de Scrum *master* brinda un servicio a la organización. Entre estas, podemos mencionar las siguientes actividades:

- Planificar, liderar y guiar a la organización en la adopción e implementación de Scrum.
- “Motivar cambios que incrementen la productividad del Equipo Scrum.
- Trabajar con otros [u otras] Scrum Masters para incrementar la efectividad de (...) Scrum” (Schwaber y Sutherland, 2016, <https://bit.ly/3gwCoif>).

Scrum masters al servicio de *product owners*

Según Schwaber y Sutherland (2016), los servicios que Scrum *masters* prestan a *product owners* son los que enumeramos a continuación:

- facilitar técnicas para gestionar el *product backlog* de manera efectiva;
- entender la planificación del producto en un entorno empírico;
- asegurar la priorización de los ítems del *product backlog* para maximizar el valor; y
- entender y practicar la agilidad.

Scrum masters al servicio del Equipo de Desarrollo

Asimismo, según Schwaber y Sutherland (2016), los servicios que Scrum *masters* prestan al Equipo de Desarrollo son los siguientes:

- guiar al Equipo para que se autoorganice y trabaje en su multifuncionalidad;
- ayudar al Equipo a hacer productos que tengan un valor alto;
- suprimir estorbos en el progreso del Equipo;
- facilitar los diversos eventos de Scrum (...); y
- guiar al Equipo en organizaciones en las que todavía no se ha adoptado Scrum ni entendido en su totalidad (Schwaber y Sutherland, 2016).

Alaimo y Salías (2015) nos dicen que las personas que tienen el rol de Scrum *master* también deben “detectar problemas y conflictos interpersonales dentro del equipo de trabajo” (p. 23). Lo ideal es que el equipo mismo sea quien los resuelva, pero de no ser posible, estas personas pueden “involucrarse y eventualmente a niveles más altos de la gerencia” (p. 24).

Figura 6: Scrum *master* al servicio de la organización



Fuente: [imagen sin título sobre Scrum *master* al servicio de la organización], 2020, <https://bit.ly/2Divj6C>

El Equipo de Desarrollo

El Equipo de Desarrollo está compuesto por personas que trabajan para entregar un incremento de producto “terminado”, que se pueda poner en producción, al final de cada *sprint*. Solo quienes forman parte del Equipo participan en la construcción del producto (Schwaber y Sutherland, 2016).

“Los Equipos de Desarrollo son empoderados por la organización para organizar y gestionar su propio trabajo. La sinergia resultante optimiza su eficiencia y efectividad” (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Los Equipos de Desarrollo son autoorganizados, pues nadie les indica cómo convertir ítems del *product backlog* en un incremento de producto. Son, también, multifuncionales, ya que cuentan como equipo con todas las habilidades necesarias para crear un potencial incremento.

Otra característica importante es que dentro de los equipos de desarrollo no hay títulos ni jerarquías, independientemente del trabajo que realice cada persona; tampoco se reconocen subequipos, sin importar los dominios particulares que requieran ser tenidos en cuenta. Quienes son integrantes individuales del equipo pueden contar con habilidades especializadas y estar en áreas en las que participen más, “pero la responsabilidad recae en el equipo como un todo” (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

El tamaño óptimo del Equipo de Desarrollo es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como

para completar una cantidad de trabajo significativa. Tener menos de tres [integrantes] en el Equipo de Desarrollo reduce la interacción y resulta en ganancias de productividad más pequeñas (...) Tener más de nueve [integrantes] en el equipo requiere demasiada coordinación (...) Los roles de Product owner y Scrum Master no cuentan en el cálculo del tamaño del equipo. (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

El Equipo de Desarrollo tiene tres responsabilidades fundamentales e indelegables:

- Proveer las estimaciones de cuánto esfuerzo será requerido para cada una de las características del producto. (...)
- Comprometerse al comienzo de cada Sprint a construir un determinado incremento de producto para cumplir con el objetivo buscado.
- Entregar el producto terminado al finalizar cada Sprint (Alaimo y Salías, 2015, p. 22).

Figura 7: Autoorganización



Fuente: [imagen sin título sobre autoorganización], s.f., <https://bit.ly/3e6cBvH>

Artefactos en Scrum

El lugar que tiene el diseño de los artefactos definidos por Scrum es muy importante, pues permite maximizar la transparencia de la información clave y otorgar espacios en donde se puedan llevar a cabo la inspección y adaptación. De esta forma, es posible construir un producto.

Product backlog:

Es el listado ordenado de todo lo que podría formar parte del producto y es la única fuente de requisitos para cualquier cambio por realizarse en dicho producto. La persona que cumpla el papel de *product owner* es la única responsable de su contenido, disponibilidad y orden (Raya, 2020).

El *product backlog* evoluciona a medida que el producto y el entorno también lo hacen. Es, además, dinámico, dado que cambia para identificar lo que el producto necesita para ser el adecuado. Mientras el producto exista, el *product backlog* también lo hará (Schwaber y Sutherland, 2016).

En el *product backlog* se enumeran todas las características, funcionalidades, requisitos, mejoras y correcciones del producto. Estos ítems deben contar con una descripción, ordenación, estimación y valor.

A medida que se utiliza un producto y se obtiene retroalimentación, el *product backlog* crece, y los requisitos cambian. Es por esto que decimos que es un artefacto vivo. Otros factores claves para que cambien los requisitos pueden ser cambios en el negocio, en las condiciones del mercado o en la tecnología.

El refinamiento del *product backlog* tiene lugar cuando se añaden detalle, estimaciones y orden a los ítems del *product backlog*. Es un proceso continuo, en el cual el o la *product owner* y el Equipo de Desarrollo examinan, revisan y discuten acerca de los detalles de los ítems. El Equipo Scrum coordina cómo y cuándo se hará, teniendo en cuenta que no debe consumir más del 10 % de la capacidad del Equipo de Desarrollo. Es importante aclarar que los ítems del *product backlog* pueden ser actualizados en cualquier momento por el o la *product owner*, y a su criterio (Schwaber y Sutherland, 2016).

Los ítems del *product backlog* que se encuentran en la parte superior son generalmente más claros y detallados que los de menor orden. Las estimaciones son más precisas, puesto que se cuenta con mayor claridad y detalle. Los ítems con los que trabajará el Equipo de Desarrollo en el próximo *sprint* están descompuestos, de forma que cualquier elemento pueda ser “terminado” dentro del *sprint* (Alaimo y Salías, 2015).

El Equipo de Desarrollo es quien estima, mientras que quien cumple la función de *product owner* podría ayudar al Equipo a entender y elegir soluciones de compromiso.

En caso de que varios Equipos Scrum trabajen juntos en el mismo producto, se utiliza un único *product backlog* y se emplea un atributo de los ítems para agrupar los elementos.

Priorización del *product backlog*:

- **Priorización por valor de negocio de cada PBI:**
 - Podemos decir que el valor de negocio es la relevancia que un ítem tiene para el cumplimiento del objetivo de negocio que estamos buscando.
- **Priorización por retorno de la inversión (ROI) de cada PBI:**
 - Consta de calcular el beneficio económico que se obtendrá en función de la inversión que se deba realizar.

- **Prioridades según la importancia y el riesgo de cada PBI:**
 - Ya sea que los ítems se prioricen por valor de negocio o por ROI, estos pueden verse afectados por el nivel de riesgo asociado a cada uno de ellos.

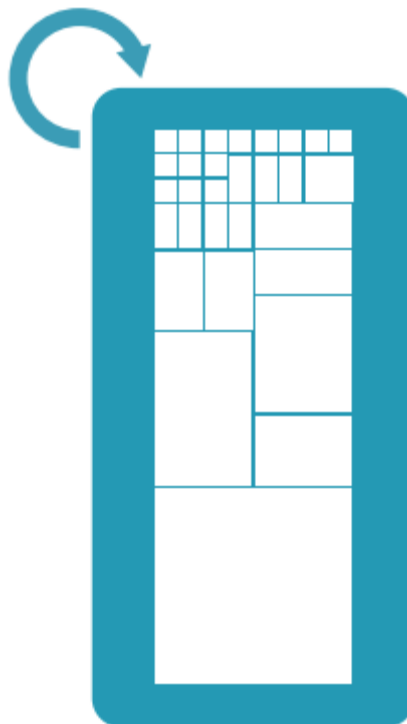
Seguimiento de progreso:

En cualquier momento, la persona que cumple el rol de *product owner* tiene la posibilidad de sumar el trabajo total restante para alcanzar el objetivo. “Compara esta cantidad con el trabajo restante, para evaluar el progreso hacia la finalización del trabajo proyectado en el tiempo deseado para el objetivo” (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Las prácticas de proyección más utilizadas para predecir el progreso son las siguientes:

- Trabajo consumido (*burndown*)
- Trabajo avanzado (*burnup*)
- Flujo acumulado (*cumulative flow*)

Figura 8: *Product backlog*



Fuente: [imagen sin título sobre *Product Backlog*], 2018, <https://bit.ly/31TXeE0>

Sprint backlog:

El *sprint backlog* es el conjunto de ítems del *product backlog* seleccionados para el *sprint*, junto con un plan para entregar el incremento de producto. De esta forma, se consigue el objetivo del *sprint*. El *sprint backlog* lo realiza el Equipo de Desarrollo y contempla la funcionalidad que formará parte del próximo incremento y el trabajo necesario para su entrega (Schwaber y Sutherland, 2013).

En el *sprint backlog* se muestra todo el trabajo que el Equipo de Desarrollo identifica como necesario para alcanzar el objetivo del *sprint*.

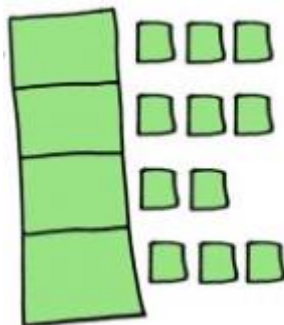
El *sprint backlog* cuentan con un nivel de detalle suficiente como para que se pueda seguir el progreso durante las *daily Scrum* (Scrum diario). El Equipo de Desarrollo va actualizando el estado de los pendientes del *sprint backlog* durante el *sprint*; a medida que el Equipo de Desarrollo trabaja sobre el plan, va aprendiendo más acerca del trabajo necesario para conseguir el objetivo del *sprint*.

Si se descubre que es necesario realizar nuevo trabajo, se agrega al Sprint Backlog; si algunos de los elementos del plan del Sprint se tornan innecesarios, se remueven del Sprint Backlog. Solo el Equipo de Desarrollo tiene la potestad de alterar el Sprint Backlog durante la ejecución del Sprint (Alaimo y Salías, 2015, p.29).

Seguimiento del progreso del *sprint*

En cualquier momento durante un Sprint, es posible sumar el trabajo restante total en los elementos de la Lista de Pendientes del Sprint. El Equipo de Desarrollo hace seguimiento de este trabajo restante total al menos en cada Scrum Diario para proyectar la posibilidad de conseguir el Objetivo del Sprint. Haciendo seguimiento del trabajo restante a lo largo del Sprint, el Equipo de Desarrollo puede gestionar su progreso. (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

Figura 9: Sprint backlog



Incremento:

El Incremento es la suma de todos los elementos de la Lista de Producto completados durante un Sprint y el valor de los incrementos de todos los Sprints anteriores. Al final de un Sprint, el nuevo Incremento debe estar “Terminado”, lo cual significa que está en condiciones de ser utilizado y que cumple la Definición de “Terminado” del Equipo Scrum (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Es importante aclarar que es un **incremento** porque es una característica funcional nueva o modificada de un producto que está siendo construido evolutivamente, y **terminado** porque está en condiciones de ser utilizable y, además, cumple con la *definition of done* establecida por el Equipo Scrum.

Figura 10: Definición de “terminado”



Es el conjunto de características que un ítem del *product backlog* debe cumplir para que el Equipo de Desarrollo pueda determinar si ha terminado de trabajar en él. Cuando un ítem del *product backlog* o un incremento se describe como “terminado”, todo el equipo debe entender lo que significa “terminado”. El Equipo entero debe tener un entendimiento compartido de lo que significa que el trabajo esté completado; de esta forma se podrá evaluar cuándo se ha finalizado el trabajo sobre el incremento de producto (Schwaber y Sutherland, 2013).

Esta definición ayuda al Equipo de Desarrollo a saber cuántos ítems del *product backlog* puede seleccionar durante la *planning* del *sprint*.

Eventos en Scrum

En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (*time-boxes*), de tal modo que todos tienen una duración máxima. Una vez que comienza un Sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento. (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

Los demás eventos pueden terminar siempre que se alcance el objetivo del evento.

Además del propio Sprint, que es un contenedor del resto de eventos, cada uno de los eventos de Scrum constituye una oportunidad formal para la inspección y adaptación de algún aspecto. Estos eventos habilitan la transparencia e inspección. La falta de alguno de estos reduce la transparencia e inspección. (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

Sprint

El corazón de Scrum es el *sprint*, es un bloque de tiempo fijo (*time-box*) que va de 1 a 4 semanas, durante el cual se crea un incremento de producto “terminado”, utilizable y potencialmente desplegable. Es aconsejable que la duración de los *sprints* sea consistente a lo largo del desarrollo. Cada *sprint* comienza inmediatamente después de la finalización del anterior (Schwaber y Sutherland, 2013).

En el transcurso de las sprints ocurren los *sprint planning meeting*, las *dailies*, el trabajo de desarrollo, la *sprint review*, y la *sprint retrospective*. Otro punto importante para destacar es que no se realizan cambios que puedan afectar al objetivo del *sprint*, los objetivos de calidad no disminuyen, y el alcance puede ser aclarado y renegociado entre la persona que sea *product owner* y el Equipo de Desarrollo (Schwaber y Sutherland, 2013).

Cada *sprint* tiene una definición de lo que se va a construir, un diseño y un plan flexible que guiará dicha construcción, el trabajo y el producto resultante.

Los Sprints están limitados a un mes calendario. Cuando el horizonte de un Sprint es demasiado grande, la definición de lo que se está construyendo podría cambiar, la complejidad podría elevarse y el riesgo podrían aumentar. Los Sprints habilitan la predictibilidad al asegurar la inspección y adaptación del progreso y también limitan el riesgo de elevar los costos (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Adelantos o retrasos en un *sprint*

Dado que el *time-boxing* no nos permitirá modificar la fecha de entrega o finalización del *sprint*, nos guiaremos por su alcance, es decir, en el caso de adelantarnos, deberemos incrementar el alcance del *sprint* agregando nuevos ítems del *product backlog* y reducirlo en el caso de retrasarnos.

Cancelación de un *sprint*

Un *sprint* puede cancelarse antes de que se acabe el tiempo, y solo la persona que es dueña del producto tiene autoridad para hacerlo. Cabe destacar, sin embargo, que puede tomar la decisión de cancelar el *sprint* influenciada por las personas que estén interesadas, por el Equipo de Desarrollo o por quien cumpla la función de Scrum *master* (Schwaber y Sutherland, 2013)

La cancelación de un *sprint* tendría sentido en el caso de que su objetivo resultase obsoleto. Este fenómeno podría ser resultado de un cambio de dirección por parte de la compañía o de cambios en las condiciones del mercado o de la tecnología. Sin embargo, por su corta duración, es muy raro que la cancelación tenga sentido alguno.

Quando se cancela un Sprint, se revisan todos los Elementos de la Lista de Producto que se hayan completado y “Terminado”. Si una parte del trabajo es potencialmente entregable, el Dueño [o Dueña] del Producto normalmente lo acepta. Todos los Elementos de la Lista de Producto no completados se vuelven a estimar y se vuelven a introducir en la Lista de Producto. El trabajo finalizado en ellos pierde valor con rapidez y frecuentemente debe volverse a estimar (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Reunión de planificación de *sprint* (*Sprint Planning Meeting*)

El trabajo a realizar durante el Sprint se planifica en la Reunión de Planificación de Sprint. Este plan se crea mediante el trabajo colaborativo del Equipo Scrum completo. La Reunión de Planificación de Sprint tiene un máximo de duración de ocho horas para un Sprint de un mes. Para Sprints más cortos, el evento es usualmente más corto. El [o la] Scrum Master se asegura de que el evento se lleve a cabo y que los [y las] asistentes entiendan su propósito. El [o la] Scrum Master enseña al Equipo Scrum a mantenerse dentro del bloque de tiempo. La Reunión de Planificación de Sprint responde a las siguientes preguntas:

- ¿Qué puede entregarse en el Incremento resultante del Sprint que comienza?

- ¿Cómo se conseguirá hacer el trabajo necesario para entregar el Incremento?
(Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Parte uno: ¿Qué puede ser terminado en este *sprint*?

El Equipo Scrum completo colabora en el entendimiento del trabajo del *sprint*.

Mientras que quien cumple el rol de *product owner* propone al equipo el objetivo que el *sprint* debería lograr y los ítems del *product backlog* que se deberían contemplar para alcanzar el objetivo del *sprint*, el Equipo de Desarrollo trabaja en proyectar la funcionalidad que se desarrollará durante el *sprint* (Schwaber y Sutherland, 2013).

La entrada a esta reunión está conformada por el *product backlog*, el último incremento de producto, la capacidad del Equipo de Desarrollo para el *sprint*, y el rendimiento pasado del Equipo de Desarrollo. La cantidad de ítems del *product backlog* seleccionados para el *sprint* depende exclusivamente del Equipo de Desarrollo. Solo este puede evaluar qué es capaz de entregar durante el próximo *sprint* (Schwaber y Sutherland, 2013).

Después de que el Equipo de Desarrollo proyecta qué ítems del *product backlog* entregará en el *sprint*, el Equipo Scrum elabora el objetivo del *sprint* que debería lograrse.

Al finalizar esta primera parte de la reunión, tanto *stakeholders* (si hubiese) como la persona que sea *product owner* podrían irse, y dejarían que la o el Scrum *master* y el Equipo de Desarrollo den comienzo a la segunda parte.

Parte dos: ¿Cómo se conseguirá completar el trabajo seleccionado?

Una vez que se ha establecido el objetivo, el Equipo de Desarrollo decide **cómo** construirá esta funcionalidad para formar un incremento de producto “terminado”. Los ítems del *product backlog* seleccionados para este *sprint*, junto con el plan para terminarlos, recibe el nombre de *sprint backlog*.

El Equipo de Desarrollo comienza a hacer un diseño de alto nivel del sistema y a determinar la forma en la que llevará adelante el trabajo. El trabajo podría ser de tamaño o esfuerzo estimado variables; sin embargo, durante la *sprint planning*, se planifica suficiente trabajo como para que el Equipo de Desarrollo pueda completarlo en el *sprint* que comienza (Schwaber y Sutherland, 2013).

Para el final de esta reunión, el trabajo planificado por el Equipo de Desarrollo para los primeros días del Sprint es descompuesto en unidades de un día o menos. El Equipo de desarrollo se autoorganiza para completar trabajo comprometido a lo largo del Sprint. (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

La persona que cumpla el papel de *product owner* puede ayudar a clarificar los ítems del *product backlog* seleccionados. Si el Equipo de Desarrollo determina que tiene demasiado trabajo o que no tiene suficiente trabajo, el o la *product owner* podría renegociar el alcance. En caso de necesitarlo, el Equipo de Desarrollo podría invitar a otras personas a que asistan con el fin de que proporcionen asesoría técnica (Schwaber y Sutherland, 2013).

Al finalizar la reunión de planificación de *sprint*, el Equipo de Desarrollo debería haber arribado a un *sprint backlog*, por lo cual, debería ser capaz de explicar a las personas cuyos roles son de *product owner* y de *Scrum master* cómo pretende trabajar como un equipo autoorganizado para lograr el objetivo del *sprint* y crear el incremento esperado.

Objetivo del *sprint* (*Sprint Goal*)

El Objetivo del Sprint es una meta establecida para el Sprint que puede ser alcanzada mediante la implementación de la Lista de Producto. Proporciona una guía al Equipo de Desarrollo acerca de por qué está construyendo el incremento. Es creado durante la reunión de Planificación del Sprint. El objetivo del Sprint ofrece al Equipo de Desarrollo cierta flexibilidad con respecto a la funcionalidad implementada en el Sprint. Los elementos de la Lista del Producto seleccionados ofrecen una función coherente, que puede ser el objetivo del Sprint. El objetivo del Sprint puede representar otro nexo de unión que haga que el Equipo de Desarrollo trabaje en conjunto y no en iniciativas separadas.

A medida que el Equipo de Desarrollo trabaja, se mantiene el objetivo del Sprint en mente. Con el fin de satisfacer el objetivo del Sprint se implementa la funcionalidad y la tecnología (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>).

Scrum diario (*daily Scrum*)

La Daily Scrum es una reunión con un bloque de tiempo de 15 minutos para que el Equipo de Desarrollo sincronice sus actividades y cree un plan para las siguientes 24 horas. Esto se lleva a cabo inspeccionando el trabajo avanzado desde el último Scrum Diario y haciendo una proyección acerca del trabajo que podría completarse antes del siguiente.

El Scrum Diario se realiza a la misma hora y en el mismo lugar todos los días para reducir la complejidad. Durante la reunión, cada [integrante] del Equipo de Desarrollo explica:

- ¿Qué hice ayer que ayudó al Equipo de Desarrollo a lograr el Objetivo del Sprint?
- ¿Qué haré hoy para ayudar al Equipo de Desarrollo a lograr el Objetivo del Sprint?

- ¿Veo algún impedimento que evite que el Equipo de Desarrollo o yo logremos el Objetivo del Sprint? (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>)

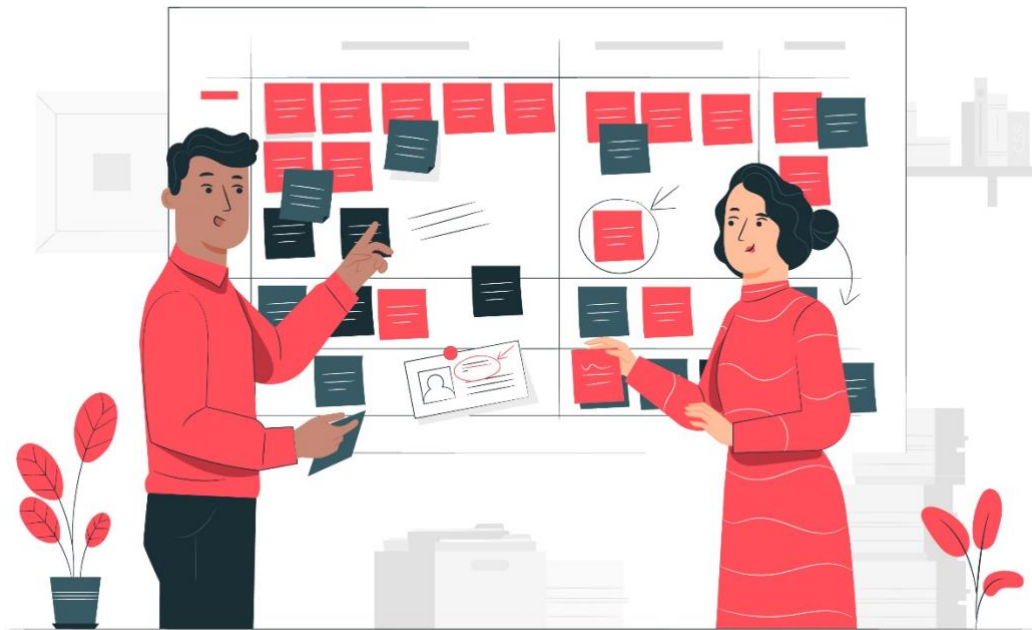
El Equipo de Desarrollo utiliza la *daily Scrum* para evaluar el progreso y la tendencia que sigue en vista al objetivo del *sprint*. Cada día, el Equipo de Desarrollo debería pensar cómo trabajar de manera autoorganizada para crear el incremento esperado hacia el final del *sprint*. Las personas que son integrantes del equipo por lo general se vuelven a reunir inmediatamente después de la *daily Scrum*, para tener discusiones detalladas o para adaptar o replanificar el resto del trabajo del *sprint* (Schwaber y Sutherland, 2013).

La persona cuyo rol es de Scrum *master* vela por que se realice la reunión, pero el Equipo de Desarrollo es el responsable de dirigirla. El o la Scrum *master* instruye al Equipo de Desarrollo para que se lleve a cabo en el *time-box* establecido (15 minutos), y también se asegura de que solo quienes conforman el Equipo de Desarrollo participen (Schwaber y Sutherland, 2013).

Ventajas de la *daily Scrum*:

- Mejora la comunicación.
- Elimina la necesidad de mantener otras reuniones.
- Ayuda a identificar y eliminar impedimentos relativos al desarrollo.
- Promueve la toma de decisiones rápida.
- Mejora el nivel de conocimiento del Equipo de Desarrollo.
- Constituye una reunión clave de inspección y adaptación (Schwaber y Sutherland, 2013).

Figura 11: *Daily Scrum*



Fuente: [imagen sin título sobre *daily Scrum*], s.f., <https://bit.ly/2BE9gqF>

Revisión de *sprint* (*sprint review*)

Al final del *sprint*, se lleva a cabo una *sprint review*, donde tanto *stakeholders* como el Equipo revisan lo realizado durante el *sprint*, es decir, inspeccionan el incremento. Sobre la base de esto, las personas cuya función es la de asistir colaboran para determinar las siguientes medidas que podrían tomarse para optimizar el valor, y dan *feedback* al Equipo de Desarrollo para que obtenga información y para fomentar la colaboración (Schwaber y Sutherland, 2013).

Para *sprints* de un mes, el tiempo es de cuatro horas. Para *sprints* más cortos, el tiempo es proporcionalmente menor. El o la Scrum *master* se asegura de que el evento se lleve a cabo y que quienes asisten entiendan su finalidad y que el *time-box* sea el fijado (Schwaber y Sutherland, 2013).

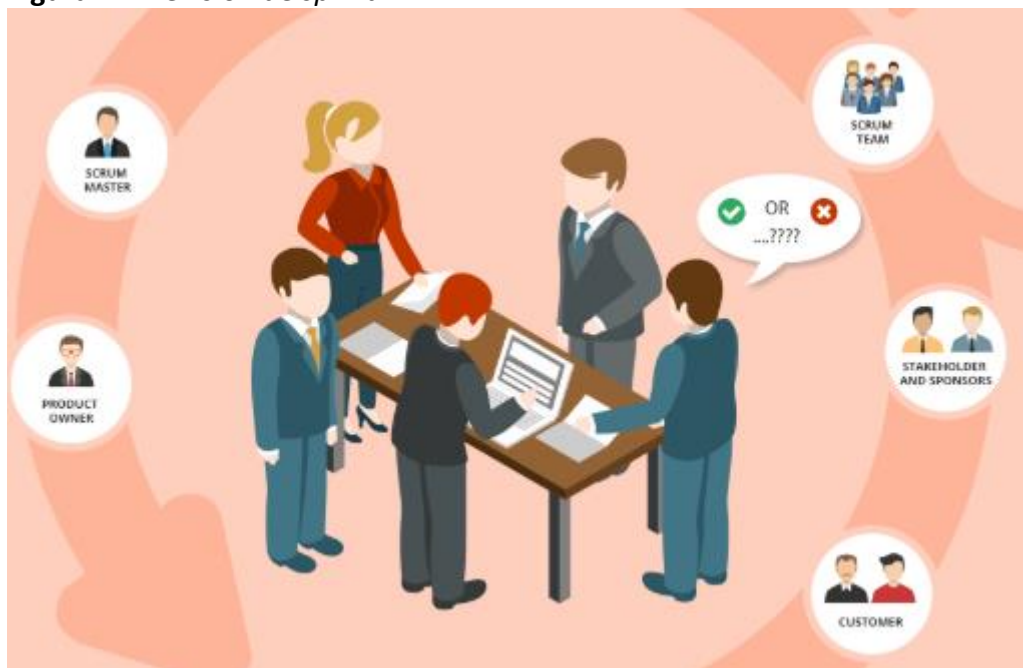
Siguiendo a Schwaber y Sutherland (2013), la *sprint review* incluye los siguientes elementos:

- Las personas asistentes son el Equipo Scrum y quienes tengan interés y hayan recibido invitación de la o el *product owner*.
- La persona cuyo rol es el de *product owner* explica qué elementos de la lista de producto se han “terminado” y cuáles no.
- El Equipo de Desarrollo comenta acerca de lo que se hizo bien durante el *sprint*, los problemas que aparecieron y la forma en la que se resolvieron.

- El Equipo de Desarrollo muestra el trabajo que se ha “terminado” y contesta cuestiones respecto del incremento.
- La persona que es *product owner* expone la lista de producto en su estado actual y (de ser necesario) estipula fechas probables de finalización, según el progreso que se haya logrado a la fecha.
- Todo el grupo opina respecto de qué pasos seguir a continuación, de manera tal que la *sprint review* dé información de entrada sustancial para las reuniones de planificación de *sprints* venideras.
- Se analiza cómo el mercado o el uso potencial del producto podrían haber cambiado lo que tiene más valor para llevar a cabo a continuación.
- Se revisan la línea de tiempo, las capacidades potenciales, el presupuesto y el mercado para la siguiente entrega prevista del producto.

El resultado de la *sprint review* es un *product backlog* revisado, que define los ítems posibles para el siguiente *sprint*. Es posible, además, que el *product backlog* sea priorizado para enfocarse en nuevas oportunidades.

Figura 12: Revisión de *sprint*



Fuente: [imagen sin título sobre revisión de *sprint*], s.f., <https://bit.ly/3iG74zi>

Retrospectiva de *sprint* (*sprint retrospective*)

“La Retrospectiva es el espacio que tiene el Equipo Scrum de inspeccionarse a sí mismo y crear un plan de mejoras que sean abordadas durante el siguiente Sprint” (Schwaber y Sutherland, 2013, <https://bit.ly/38x25fR>). Se podría decir que es el corazón de la mejora continua y las prácticas emergentes. Tiene lugar después de la *sprint review* y antes de la siguiente *sprint planning*. Su *time-box* es de tres horas para *sprints* de un mes. Para *sprints* más cortos, el tiempo será proporcionalmente menor (Schwaber y Sutherland, 2013).

La persona que sea *Scrum master* vela por que se lleve a cabo y que las personas asistentes entiendan su propósito. También se encarga de que se respete el *time-box* prefijado y participa como integrante del equipo, ya que la responsabilidad del proceso Scrum recae sobre el menor (Schwaber y Sutherland, 2013).

Para Schwaber y Sutherland (2013), la retrospectiva tiene los siguientes propósitos:

- inspeccionar cómo fue el último Sprint en cuanto a personas, relaciones, procesos y herramientas;
- identificar y ordenar los elementos más importantes que salieron bien y las mejoras;
- crear un plan para implementar las mejoras a la forma en la que el Equipo Scrum desempeña su trabajo.

El [o la] *Scrum master* alienta al equipo para que mejore, dentro del marco de proceso Scrum, su proceso de desarrollo y sus prácticas para hacerlos más efectivos y amenos para el siguiente Sprint. Durante cada Retrospectiva de Sprint, el Equipo Scrum planifica formas de aumentar la calidad del producto mediante la adaptación de la Definición de “Terminado” (Definition of “Done”) según sea conveniente.

Este tipo de actividad necesita un ambiente seguro donde el Equipo Scrum pueda expresarse libremente, sin censura ni temores, por lo cual se restringe solo al Equipo de Desarrollo y Scrum Master.

Para el final de la Retrospectiva de Sprint, el Equipo Scrum debería haber identificado mejoras que implementará en el próximo Sprint. El hecho de implementar estas mejoras en el siguiente Sprint constituye la adaptación fruto de la inspección del Equipo de Desarrollo. (<https://bit.ly/38x25fR>)

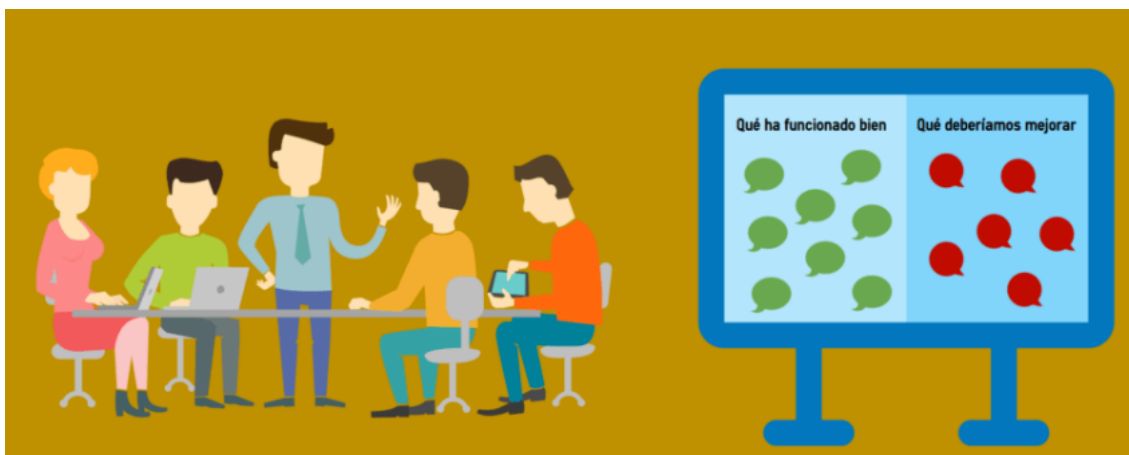
Refinamiento del *product backlog*

“El refinamiento del Backlog es una actividad constante a lo largo de todo el Sprint, aunque algunos equipos prefieren concentrarla en una reunión que se realiza durante el Sprint” (Alaimo y Salías, 2015, p. 36). Su objetivo es profundizar en el entendimiento de los ítems del *product backlog* que se encuentran más allá del *sprint* actual, y así dividirlos en ítems más pequeños y estimarlos (por lo general, se revisan aquellos que pueden ser tenidos en cuenta en los próximos *sprints*).

Otro objetivo importante que se debe perseguir es la detección de riesgos, y, en función de ellos, ajustar las prioridades del *product backlog* (Alaimo y Salías, 2015).

Debe participar todo el Equipo Scrum, y en caso de que se realice una reunión (una o dos por *sprint*), la responsabilidad de convocarla es de la persona que sea *product owner*, y la de facilitarla es de aquella que sea *Scrum master* (Alaimo y Salías, 2015).

Figura 13: Refinamiento del *product backlog*



Fuente: [imagen sin título sobre refinamiento del *Product Backlog*], s.f.,
<https://bit.ly/2O5IR7s>

Desarrollo evolutivo

A continuación, veremos dos conceptos básicos de desarrollo evolutivo dentro de marcos ágiles, que servirán a la hora de construir producto o servicios:

Minimum Viable Product (MVP)

El Minimum Viable Product (MVP) es un concepto de Lean Startup, y hace referencia a la versión mínima de un producto, que nos permite recolectar la mayor cantidad de información de nuestro mercado y clientes [y clientas] con el menor esfuerzo posible.

Consiste en enfocarnos en las características mínimas y necesarias para que el producto pueda lanzarse al mercado. Esto nos evitará: crear productos que nadie necesita, maximizar el aprendizaje por dinero invertido. (Alaimo y Salías, 2015, p. 39)

La frase del cocreador de LinkedIn lo resume muy bien: “Si no te avergüenzas de tu primer producto, es que lo lanzaste muy tarde” (Hoffman citado en Llamas, 2018, <https://bit.ly/3gyeMtw>).

Minimum Marketable Features (MMF)

Todas las metodologías ágiles coinciden en que un producto debe construirse de forma evolutiva en pequeñas entregas. Para ello, debemos hacer que cada entrega pueda aportar suficiente valor a usuarios y usuarias finales. Esos grupos de características se denominan *Minimum Marketable Features* (MMF) y se definen como el conjunto más

pequeño posible de funcionalidad que, por sí misma, tiene valor en el mercado (Alaimo y Salías, 2015).

User Story Mapping

“Conjugando el Desarrollo Evolutivo, la Priorización del Backlog y el concepto de Minimum Marketable Feature, Jeff Patton plantea una técnica de Análisis Ágil llamada User Story Mapping” (Alaimo y Salías, 2015, p. 40).

Consiste en ordenar historias de usuarios y usuarias a lo largo de dos dimensiones independientes. El mapeo “organiza las actividades del usuario [o usuaria] a lo largo del eje horizontal en un orden por prioridad” (Avedaño, 2019, <https://bit.ly/38xXb2b>). Y hacia abajo del eje vertical, representa una creciente sofisticación de la implementación.

Una vez organizado el mapeo, la primera fila horizontal representa un "esqueleto ambulante", es decir, una versión básica pero utilizable del producto. Trabajar en filas sucesivas desarrolla el producto con una funcionalidad adicional.

Historias de usuario

El origen de las historias de usuario se remonta a la metodología de desarrollo ágil XP (eXtreme Programming), creada por Kent Beck en 1999. Habitualmente, se utilizan en todas las metodologías de desarrollo y marcos ágiles para escribir los requisitos y las pruebas de validación.

Para solucionar los aspectos negativos de las metodologías tradicionales de desarrollo, los marcos ágiles ofrecen otra forma de abordar los requisitos mediante las historias de usuario.

Ya no es necesario formular listas largas de requisitos funcionales y características técnicas, sino que se prioriza hablar de las funcionalidades deseadas y fomentar la conversación cara a cara y el análisis compartido entre quienes necesitan usar el producto y quienes lo construirían.

Con el foco puesto en que esas conversaciones existan, podemos decir que las historias de usuario son especificaciones funcionales que invitan a la conversación para que el detalle sea consecuencia de esta última.

Componentes de las historias de usuario

Estas historias están compuestas de tres elementos fundamentales, conocidos como “las tres Cs”. Sin ellos, las historias de usuario no estarían completas.

Figura 14: Componentes de historias de usuario



Fuente: [imagen sin título sobre componentes de una historia de usuario], s.f., <https://bit.ly/3e8mbOv>

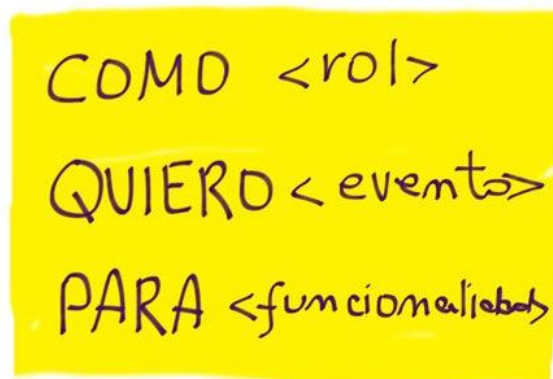
- **Card (ficha):** toda historia de usuario debe poder describirse en una pequeña tarjeta. Si no puede describirse en ese tamaño, nos está indicando que estamos comunicando demasiada información y que debería compartirse cara a cara (Alaimo y Salías, 2015).
- **Conversación:** toda historia debe ser conversada con la persona que sea *product owner*. Esta conversación debe ser cara a cara y se tiene que saber responder a cuestiones sobre el valor y el resultado esperado de la implementación. También se deberían poder expresar opiniones y sentimientos (Alaimo y Salías, 2015).
- **Confirmación:** la confirmación es un acuerdo que refleja que todas las personas comprenden el resultado esperado de la historia. Esta debe estar lo suficientemente explicada para que el Equipo de Desarrollo sepa qué es lo que debe construir y qué es lo que la o el *product owner* espera. Esto también se conoce también como “criterios de aceptación” (Alaimo y Salías, 2015).

Redacción de una historia de usuario:

Las historias de usuario suelen escribirse siguiendo un formato muy sencillo:

- Como (rol)
- Quiero (funcionalidad)
- Para (beneficio)

Figura 15: Redacción de una historia de usuario



Fuente: Urtanta, s.f., <https://bit.ly/2AHaXD8>

Al escribirlas en este formato, ayuda a quien las lee a ponerse en el lugar del usuario o la usuaria, dado que están escritas en **primera persona**; esto facilita la **priorización** de la persona que sea *product owner* y evita que deba dedicar mucho esfuerzo para comprender cuál es la funcionalidad, su valor y a quién se beneficia. Además, permite conocer el **propósito** de una funcionalidad, lo que habilita al Equipo de Desarrollo a proponer nuevas alternativas que cumplan con el mismo propósito.

INVEST

Esta metodología, desarrollada por Bill Wake, nos ayuda a garantizar que las historias de usuario contengan valor de negocio, aunque se desarrollen en una sola iteración. Sus siglas (INVEST) nos recuerdan que estas historias deben tener las siguientes características:

I - *Independent* (independientes),
N - *Negotiable* (negociables),
V - *Valuable* (valiosas),
E - *Estimable* (estimables),
S - *Small* (pequeñas), y
T - *Testable* (comprobables).

- ***Independent* (independientes):** deben ser planificadas e implementadas en cualquier orden. Para ello, las historias deberían de ser totalmente independientes entre sí. Una manera de independizarlas es combinándolas en una o dividiéndolas de manera diferente.
- ***Negotiable* (negociables):** deben ser negociables, ya que los detalles serán acordados y conversados con la clientela o el usuario o la usuaria. Se deben evitar historias con muchos detalles porque limitaría la conversación acerca de estas.
- ***Valuable* (valiosas):** deben aportar algún valor al usuario o la usuaria final. Una manera de hacerlo es que quien escriba sea precisamente la usuaria o el usuario.

- **Estimable (estimable):** deben poder ser estimadas con la precisión suficiente para ayudar a la clientela a priorizar y planificar su implementación. La estimación la realizará el Equipo de Desarrollo y está relacionada con el tamaño de la historia y con el conocimiento del equipo.
- **Small (pequeña):** deben ser lo suficientemente pequeñas para que permitan ser estimadas por el Equipo de Desarrollo. Si bien no hay una medida explícita, tener entre cuatro y seis historias por *sprint* es una buena señal de tamaño.
- **Testable (comprobable):** deben poder probarse. Si la usuaria o el usuario no sabe cómo probar, significa que la historia no es del todo clara o que no es valiosa. Si el equipo no la puede probar, nunca sabrá si la ha terminado o no.

Definición de “listo”

También conocida como “*definition of ready*”, este término hace referencia al conjunto de características que una historia de usuario debe cumplir para que el Equipo de Desarrollo pueda incluirla en el *sprint backlog* y comprometerse con su entrega.

Una típica definición de “listo” podría consistir en los siguientes puntos:

- La historia de usuario debe ser INVEST.
- Todos sus prerequisites están resueltos.

Estimaciones en agilidad

Cuando nos encontramos en contextos inciertos, proveer una estimación precisa en etapas tempranas de un proyecto puede hacer que sea poco probable el cumplimiento del compromiso por parte del equipo.

“A medida que adquiramos conocimiento, nuestras estimaciones serán cada vez más precisas. El problema surge a la hora de estimar cuando las decisiones se toman en base a supuestos que probablemente no ocurran o sean incorrectos” (Alaimo y Salías, 2015, p. 61).

En relación con lo anterior, Alaimo y Salías (2015) agregan lo que citamos a continuación:

Los marcos ágiles proponen trabajar en un proyecto sin la necesidad de tener una estimación precisa y siendo conscientes de que la estimación inicial es de un orden probable, para poder ganar experiencia rápidamente y así estimar con mayor certeza prescindiendo de supuestos. (p. 61)

Para mitigar el riesgo, en los marcos ágiles se opta por reducir la precisión de las estimaciones en función del esfuerzo que se requiere para estimar. De esta manera,

los “requerimientos” y sus “estimaciones” se categorizan en diferentes niveles de precisión.

A continuación, vamos a detallar la siguiente escala de ítems del *product backlog* y estimaciones:

- **Alto nivel:** EPICAS estimada en Tamaño (XS, S, M, L, XL).
- **Nivel medio:** historia de usuario estimada en puntos de historia según serie de Fibonacci.
- **Bajo nivel:** tareas o actividades estimadas en horas.

Al comenzar el proyecto, nuestro *product backlog* se compone de bloques funcionales que podemos estimar según sus tamaños:

- XS - Muy Pequeño
- S -Pequeño
- M - Mediano
- L - Grande
- XL - Muy Grande

Esto nos permitirá tener una aproximación a la problemática de negocio y a las características del producto que se construirá.

Conociendo las prioridades de los bloques funcionales, se toman los de mayor prioridad y se descomponen en funcionalidades más específicas y de menor nivel, llamadas Historias de Usuario, a las cuales estimaremos utilizando la serie de Fibonacci: 0, 1, 2, 3, 5, 8, 13, 21, 40, 100.

Para estimar las Historias de Usuario, utilizamos una técnica comparativa llamada Estimación Relativa. Consta de asignar uno de los números de la serie de Fibonacci a cada una de las Historias de Usuario. De esta manera, aquellas historias que tengan el número 2 requerirán aproximadamente el doble de esfuerzo que las que lleven el número 1, aquellas que lleven el número 3 requerirán el triple de esfuerzo de las que lleven el número 1, etc. (Alaimo y Salías, 2015, p. 2)

Por último, tenemos el nivel más bajo de estimación: por horas. Solo aplica a las tareas o actividades que formen parte de un *sprint*. Durante la planificación del *sprint*, estas historias de usuario son divididas por el Equipo de Desarrollo en tareas o actividades y estimadas en horas.

Técnica de estimación Planning Poker

James Greening presentó en 2002 “Planning Poker (o cómo evitar análisis parálisis en la planificación de liberaciones)”, que se basa en realizar la estimación de requerimientos

de forma colaborativa dentro de un equipo (Alaimo y Salías, 2015). La técnica consiste en lo siguiente: cada integrante del equipo posee en sus manos una baraja de cartas con los números correspondientes a la serie de Fibonacci, y se siguen los siguientes pasos:

- La persona que es *product owner* presenta una historia de usuario para ser estimada.
- Quienes sean participantes proceden a realizar su estimación sin mostrarla para no influenciar al resto del equipo, poniendo su carta elegida boca abajo sobre la mesa.
- Una vez que la totalidad de participantes ha estimado, se dan vuelta las cartas y se discuten principalmente los extremos.
- Al finalizar la discusión, se levantan las cartas y se vuelve a estimar, esta vez con mayor información que la que se tenía previamente.
- Las rondas siguen hasta que se logra un consenso en el equipo.
- Se repiten los pasos anteriores con una nueva historia de usuarios (Alaimo y Salías, 2015).

Figura 16: Planning Poker



Fuente: [imagen sin título sobre Planning Poker], s.f., <https://bit.ly/3il961R>

Jira y Scrum

Si bien uno de los valores de la agilidad nos dice “Personas e interacciones sobre procesos y herramientas”, esto no quiere decir que los procesos y las herramientas no sean importantes, sino que lo son, pero en menor medida. Contar con un *product backlog* digital nos permite reflejar todas las tareas a las que el equipo dedica tiempo, incluido el trabajo interno. Esto nos ayuda a establecer las expectativas con las partes interesadas y con otros equipos. Además, una herramienta digital en tiempos cuando el trabajo presencial es dificultoso se torna imprescindible. A continuación, veremos Jira, uno de los *software* más utilizados a la hora de implementar Scrum.

Creación de un proyecto Scrum en Jira

Crear un proyecto es muy fácil. Para ello, vamos a ir a la pantalla «Proyectos» y luego haremos clic en el botón «Crear proyecto». Ahora tendremos que decidir si crearemos un proyecto clásico o de nueva generación. Una de las diferencias más importantes entre ellos es que los de nueva generación nos ahorran tiempo de configuración, pero los informes son muy limitados, lo que nos dificulta las mediciones del Equipo Scrum, como por ejemplo la capacidad de entrega o *time-to-market*. Recordemos que medir es algo muy importante dentro del marco Scrum.

Figura 17: Creación de un proyecto Scrum en Jira



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Una vez creado el proyecto, podemos elegir qué plantilla vamos a utilizar. En nuestro caso, elegiremos Scrum, pero bien podríamos hacerlo con Kanban.

Figura 18: Elección de plantilla en Jira

Crear proyecto

Nombre
Test

Clave
TEST i

☐ Comparte la configuración con un proyecto ya existente

Plantilla



Scrum
Administra historias, tareas y flujos de trabajo de un equipo de Scrum
Para equipos que entregan trabajo según un cronograma periódico

Cambiar plantilla

Crear

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Ni bien lo terminamos de crear, podemos abrir un tablero, y seleccionar «Scrum» o «Kanban». Podemos crear tableros para varios proyectos o viceversa. Por lo general los tableros se utilizan para generar reportes o para organizar el trabajo del equipo. A continuación, crearemos un tablero Scrum para el resto de configuración.

La idea es tener un proyecto Scrum con un tablero Kanban.

Figura 19: Creación de un tablero en Jira

Crear un tablero

Scrum

Scrum se centra en la planificación, la asignación y la entrega de fragmentos de trabajo enmarcados en el tiempo denominados 'sprints'

Crear un tablero de Scrum

Crear un tablero de Scrum con datos de muestra

Kanban

Kanban se centra en visualizar el flujo de trabajo y limitar el trabajo en curso para facilitar las mejoras graduales del proceso existente

Crear un tablero de Kanban

Crear un tablero de Kanban con datos de muestra

Cancelar

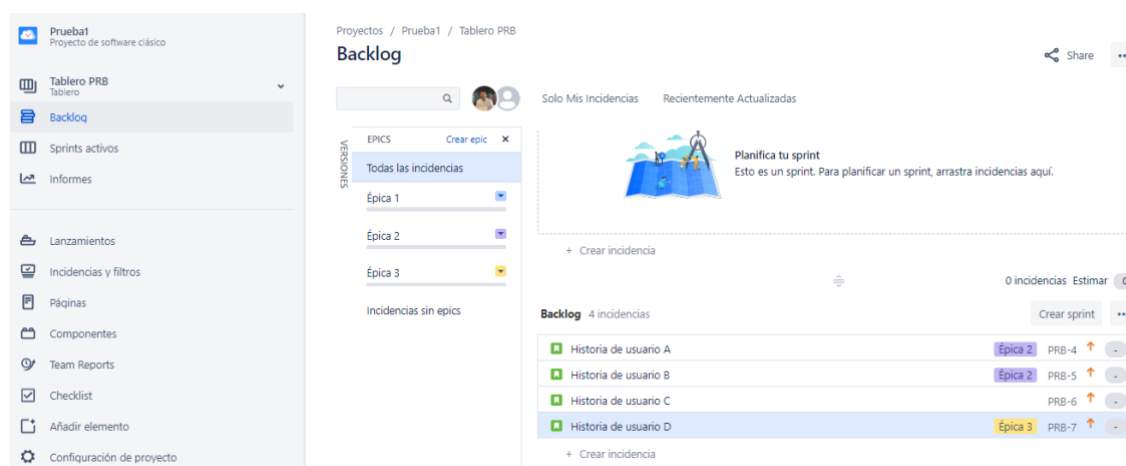
Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

La diferencia entre ambos es que en Scrum tenemos vistas del *backlog*, del *sprint backlog*, y contaremos con la opción de introducir *sprints*. En cambio, en Kanban, el trabajo se visualiza en una vista única.

Panel principal

Esta será la vista central desde donde interactuaremos con nuestro proyecto. En esta vista, podremos administrar nuestro *product backlog* y *sprint backlog* y podremos agregar o quitar ítems. También podremos abrir *sprints*, crear tareas, historias de usuario, épicas y relacionarlas entre sí.

Figura 20: Panel principal de Jira



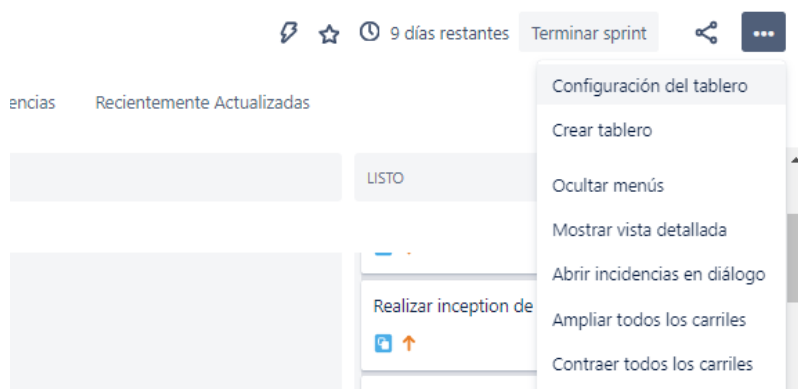
Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Tableros

Ahora vamos a configurar nuestro tablero. Esto es importante, porque será donde inspeccionaremos y visualizaremos nuestras tareas.

Lo primero que haremos es configurar las columnas y sus respectivos estados.

Figura 21: Tableros de Jira



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Figura 22: Configuración de tableros en Jira

Configuración de Tablero AE

← Regresar al tablero

CONFIGURACIÓN

General

Columnas

Carriles

Filtros rápidos

Colores de tarjetas

Diseño de tarjeta

Estimación

Días laborables

Vista de datos de incidencia

Administración de columnas

Las columnas pueden ser añadidas, eliminadas, reordenadas y renombradas. Las columnas se basan en estados globales que pueden ser movidos entre columnas. Las restricciones máximas y mínimas pueden ser definidas para cada columna asignada.

Restricción de Columna

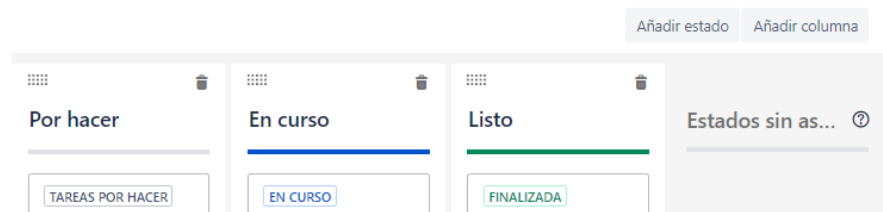
Ninguna

Es posible añadir restricciones a las columnas del tablero en relación con una estadística.

Flujo de trabajo simplificado

Usar el flujo de trabajo simplificado

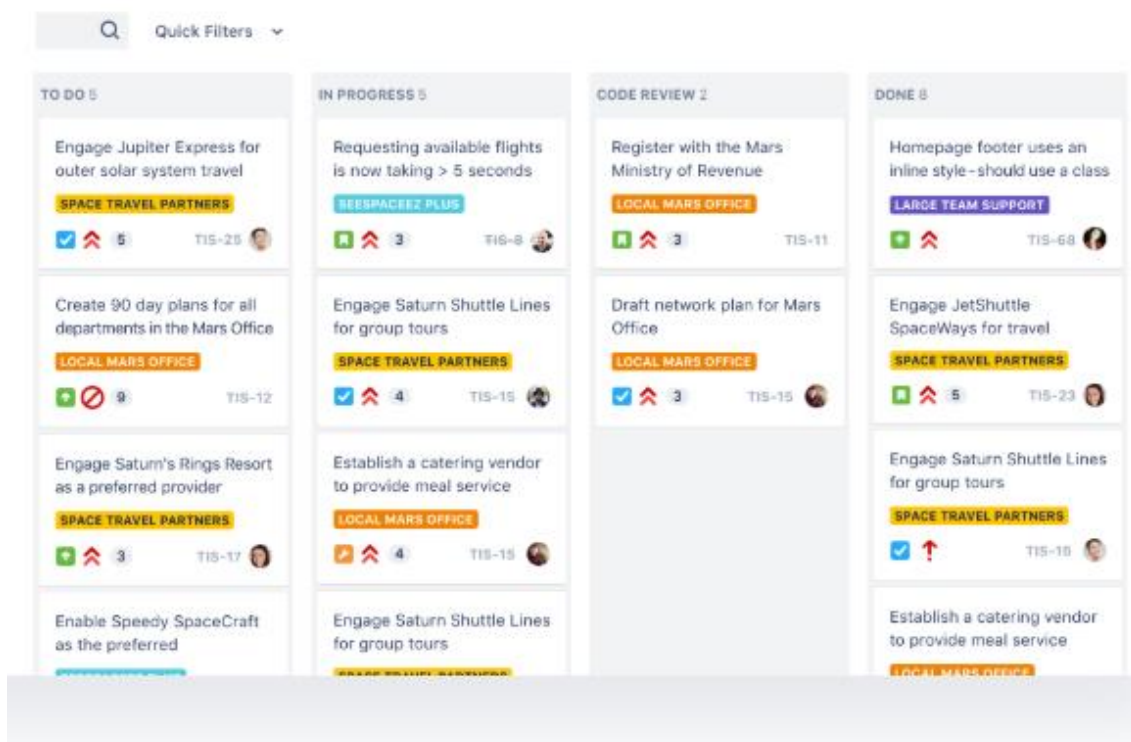
El flujo de trabajo del proyecto **Agile Evolution** está siendo administrado por Jira Software. Los administradores del proyecto pueden añadir y eliminar estados a continuación. ?



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Nuestro tablero está listo para visualizar y nuestro *sprint* está activo.

Figura 23: Tablero de jira listo para visualizar un *sprint* activo



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

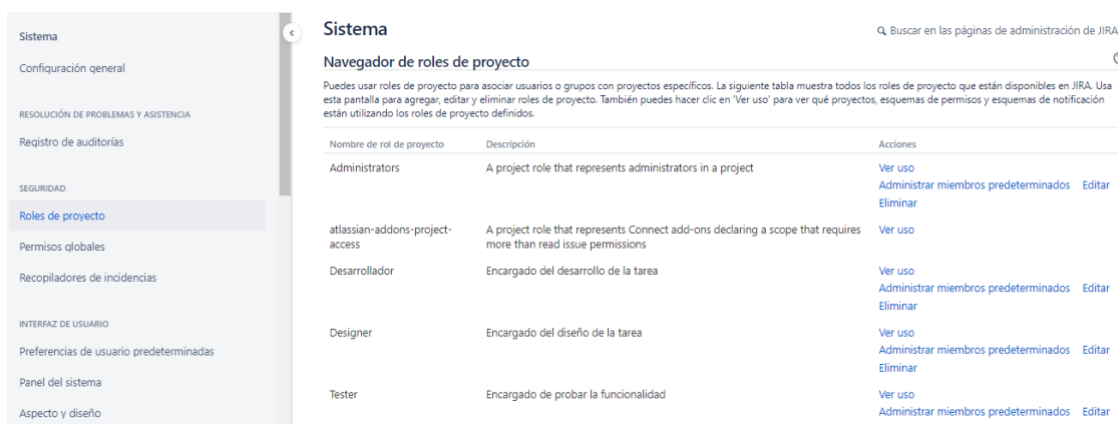
Elementos de configuración

Si ingresamos a «Menú de administración», y luego a «Jira settings», podremos configurar los distintos elementos: Workflows, Fields, IssueTypes, Screens, permisos y esquemas. Un esquema es una configuración que permite la reutilización entre varios equipos de elementos similares. A continuación, veremos cada uno de ellos.

Permisos en Jira

En el apartado de «Configuración y seguridad», podemos definir los roles para nuestro proyecto.

Figura 24: Permisos en Jira



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Aquí podremos crear los roles que necesitemos, como por ejemplo *designer*, *tester*, *product owner*, *Scrum master* y *Development Team*.

Luego, ingresamos al menú «Incidencias» y «Esquema de permisos», donde podremos definir nuestros esquemas. Un esquema de permisos define qué acciones se pueden ejecutar según el rol. Veamos un ejemplo.

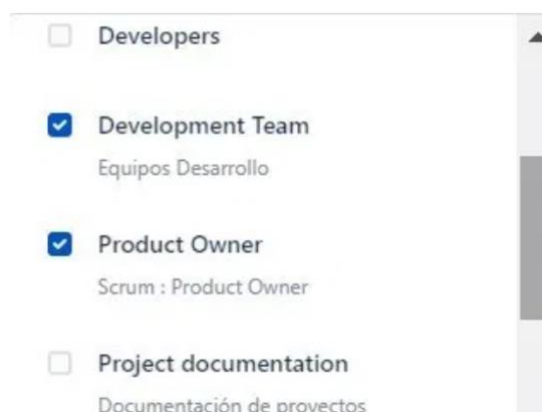
Figura 25: Esquema de permisos en Jira

Permiso	Concedido a
Usuario asignable A usuarios con este permiso se les puede asignar una incidencia.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> atlassian-addons-project-access Acceso a aplicaciones <ul style="list-style-type: none"> Cualquier usuario conectado
Asignar incidencias Capacidad para asignar incidencias a otros usuarios.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> atlassian-addons-project-access Acceso a aplicaciones <ul style="list-style-type: none"> Cualquier usuario conectado
Cerrar Incidencias Capacidad de cerrar incidencias. A menudo es útil que tus desarrolladores resuelvan incidencias y un departamento de calidad sea el encargado de realizar el cierre.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> atlassian-addons-project-access Acceso a aplicaciones <ul style="list-style-type: none"> Cualquier usuario conectado
Crear incidencias Capacidad para crear incidencias.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> atlassian-addons-project-access Acceso a aplicaciones <ul style="list-style-type: none"> Cualquier usuario conectado
Borrar incidencias Capacidad para borrar incidencias.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> Administrators atlassian-addons-project-access
Editar incidencias Capacidad para editar incidencias.	Rol de proyecto Editar Eliminar <ul style="list-style-type: none"> atlassian-addons-project-access

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

A continuación, en el proyecto, asignamos el esquema de permisos que decidamos utilizar.

Figura 26: Asignación de esquema de permisos en Jira



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Configuración de ítems de incidencias

Las incidencias representan los ítems de trabajo de Scrum. Los tipos de incidencias pueden ser asociados a campos, seguridad o *workflows* diferentes. A la hora de crear un tipo nuevo, debemos elegir entre tipo estándar o subtask.

Figura 27: Configuración de ítems de incidencias en Jira

Agregar Tipo de Incidencia

Nombre ^{*}

Descripción

Tipo

☒ Tipo de incidencia estándar

☐ Tipo de incidencia subtask

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

A continuación, veremos varios tipos de incidencia genéricos: en Scrum serían tipos de ítems del *product backlog*:

- **«Bug»:** se utiliza para errores.
- **«Improvement»:** se utiliza para mejoras sobre elementos ya existentes de nuestro producto.

- «**Feature**»: se usa para un conjunto de historias de usuario que representan una funcionalidad concreta.
- «**Document**»: se utiliza en caso que haya que generar algún tipo de documentación.
- «**Task**»: se usa para las tareas técnicas.
- «**Epic**»: las épicas son contenedores de historias.
- «**Spike**»: se utiliza para experimentos. En general, no llevan estimación.
- «**Technical Enhancement**»: la mejora técnica representa todo aquel elemento técnico que tenemos que abordar. En Scrum, se conoce como “deuda técnica”.
- «**Story**»: es un requisito funcional escrito en forma de historia de usuario.

Luego, vamos a configurar los Issue Type Schema, que son de utilidad para agrupar Issue Types, para luego ser asociados a un proyecto.

Figura 28: «Issue Type Scheme for scrum»



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Por último, debemos elegir qué esquema aplicar en nuestro proyecto.

Figura 29: Tipos de incidencias




















Tipos de incidencias

PO: Scrum Issue Type Scheme

⚙ Acciones ▾

Haz seguimiento a diferentes tipos de incidencia, tales como defectos o tareas. Puedes aplicar una configuración distinta para cada tipo de incidencia.

El esquema de tipo de incidencias define los tipos de incidencia que aplican a este proyecto. Para cambiar los tipos de incidencia utilizados, puedes seleccionar un esquema de tipo de incidencias diferente, o modificar el esquema seleccionado.

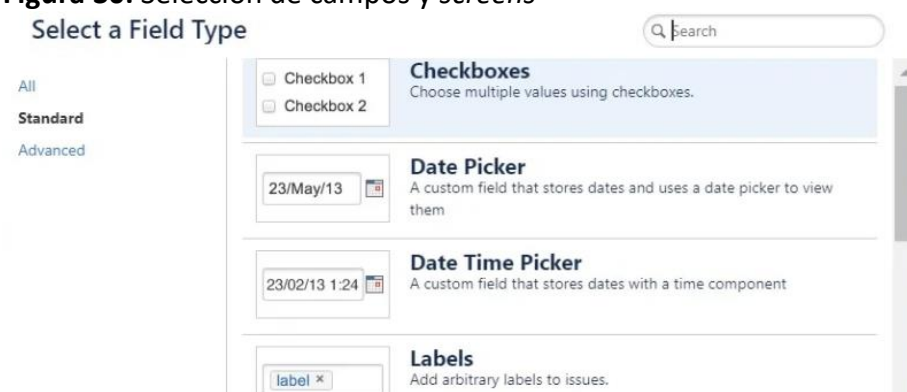
Tipo de Incidencia	Descripción	Flujo de trabajo	Configuración de campo	Pantalla
 Historia	Una función o funcionalidad expresada como objetivo del usuario.	 Software Simplified Workflow for Project PO	 Default Field Configuration	 PO: Scrum Default Screen Scheme
 Epic	Una colección de errores, historias y tareas relacionadas.	 Software Simplified Workflow for Project PO	 Default Field Configuration	 PO: Scrum Default Screen Scheme
 Error	Un problema o error.	 Software Simplified Workflow for Project PO	 Default Field Configuration	 PO: Scrum Bug Screen Scheme
<input checked="" type="checkbox"/> Tarea	Un trabajo pequeño e independiente.	 Software Simplified Workflow for Project PO	 Default Field Configuration	 PO: Scrum Default Screen Scheme
 Subtarea <small>SUBTAREA</small>	Un trabajo pequeño que forma parte de una tarea de mayor tamaño.	 Software Simplified Workflow for Project PO	 Default Field Configuration	 PO: Scrum Default Screen Scheme

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Campos y screens

Ahora definiremos qué información queremos visualizar en cada uno de ellos. Para esto, utilizaremos los *screens*, que son pantallas que se asocian con los tipos de incidencia. Un *screen*, a su vez, está compuesto de campos, y es posible personalizar dichos campos.

Figura 30: Selección de campos y *screens*



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

El siguiente paso es el Field Configurator, donde configuramos los campos. Por ejemplo, podemos marcar campos como obligatorios u ocultos.

Luego, generaremos el Field Configurator Schema. Este esquema relaciona tipos de incidencia con diferentes esquemas de campos, y puede tener campos diferentes según los distintos tipos de incidencias.

Ahora continuaremos configurando nuestras propias pantallas, y tenemos la posibilidad de definir los campos y sus pestañas.

Figura 31: Configuración de incidencias en Jira

Incidencias

Q Buscar en las páginas de administración de JIRA

Configurar pantalla

COMPARTIDO POR 1 PROYECTO

Estos son los campos, y en este orden se muestran, en el caso de las incidencias que utilizan **AD: Kanban Bug Screen**. Solo hemos mostrado los campos que no están ocultos y en los que tienes permiso para editar.

Para las ubicaciones en las que se ha habilitado **vista de incidencias nueva**

- Los campos de la primera pestaña se muestran directamente en la vista de incidencias. Las demás pestañas aparecen como paneles en la sección de datos de las incidencias.
- Los campos con texto multilínea aparecen en la sección de contenido de la vista de incidencias, salvo que los coloques en una pestaña.
- Puedes configurar el orden de los campos y las pestañas de un proyecto desde **Configuración de proyecto > Diseño de las incidencias**.

Field Tab	Añadir Pestaña
Resumen	
Tipo de Incidencia	
Componentes	
Descripción	
Informador	

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Lo siguiente que haremos será crear un esquema para relacionar operaciones (por ejemplo, la creación de una incidencia) con las pantallas.

Figura 32: Creación de esquemas en Jira

Asociar un Tipo de Incidencia con un Esquema de Pant...

Tipo de Incidencia

Historia

Esquema de pantalla

Default Screen Scheme

Añadir Cancelar

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Ahora vamos a relacionar los esquemas con los tipos de incidencias. Esto nos permite seleccionar el esquema y las operaciones que se usarán según el tipo de incidencia.

Figura 33: Relación de esquemas con tipos de incidencias en Jira

Incidencias

🔍 Buscar en las páginas de administración de JIRA

Configurar Esquema de Tipos de Incidencia a Pantallas: AE: Scrum Issue Type Screen Scheme

COMPARTIDO POR 1 PROYECTO


Asociar un tipo de incidencia con un esquema de pantallas



① Este esquema puede ser utilizado por uno o más proyectos, el [Esquema de Pantallas](#) especificado para cada tipo de incidencia será aplicado a las incidencias en estos proyectos.

La entrada *Default* especifica el Esquema de Pantalla a utilizar para cualquier tipo de incidencia que no haya sido asignada a un esquema de pantalla de manera explícita.

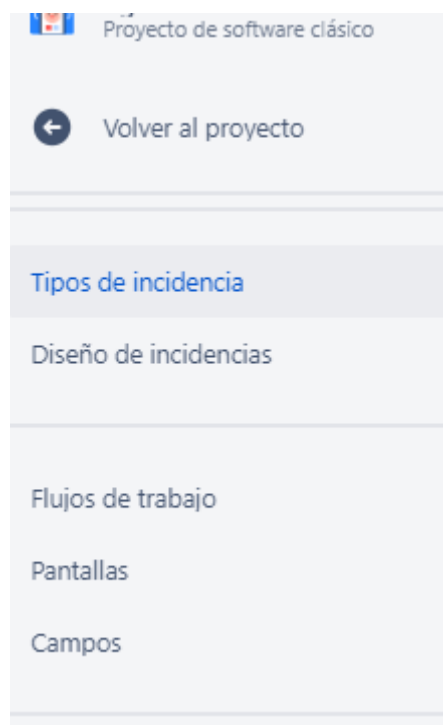
[Ver todos los esquemas de tipos de incidencia a pantallas.](#)

Tipo de Incidencia	Esquema de pantalla	Acciones
Por defecto Usado para todos los tipos de incidencia no asignados.	AE: Scrum Default Screen Scheme	Editar
 Error	AE: Scrum Bug Screen Scheme	Editar Eliminar

Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Ahora ya contamos con la configuración total sobre operaciones, información y pantallas listas para agregarlas a nuestro proyecto.

Figura 34: Lista de opciones de configuración



Fuente: captura de pantalla del *software* Jira (Atlassian, 2020).

Workflows

Los *workflows* son muy importantes a la hora de gestionar un proyecto en Jira, ya que permiten organizar cómo evolucionarán las incidencias. Los *workflows* representan esquemas formados por estados y transiciones por donde pasarán las distintas incidencias. Con el *workflow schema* decidimos qué *workflow* usarán las incidencias y lo asignaremos a un proyecto.

Figura 35: Esquema de *workflow*



Fuente: [imagen sin título sobre esquema de workflow], s.f., <https://bit.ly/38AygL9>

A continuación, definimos los términos que se presentan en el esquema:

Request: una nueva solicitud.

Refinement: incidencia que refinaremos.

Ready: incidencia lista para ser trabajada en un *sprint*.

In progress: incidencia en la cual ya estamos trabajando.

In review: incidencia seleccionada para revisión.

To deploy: incidencia lista para ser subida.

Done: hemos finalizado.

Cuando usamos Jira, debemos darles libertad a los equipos para que modelen sus propios *workflows*, dado que pueden tener necesidades diferentes según su contexto.

¿Creen que ya podemos arrancar?

Referencias

- [Imagen sin título sobre **autoorganización**], (s.f.). Recuperado de <https://smallbusiness.ng/how-to-do-a-swot-analysis-of-a-business/>
- [Imagen sin título sobre **componentes de una historia de usuario**], (s.f.). Recuperado de <https://medium.com/@henryksobczak/historias-de-usuarios-fcbd6dba29e8>
- [Imagen sin título sobre **daily scrum**], (s.f.). Recuperado de <https://callforart.net/visitorform/>
- [Imagen sin título sobre **esquema de workflow**], (s.f.). Recuperado de <https://confluence.atlassian.com/adminjiraserver/working-with-workflows-938847362.html>
- [Imagen sin título sobre **marco Scrum**], (2020). Recuperado de <https://medium.com/@rabiaatekinn/agi%CC%87le-nedi%CC%87r-1e8f4e88257>
- [Imagen sin título sobre **planning poker**], (s.f.) Recuperado de <https://thebadoc.com/f/effective-estimation-techniques>
- [Imagen sin título sobre **Product Backlog**], (2018). Recuperado de <https://medium.com/serious-scrum/the-product-backlog-7aec7daf844f>
- [Imagen sin título sobre **refinamiento del Product Backlog**], (s.f.). Recuperado de <https://beagilemyfriend.com/sprint-retrospective/>
- [Imagen sin título sobre **revisión de sprint**], (s.f.). Recuperado de <https://proyectosagiles.org/demostracion-requisitos-sprint-review/>
- [Imagen sin título sobre **roles dentro de Scrum**], (2019). Recuperado de <https://www.syntacticsinc.com/news-articles-cat/graphic-design-services/>
- [Imagen sin título sobre **Scrum Master al servicio de la organización**], (2020). Recuperado de https://medium.com/@hallmark_public_school/understanding-the-zone-of-proximal-development-762f72e0b97d
- Alaimo, D. M., Salías, M.** (2015). *Proyectos Ágiles con #Scrum. Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos* (2.da. ed.). Buenos Aires, Argentina: Kleer
- Atlassian.** (2020). Jira (8.8.0) [software de gestión de proyectos]. Australia: Atlassian.
- Avedaño, M.** (2019). *User Story Mapping - Proceso de construcción*. Recuperado de <https://www.slideshare.net/Marcoviaweb/user-story-mapping-proceso-de-construccion>
- Beck, K.** (dir.) (2001). *Manifiesto por el Desarrollo Ágil de Software*. Recuperado de <http://agilemanifesto.org/iso/es/manifesto.html>
- Buonamico, D.** (2019). *¿Qué es el Product owner en el marco de Scrum y cuáles son sus áreas de atención?* Recuperado de <https://medium.com/kleer/qu%C3%A9-es-el-product-owner-en-el-marco-de-scrum-y-cu%C3%A1les-son-sus-%C3%A1reas-de-atenci%C3%B3n-2bddbb232775>
- Cert Mind,** (2019). *SCRUM. An Agile Approach To Manage Successful Projects*. Recuperado de <https://docplayer.es/123636587-An-agile-approach-to-manage-successful-projects.html>
- IONOS,** (2019). *El modelo en cascada: desarrollo secuencial de software*. Recuperado de <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/>

- Llamas, R.** (2018). *Enséñanos tu trabajo*. Recuperado de <https://medium.com/@ricardollamas/ens%C3%A9%C3%B1anos-tu-trabajo-6a722623bbb6>
- Raya, A.** (2020). *Gestión de proyectos complejos: métodos Agile y Scrum*. Recuperado de <http://andresraya.com/gestion-de-proyectos-complejos-metodos-agile-y-scrum/>
- Royce, W. W.** (1970). *Managing the Development of Large Software Systems*. En *Proceedings IEEE WESCO* 26 pp. 328-388. Recuperado de <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf>
- Schwaber, K.** (1995). SCRUM Development Process. En Sutherland J., Casanave C., Miller J., Patel P., Hollowell G. (eds). (1995). *Business Object Design and*. Recuperado de https://link.springer.com/chapter/10.1007/978-1-4471-0947-1_11
- Schwaber, K. y Sutherland, J.** (2013). *La Guía Definitiva de Scrum: Las Reglas del Juego*. Recuperado de <https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf>
- Schwaber, K. y Sutherland, J.** (2016). *La Guía Definitiva de Scrum: Las Reglas del Juego*. Recuperado de <https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf>
- Takeuchi, H., Nonaka, I.** (1986). The New New Product Development Game. En *Harvard Business Review*. Recuperado de https://www.academia.edu/40544365/The_New_New_Product_Development_Game
- Urtanta** (s.f.). Historias de usuario scrum. Recuperado de <https://urtanta.com/historias-de-usuario/>