FILOSOFÍA AGILE Y SUS METODOLOGÍAS

I11 – Sistemas de Información

Descripción breve

Incorporación teórica-práctica en la cátedra “Sistemas de Información” de la filosofía Agile y sus metodologías, con foco en Scrum.

Valentín Silvestri

valentinsilvestri@outlook.com

Contenido

Parte 1 – Metodologías de desarrollo de Software

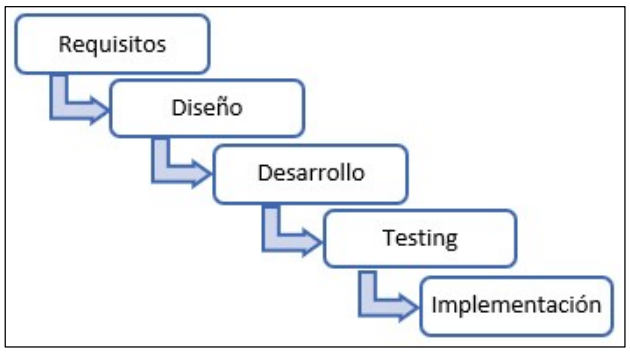
El movimiento Ágil nace como una solución a deficiencias que presentan las metodologías tradicionales utilizadas para el desarrollo de software, y el mismo se formalizo en el año 2001 a través del Manifiesto Ágil[1] como un conjunto de valores y principios[2].

A continuación, desarrollaremos las principales metodologías utilizadas en el desarrollo de software, desde sus inicios hasta llegar a las contemporáneas metodologías ágiles.

Metodologías tradicionales

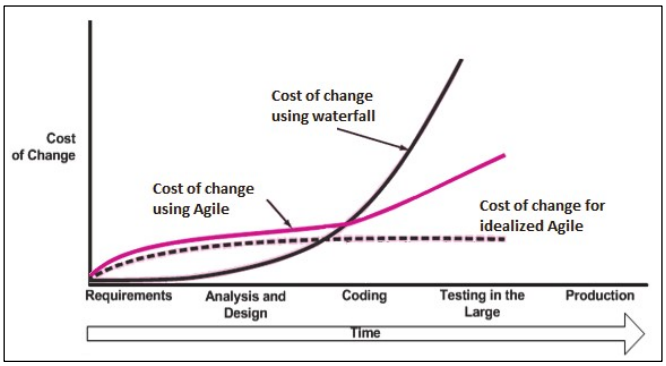
El primer enfoque utilizado para el desarrollo de software consistía en "**Prueba y Error**", donde se desarrollaba y luego se arreglaba cuando una falla era detectada. El software se entregaba terminado al final del proceso y se esperaba descubrir los errores o la insatisfacción del cliente durante su uso.

Como mejora del anterior método de trabajo surge el **Modelo en Cascada** a mediados de 1950, en el cual se definieron una serie de etapas que constituyen proceso de desarrollo de software. Estas etapas son la determinación de los Requisitos, el Diseño, el Desarrollo, las Pruebas o *Testing*, y la Implementación, donde cada etapa debe culminarse para poder continuar con las posteriores.



A pesar de su amplia adopción y su uso continuo, continúa teniendo inconvenientes con la retroalimentación entre etapas; una vez finalizada una de ellas no se puede volver hacia atrás.

Entre sus problemas se encuentra el riesgo de no cumplir a tiempo con la fecha estimada de programación, lo cual puede tener un impacto incremental en las posteriores etapas. También presenta problemas de flexibilidad, con la etapa de pruebas sobre el final del proceso. Por último, la participación del cliente es escasa y limitada.



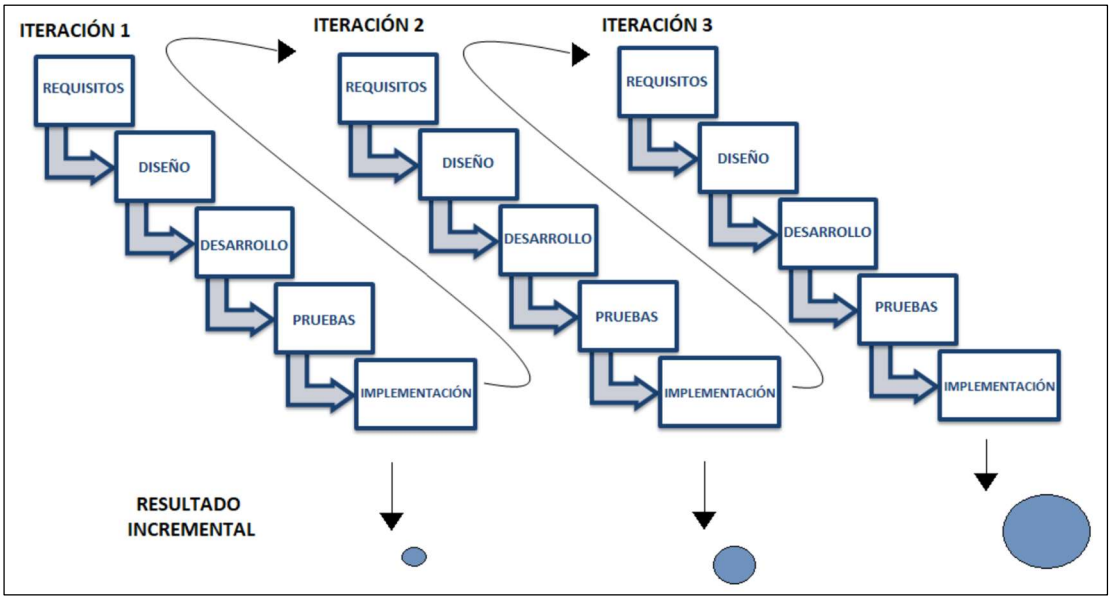
Representación de los costos de los cambios en función del tiempo: Metodologías Ágiles vs Modelo de Cascada

Algunas de las variantes más conocidas del Modelo en Cascada son el modelo **Royce**, que pone el énfasis en las etapas de requisitos y diseño, y a su vez tiene distintas fases dentro de las mismas, y el modelo **Sashimi**, en el cual existe superposición de etapas, por lo que no hay que esperar a que una etapa termine completamente para que la siguiente etapa pueda comenzar.

A fines de los 90's surgen los **Modelos Iterativos e Incrementales**, también conocidos como **Modelos en Espiral**. Los modelos iterativos buscan realizar pequeñas entregas en fragmentos cortos de manera regular, y los modelos incrementales plantean aprender de las entregas anteriores y reservar tiempo para realizar mejoras, entregando resultados que satisfacen requerimientos al final de cada iteración.

[1] *Manifiesto for Agile Software Development* – https://agilemanifesto.org/
[2] *Principles behind the Agile Manifesto* – https://agilemanifesto.org/principles.html

Estos modelos logran la retroalimentación que le falta al modelo en Cascada, la cual permite realizar cambios en los fragmentos ya entregados aprendiendo de las fallas y errores. Además, al estar en constante evaluación, la detección y corrección de errores se dan al mismo tiempo y no al final del proceso, brindando así la posibilidad de evitar grandes reprocesos, costos y tiempo innecesarios.



Representación de un Modelo Iterativo e Incremental

Dentro de este enfoque los modelos más reconocidos e implementados son el Proceso Unificado – *UP*, el Proceso Unificado Rational – *RUP* y el Proceso Unificado Ágil – *AUP*.

El **Proceso Unificado** combina las siguientes buenas prácticas:

- Ciclo de vida iterativo.
- Desarrollo dirigido por el riesgo.
- Correcta descripción del problema, consistente y documentada.
- Involucra al usuario continuamente mediante evaluación y retroalimentación.
- Se verifica constantemente la calidad.
- Se adapta al cambio y se realizan modificaciones.

UP se organiza en un conjunto de mini-proyectos cortos de duración fija, conocidos como iteraciones, que dan como resultado un producto que se puede probar y ejecutar brindando valor al trabajo final. Posee retroalimentación cíclica e incluye sus propias etapas de análisis de requisitos, diseño, desarrollo, pruebas e implementación. También es incremental porque en cada iteración se agrega valor.

A diferencia del Modelo en Cascada, el desarrollo del proyecto se lleva a cabo en 4 fases: Inicio, Elaboración, Construcción y Transición, dentro de las cuales se realizan iteraciones, excepto en la etapa de inicio.

1. *Inicio:* Se analiza el objetivo del negocio y se determina la viabilidad del proyecto junto con su alcance. En esta etapa, se identifican todas las entidades que interactúan con el proyecto y se realiza una estimación de plazos y costos, como así también de la duración de las iteraciones.
2. *Elaboración:* Se establece un plan para el desarrollo, identificando los riesgos claves del proyecto junto con nuevos requisitos y alcances. Al finalizar esta fase se tiene un modelo detallado de los requerimientos del sistema, una correcta descripción y el plan propiamente dicho para llevarlo a cabo.
3. *Construcción:* Consiste en el diseño, programación y testeo. Es una implementación iterativa del resto de los requisitos de menor riesgo y elementos más sencillos. Se realiza el producto, se llevan a cabo pruebas y se deja listo para el despliegue o entrega final.

4. *Transición:* Consiste en el traspaso del sistema desde el entorno de desarrollo al entorno del usuario final. Es una de las fases más importantes debido al costo y tiempo que requiere.

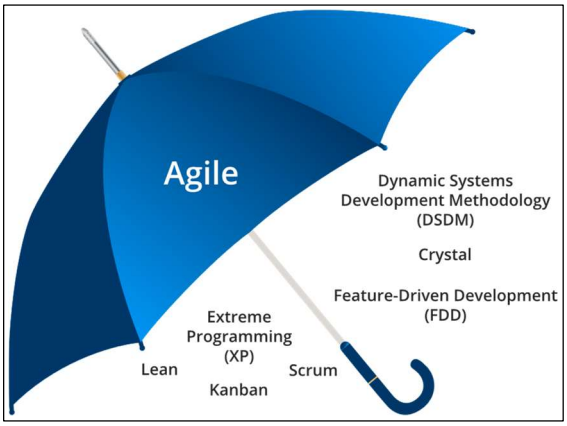
Proceso Racional Unificado es un refinamiento detallado del UP y consiste en un conjunto de documentación actualizada y detallada. Describe todos los artefactos, actividades y roles, así como proporciona guías e incluye plantillas para la mayoría de éstos.

Lo que se buscaba con UP y RUP era poder implementar un proceso adaptable, es decir que busque estar en constante actualización y modificación según los gustos o cambios de los consumidores, pero la gran cantidad de actividades y artefactos opcionales, así como también, la inexperiencia o falta de *know how* de los programadores, hicieron que tenga ciclos de desarrollos relativamente largos y que no se implemente como una herramienta ágil.

Este concepto de proceso capaz de responder a las necesidades cambiantes de la demanda en ciclos de programación cortos, comenzó a conocerse como **Proceso Unificado Ágil** y logró asentarse con cierto grado de formalidad y bases teóricas dando lugar a los Modelos Ágiles.

Metodologías Ágiles

A principios del 2000, empiezan a aparecer los Modelos Ágiles, donde ya no existe una estructura rígida de pasos para desarrollar software, sino que se trabaja en ciclos de desarrollo cortos en conjunto con el cliente y equipos multidisciplinarios, donde los requerimientos y modificaciones surgen en el día a día, con la capacidad de adaptarse al cambio, logrando una plena satisfacción del cliente, generando ganancias y ahorrando costos con entregas de valor en cada iteración, y generando un cambio de mentalidad en toda la organización.



El Manifiesto Ágil

El Manifiesto Ágil fue escrito en 2001 por un grupo de 17 desarrolladores de software que se reunieron para debatir y alinear conceptos acerca del desarrollo ágil. Plantearon sus puntos de vista y entendieron la importancia de crear un modelo en el que cada iteración del ciclo de desarrollo "aprendiera" de la iteración anterior. El resultado fue una metodología más flexible, eficiente y orientada al equipo, mucho más que cualquiera de los otros modelos anteriores.

Dicho Manifiesto consistió en la agrupación de ideales en cuatro grandes grupos definidos como valores.

Manifiesto por el Desarrollo Ágil de Software

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido 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.*

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

A su vez, se estableció una serie de 12 principios rectores de la filosofía Ágil.

Principios del Manifiesto Ágil

1- Nuestra mayor 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 software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4- Los responsables de negocio y los desarrolladores 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 funcionando es la medida principal de progreso.

8- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

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 autoorganizados.

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

A continuación, se describen las principales metodologías Ágiles de desarrollo de software:

1. SCRUM

La metodología Scrum surgió en la década del 80 y se formalizó casi 10 años después. Su principal característica es la colaboración entre los miembros del equipo y el cliente, quienes están en constante interacción y participación, haciendo que los requerimientos vayan cambiando y modificándose en todo momento.

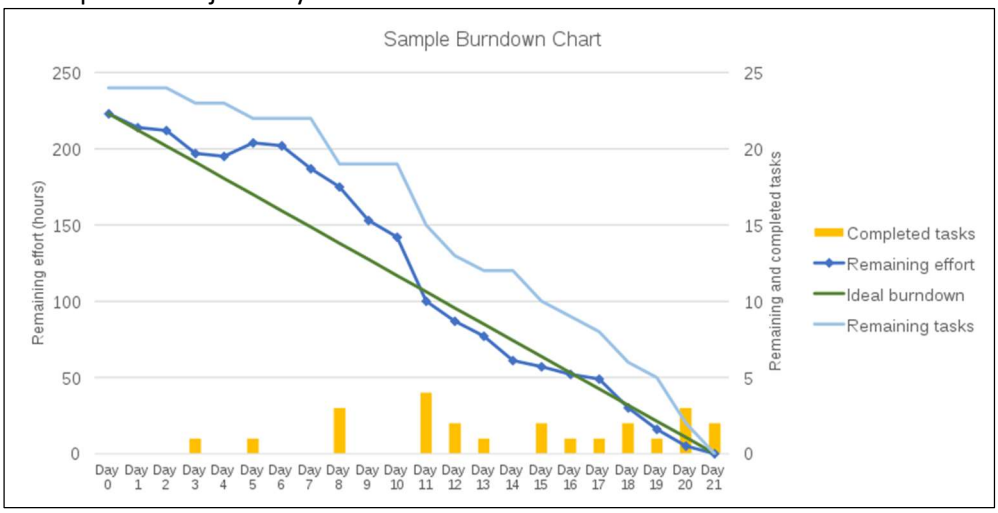
El enfoque Scrum, es un proceso de desarrollo iterativo e incremental que se caracteriza por tener ciclos cortos de trabajo, denominados *Sprints*, en los cuales se busca generar un producto entregable con un incremento de valor para el cliente.

Un *Sprint* dura de una a cuatro semanas de acuerdo al tamaño y complejidad del proyecto, pero no debe superar esta longitud, ya que se perdería el dinamismo y flexibilidad que caracteriza los procesos ágiles. Además, en Scrum, cada *Sprint* cuenta con tres elementos característicos necesarios para administrar el trabajo y dar seguimiento del progreso. Estos tres elementos son: el *Product Backlog*, el *Sprint Backlog* y el *Gráfico de Burndown*.

El *Product Backlog* es el conjunto de características y requisitos que debe tener el producto final para satisfacer al cliente. Es definido, organizado y priorizado por el *Product Owner* en conjunto con el usuario final, y va sufriendo modificaciones a lo largo de todo el proceso.

En cada *Sprint* o ciclo de trabajo se determinan un conjunto de tareas a cumplir en ese período que se obtienen del *Product Backlog*, dando lugar al *Sprint Backlog*. Estas tareas se dividen entre los miembros del equipo, quienes conocen la importancia de cumplirlas en tiempo y forma.

El *Gráfico de Burndown* es una representación gráfica del progreso del proyecto y del trabajo obtenido en cada *Sprint*. Es útil para que tanto el *Product Owner*, como el *Scrum Master* y el equipo de desarrollo, analicen si cumplen los objetivos y metas.



Representación de un *Burndown Chart* o diagrama de quemado

La formalización del proceso se realiza mediante reuniones periódicas, las cuales poseen funciones definidas y diferentes plazos de realización. Entre ellas se encuentran la *Daily Scrum*, *Sprint Planning*, la *Sprint Review* y la *Sprint Retrospective*.

La *Daily Scrum* es una reunión diaria donde todo el equipo de trabajo, junto con el *Scrum Master* y el *Product Owner*, muestran e informan en qué tareas están trabajando en ese momento. Además, se presentan las dificultades y problemas para resolver de manera colaborativa con los pares. Es una reunión que no dura más de 15 minutos.

La *Sprint Planning* es la reunión principal donde se definen qué tareas se van a realizar en el plazo de desarrollo que dura el *Sprint*. Es guiada por el *Product Owner* quien presenta las tareas del *Product Backlog* y genera el *Sprint Backlog*. Es una reunión que no supera las 8hs para un *Sprint* de un mes.

Luego de cada *Sprint*, se realizan las últimas dos reuniones, la *Sprint Review* y la *Sprint Retrospective*. La *Sprint Review* es una reunión informal donde todo el equipo analiza el trabajo realizado, si se cumplió el objetivo y el incremento de valor al producto. Además, se discute sobre cómo seguir de aquí en adelante. Mientras que en la *Sprint Retrospective*, se reflexiona sobre el trabajo realizado durante el *Sprint*, que cosas salieron de manera correcta y cuáles no. Es una reunión donde todos opinan como se sintieron al respecto, buscan formas de mejorar y planificar cambios que sean de impacto positivo.

Scrum es una buena práctica, en especial en proyectos o entornos complejos, donde es necesaria la participación constante del cliente y donde no se tiene definido con completa certeza las especificaciones del producto, lo que hace que no se pueda tener una adecuada estimación y fecha de lanzamiento. En la *Parte 2 – Scrum* se profundizará en los conceptos de esta metodología.

2. Extreme Programming – XP

XP fue introducido en el desarrollo de software en la década del 90 con el objetivo de mejorar la forma de desarrollar tareas sofisticadas y realizarlas lo más rápido posible. Este método se destaca por tener un fuerte enfoque en los aspectos técnicos del desarrollo de software. Para esto se utilizan distintas prácticas e ideas de ingeniería que ayudan a mejorar la calidad y permiten estar continuamente adaptándose al entorno. Las principales prácticas en este enfoque son:

- **Desarrollo guiado por pruebas (TDD):** Es un ciclo rápido de pruebas, codificación y refactorización. A diferencia de los ciclos de desarrollo regular, los programadores escriben las pruebas antes del código, centrando la atención en crear interfaces fáciles de usar. Esto es sumamente importante para aumentar la velocidad de desarrollo y reducir errores. También

ayuda a mejorar su diseño, documenta sus interfaces y protege contra futuros errores. El ciclo de retroalimentación es tan rápido que los errores son fáciles de encontrar y corregir.

- **Refactorización:** Es un proceso de mejora continua, donde el proceso es sólo técnico, es decir, se cambia la estructura del código y se reformula, pero no se cambia su significado o comportamiento. Se genera un código distinto que el anterior, que mejore la calidad, simplificado y aclarado, para tener luego un código fácil de mantener y extender a otras aplicaciones.
- **Integración continua:** Esta práctica es muy importante cuando se desarrolla un software de manera compartida, ya que busca eliminar los problemas de integración. Los ciclos más cortos conducen a cambios más pequeños, lo que significa que hay menos posibilidades de que los cambios de una persona se superpongan con los de otra.
- **Programación en pareja:** Para llevar a cabo esta técnica se trabaja de a dos personas (desarrolladores), mientras una realiza la codificación la otra observa de manera atenta el proceso, analiza la situación y brinda sugerencias. Lo que permite esta práctica es una gran disminución de errores ya que se basa en tareas técnicas abstractas centrándose plenamente en el código. Además, permite que se esté en constante conocimiento del proyecto entre los pares llegando a altos niveles de eficiencia a costa de un mayor tiempo de ejecución que se ve compensado a largo plazo por la calidad del software entregado.

En XP se realizan casi todas las actividades de desarrollo de software de manera simultánea, estas actividades son: Planificación, Análisis, Diseño, Codificación, Pruebas e Implementación. Por lo tanto, en lugar de trabajar con fases, lo hace mediante iteraciones, es decir, ciclos cortos de trabajo donde se busca incrementar lo desarrollado. La idea es que estos ciclos no superen la semana.

Los equipos en este enfoque son multifuncionales ya que es necesario que distintos aspectos se conozcan para que tenga éxito el proyecto. Es decir, se debe conocer en detalle cómo diseñar y programar el software, por qué el software es importante, las reglas que el software debe seguir, cómo debe comportarse el software, cómo debe verse la interfaz de usuario, dónde es probable que se encuentren los defectos, cómo interactuar con el resto de la empresa y dónde mejorar los hábitos de trabajo.

Las ventajas de estas prácticas son la alta participación del cliente, el hecho de que los ciclos iterativos cortos permiten estar actualizados y el fuerte énfasis en las prácticas técnicas.

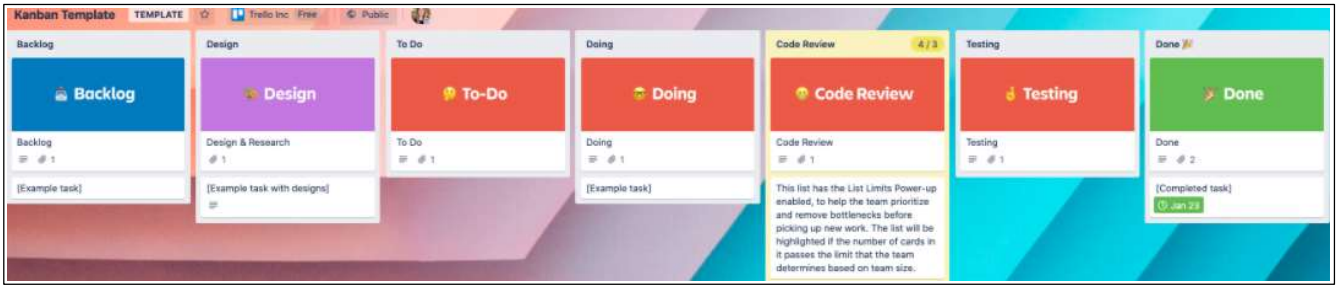
El enfoque Extreme Programming es una buena herramienta cuando se requieren demasiados detalles técnicos y un alto conocimiento del rubro. Generalmente es conveniente cuando hay que cumplir un plazo estricto de entrega. Además, es una muy buena práctica para ser combinada con los elementos componentes de Scrum generando enfoques bimodales.

3. KANBAN

Kanban se desarrolló dentro del Sistema de Producción Toyota en la década del 50, cuando Taiichi Ohno lo usó para referirse al sistema de visualización que empleaban como método de control de producción en sus procesos. En sus orígenes este método, surgió como una solución para poder coordinar en una cadena de montaje, la entrega a tiempo de cada parte, en el momento que se necesitaba, lo cual evitaba desperdicios como la sobreproducción y el almacenamiento innecesario de productos.

En la gestión ágil, se aplica Kanban como uno de los marcos de gestión de proyecto. El desarrollo del mismo se basa en la visualización del flujo de trabajo y la priorización del *Work In Progress* – WIP (trabajo en progreso), lo cual conlleva a una mayor flexibilidad en la planificación, respuestas rápidas y claridad en los objetivos. Además, al visualizar de la información mediante tableros, mejora la eficiencia en la ejecución de las tareas.

A diferencia de Scrum, no hay ciclos de trabajo con objetivos definidos, sino que se centra en hacer pequeños trabajos a medida que surgen. Kanban funciona muy bien luego del lanzamiento del producto y se suele utilizar mucho en el mantenimiento del software entregado.



Representación de un tablero Kanban en la aplicación Trello[3]

Un tablero usual y simple suele tener tres columnas, donde la primera indica el trabajo pendiente o *To Do*, es decir, el que está en cola de ejecución. La segunda columna se refiere al WIP o *In Progress* e indica las tareas que se están realizando actualmente. La tercera columna, *Done*, posee las actividades ya finalizadas. Estos tableros se pueden implementar de forma virtual o física, logrando una representación gráfica clara y sencilla para ver el estado del proyecto en cada momento.

4. Otros enfoques ágiles

- *Crystal*

La familia de metodologías *Crystal* se basa en los conceptos de *Rational Unified Process* e incluye un conjunto de metodologías diferentes para adaptarse a las diferentes circunstancias de cada proyecto individual. Se caracterizan según dos dimensiones: tamaño y criticidad. Las metodologías *Crystal* son: *Crystal Clear*, *Crystal Yellow*, *Crystal Orange* y *Crystal Red*, en las cuales el color del nombre representa el tamaño del proyecto y las personas involucradas en el mismo. En *Crystal*, el desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar.

- *Feature Driven Development – FDD*

Se centra en las fases de diseño y construcción, es decir, no cubre todo el proceso de desarrollo de software. Consta de cinco procesos secuenciales y proporciona los métodos, técnicas y pautas que se necesitan para poder entregar el producto. Por lo tanto, el enfoque FDD es un proceso iterativo que utiliza una combinación de las mejores prácticas ágiles que se consideran efectivas en el desarrollo, que enfatiza los aspectos de calidad en todo el proceso, y que incluye entregas frecuentes y tangibles, junto con un monitoreo preciso del progreso del proyecto.

- *Dynamic Systems Development Method – DSDM*

Se originó en el año 1994 con el objetivo de crear una metodología para el Desarrollo Rápido de Aplicaciones – RAD unificada. Es un proceso iterativo e incremental que fragmenta el proyecto en períodos cortos de tiempo. Define entregables para cada uno de estos períodos y el equipo de desarrollo trabaja en conjunto con el usuario. DSDM es un marco que fija el tiempo y los recursos disponibles, y luego ajusta la cantidad de funcionalidades que tendrá el producto final. Consta de cinco fases: el estudio viabilidad, el estudio del negocio, el modelado funcional, el diseño y construcción, y la implementación.

- *Adaptative Software Development – ASD*

El desarrollo de software adaptativo se centra principalmente en problemas de desarrollo de sistemas grandes y complejos. Entre sus principales características se destaca la flexibilidad frente a cambios y el desarrollo como un proceso iterativo, orientado a los componentes de software, más que a las tareas. Posee un ciclo de vida compuesto por las fases de especulación, colaboración y aprendizaje. En

[3] *Kanban 101: How Any Team Can Be More Agile* – https://blog.trello.com/kanban-101

la primera de ellas, se inicia el proyecto y se planifican las características del software; en la segunda, se desarrollan las características y finalmente en la tercera, se revisa su calidad y se entrega al cliente.

- *Híbrido*

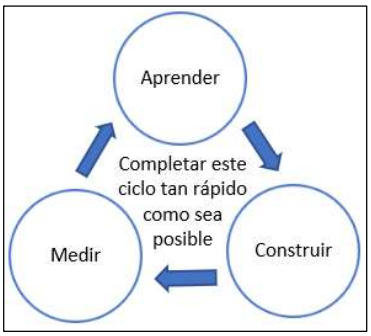
Este enfoque es una mezcla de dos modelos diferentes de desarrollo de software: Agile y Cascada. Combina el desarrollo iterativo e incremental (desarrollo flexible de Agile), con la planificación en etapas del Modelo en Cascada. Se pueden utilizar ambos enfoques en las diferentes fases del proyecto o también aplicar las prácticas ágiles en el proceso en cascada.

- *Scrumban*

Este enfoque utiliza las mejores prácticas de Scrum y Kanban para generar ventajas en el desarrollo de software. Se adiciona la visualización del flujo de trabajo y el concepto de WIP de Kanban, a las prácticas y reuniones de Scrum, lo que ayuda a una mayor comprensión del estado del proyecto y permite impulsar cambios incrementales. Generalmente se utiliza para el mantenimiento de un proyecto luego de ser entregado, como por ejemplo en el soporte de software.

- *Lean Software Development*

Al igual que Kanban, este enfoque se originó dentro del Sistema de Producción Toyota. En sus comienzos surgió como una técnica aplicada a la industria, en donde se buscó formar las bases de una manera de pensar completamente nueva sobre fabricación, logística y, finalmente, desarrollo de productos. La aplicación de los principios Lean para el desarrollo de software fue introducida en 2003. Este enfoque incluye 7 principios, los cuales son: eliminar el desperdicio, generar calidad, crear conocimiento, diferir el compromiso, entregar rápidamente, respetar a las personas y optimizar el todo.



Gestión de proyectos: Tradicional vs Ágil

Se entiende por "**Gestión Tradicional de Proyectos**" a una metodología establecida donde existen etapas definidas y fijas, y el proyecto se ejecuta en un ciclo secuencial. Esta gestión se utiliza para proyectos que tienen resultados y duración predecibles con plazos de tiempo definidos para lograr los objetivos. En general, siguen una misma secuencia de pasos: Inicio → Planificación → Ejecución → Medición.

El enfoque tradicional de proyectos hace hincapié en procesos lineales, documentación excesiva y planificación anticipada. Presenta como beneficios contar con objetivos claros y definidos, control sobre procesos y documentación adecuada.

La "**Gestión Ágil de Proyectos**" consiste en una metodología orientada al cambio, a la obtención rápida de resultados y satisfacción del cliente. Busca ser flexible en cuanto a cambios respecto a requerimientos iniciales y gasta menos tiempo en planificación anticipada. Elimina el exceso de burocracia y apunta a la simplicidad y mejora continua con entregas tempranas en cortos plazos de ejecución. Basada en la comunicación constante y la participación del cliente.

Entre los beneficios que podemos encontrar en esta gestión, se destacan la asignación flexible de prioridades, los procesos iterativos e incrementales con entregas tempranas, la mejor calidad final y una mayor transparencia.

Las principales diferencias entre la Gestión Tradicional y la Gestión Ágil pueden resumirse en el siguiente cuadro:

Características	TRADICIONAL	ÁGIL
Estructura Organizativa	Lineal	Iterativa e Incremental
Documentación	Detallada y excesiva	Adecuada y sujeta a cambios
Planificación	Anticipada y detallada	Se planifica para cortos períodos
Requisitos	Claros y definidos antes de comenzar. Contrato estricto.	Dinámicos, cambiantes. Flexibilidad en la demanda.
Modelo de Desarrollo	Ciclo de Vida	Entrega evolutiva
Participación del Cliente	Baja	Alta
Gestión del Cambio	Su naturaleza es resistirse al cambio. Todo está definido desde el inicio.	El cambio forma parte del proceso.
Gestión de Equipos de Trabajo	Pone énfasis en los procesos y herramientas más que en las personas. Objetivos individuales.	Pone énfasis en las personas. Equipos multidisciplinarios y autoorganizados. Objetivos en común.
Preferencias del Modelo	Favorece la anticipación	Favorece la adaptación

Si bien las diferencias entre los modelos de gestión presentados son claras, esto no implica que una metodología sea mejor que otra, sino que deben escogerse las metodologías a aplicar de acuerdo a las características de cada proyecto en particular.

Prácticas Ágiles

Existen diversas prácticas que se basan en el Manifiesto Ágil y son implementadas en las distintas metodologías mencionadas anteriormente. A continuación, mencionamos las principales:

1. Planificación ágil del trabajo

Los enfoques ágiles buscan realizar el trabajo de una manera eficiente, realizando entregas periódicas que aportan valor al cliente, generando un producto entregable en el corto plazo. Al igual que en el enfoque Scrum, se establecen ciclos de trabajo llamados *Sprint*, con plazos entre una y cuatro semanas. Dentro de los mismos, se divide el trabajo en fragmentos, los cuales se priorizan para ser llevados a cabo.

La planificación ágil implica programar el trabajo a realizar durante una iteración o *Sprint* y asignar tareas o elementos de trabajo individuales a todos los miembros del equipo. Como se mencionó anteriormente, en los enfoques ágiles no existe documentación en exceso, sino más bien la adecuada y la cual es posible modificar. Se evita la planificación anticipada, y los cambios y modificaciones forman parte del proceso.

Por último, los planes deben estar accesibles para todo el equipo y ser flexibles, con el objetivo de poder responder a las exigencias de la demanda.

2. Reuniones de pies o *Standups*

Consisten en reuniones diarias poco estructuradas, con el objetivo de alinear los esfuerzos del equipo, teniendo una duración máxima de 15 minutos. Los equipos utilizan estas reuniones para responsabilizarse por las tareas que va a realizar cada uno, descubrir problemas y garantizar que el trabajo fluya sin inconvenientes.

En general, todos responden las siguientes preguntas: *¿Qué completé desde la última reunión?, ¿Qué voy a realizar hasta la próxima reunión diaria?* y *¿Cuáles son mis impedimentos?*

Estas preguntas generan respuestas que permiten al equipo autoorganizarse y ser responsables con el trabajo comprometido. La tercera pregunta le brinda a cada integrante la oportunidad de alertar con anticipación sobre cualquier problema potencial. También crea una atmósfera, donde las personas no tienen miedo a hablar sobre las dificultades y se puede buscar ayuda fácilmente. Las reuniones diarias incentivan a mantener la transparencia dentro de la organización.

3. Retroalimentación y aprendizaje

Luego de cada ciclo se realizan reuniones de revisión, en las cuales el principal objetivo es detectar lo que se realizó de manera correcta y aquellas cosas que se pueden modificar o mejorar. Se tiene un espíritu de mejora continua, donde todos los miembros del equipo participan y plasman el trabajo logrado. La reunión se desempeña de manera informal, y en la misma se plantean los nuevos requisitos o retoman aquellos que no fueron terminados. Se realizan debates y se definen las historias de usuarios a desarrollar, y los líderes, o quienes guían estas reuniones, recopilan información del equipo.

El objetivo de estas reuniones es mejorar continuamente los procesos de acuerdo con las necesidades de cada miembro del equipo, lo que hace aumentar su moral, mejorar la eficiencia y aumentar el rendimiento del trabajo.

4. Historia de usuarios o *User Story*

Los enfoques ágiles adoptan la práctica de escribir breves descripciones, acerca de las piezas y partes necesarias, y de las especificaciones de lo que el cliente o usuario quiere. Por lo tanto, una historia de usuario es una descripción simple de un requisito de producto y debe tener, como mínimo, las siguientes partes: título, responsable, acción a realizar y beneficio a obtener.

La historia de usuario es lo suficientemente general como para que los desarrolladores decidan cómo van a realizar la tarea y encontrar distintas formas de entregar valor al usuario. Además, estos

registros, generan que los clientes se involucren más en el proceso de desarrollo, ya que les permiten describir en forma general lo que buscan del producto sin tener que conocer detalles específicos de sistemas y tecnología.

Otra información extra que pueden tener las historias de usuarios es "cuando" se realiza la acción, algún número de identificación de la característica, su valor y esfuerzo que conlleva y la persona a cargo de realizarla, entre otras.

En el *Anexo I – User Stories y conceptos relacionados* se profundizará en esta herramienta, aplicada principalmente a Scrum.

5. Estimación de trabajo y velocidad de seguimiento

Los miembros del equipo ágil deben estimar el trabajo a realizar para saber si el alcance de la iteración es el correcto, para ello estiman su trabajo mediante un proceso conocido como "Sistema de puntos". Estos puntos representan un enfoque de estimación basado en el tamaño y la complejidad relativa de las tareas. Estos puntos se asignan en números enteros, por ejemplo: 1, 2, 3; representando la capacidad del equipo y se definen entre todos, por lo cual, no serían aplicables a otro equipo de trabajo. Las tareas pequeñas y simples son tareas de un punto, las tareas ligeramente más grandes o complejas son tareas de dos puntos, y así sucesivamente. Realizar una buena estimación es fundamental, ya que cuando se establece la lista de tareas a cumplir en una iteración se debe corroborar que ésta no sea tan grande como para no poder completarla en un ciclo.

En cuanto a la velocidad de seguimiento, al finalizar cada iteración se analizan y contabilizan la cantidad de requisitos que fueron finalizados, sumándose los puntos de historias completadas. Así, se obtiene la velocidad del equipo y a lo largo del proyecto se puede ver la tendencia y calcular el promedio de la misma.

Los equipos ágiles utilizan una variedad de técnicas de estimación, que incluyen el mapeo de afinidad, el sistema de cubetas y la planificación del póker[4].

[4] *Planning Poker – Agile Alliance: https://www.agilealliance.org/glossary/poker/*

Parte 2 – Scrum

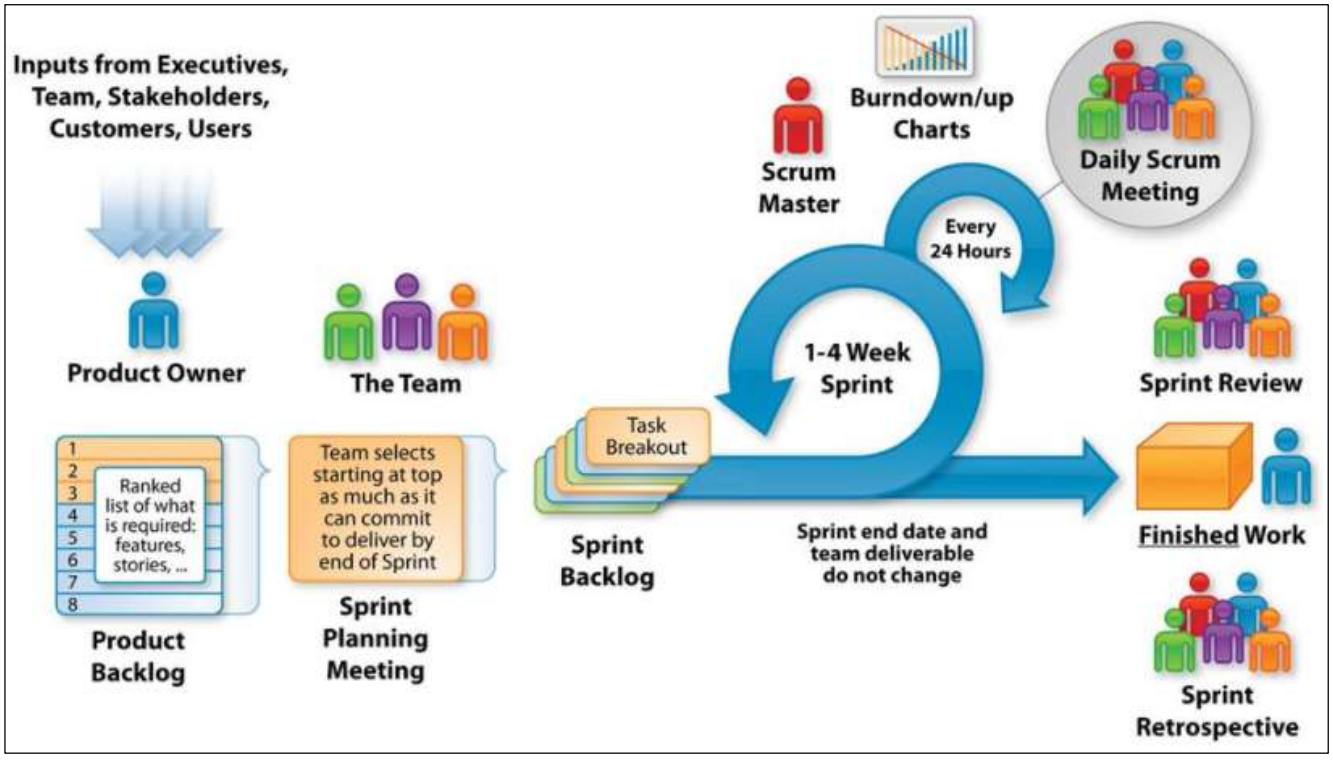
Scrum surge en 1986 con un artículo publicado por Hirotaka Takeuchi y Ikujiro Nonaka en la revista Harvard Business Review, titulado *"The New New Product Development Game"* [5]. El artículo es el resultado del estudio de los sistemas de producción de algunas de las principales empresas manufactureras de la década del 80, y en las cuales se encontraron seis puntos de coincidencia:

1. La responsabilidad recae en el equipo.
2. Los equipos se auto organizan.
3. Las fases de desarrollo se solapan entre sí.
4. El equipo es de carácter multidisciplinar.
5. Hay un control sutil destinado a la selección del personal y puntos de control.
6. Existe una trasferencia de conocimiento entre distintos equipos.

A principios de los años 90's Ken Schwaber comenzó a utilizar las primeras aproximaciones de este método, mientras que, paralelamente, Jeff Sutherland comenzó a desarrollar un enfoque similar en la Easel Corporation. En 1995 ambos presentaron estos conceptos en la Conferencia OOPSLA y publicaron un informe describiendo la metodología[6], en 2010 escribieron la primera versión de la Guía Scrum[7].

¿Qué es Scrum?

Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. Scrum es un proceso iterativo e incremental utilizado para la construcción de productos, un proceso que se compone de diferentes iteraciones denominadas *Sprints*. Estos *Sprints* tienen un tiempo establecido y un objetivo, que es construir un producto o un incremento de producto que pueda ser utilizado por los clientes.



Representación de los roles, artefactos y reuniones, así como de la dinámica de trabajo en Scrum.

[5] *"The New New Product Development Game" por Hirotaka Takeuchi and Ikujiro Nonaka (1986).* https://hbr.org/1986/01/the-new-new-product-development-game

[6] *"Scrum Development Process" por Ken Schwaber, Jeff Sutherland, et al. (1997).* http://www.jeffsutherland.org/oopsla/schwapub.pdf

[7] *"La Guía Definitiva de Scrum: Las Reglas de Juego" por Ken Schwaber, Jeff Sutherland (2020).* https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

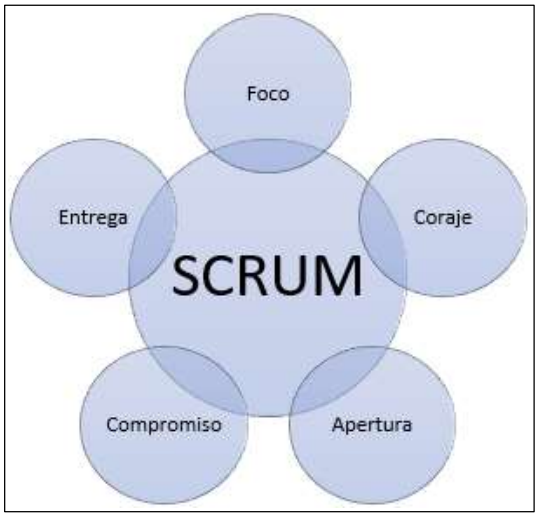
Valores ágiles analizados desde la perspectiva de Scrum

Scrum es el modelo más utilizado dentro de las Metodología Ágiles, teniendo muchos de los valores y principios del Manifiesto Ágil su origen en Scrum.

1. ***Individuos e interacciones sobre procesos y herramientas.*** Scrum se apoya en la confianza hacia las personas, sus interacciones y los equipos. Los equipos identifican lo que hay que hacer y toman la responsabilidad de hacerlo, removiendo todos los impedimentos que encuentren en su camino y estén a su alcance. Los equipos trabajan en conjunto con otras partes de la organización cuando los impedimentos están fuera de su ámbito de control.
2. ***Software funcionando sobre documentación exhaustiva.*** Scrum requiere que al final de cada Sprint se obtenga un incremento funcional potencialmente entregable. La documentación es entendida como un producto intermedio sin valor de negocio; por lo que los equipos pueden documentar, pero ninguno de estos documentos puede ser considerado como el resultado de un *Sprint*. El progreso del proyecto se mide en base al producto funcionando que se entrega iterativamente.
3. ***Colaboración con el cliente sobre la negociación contractual.*** El *Product Owner* es el responsable de la relación que existe con los stakeholders, los usuarios finales y otros interesados, pudiendo a su vez estos participar en la *Sprint Review*. El *Product Owner* es parte del *Scrum Team* y trabaja colaborativamente con el resto de los miembros del equipo para asegurarse que el producto construido tenga la mayor cantidad posible de valor al final de cada Sprint.
4. ***Respuesta ante el cambio sobre seguir un plan.*** Scrum asegura que todos dentro del equipo tengan toda la información necesaria para poder tomar decisiones informadas sobre el proyecto en cualquier momento. El progreso es medido al final de cada *Sprint* mediante software funcionando y la lista de características pendientes está visible continuamente y para todos los miembros. Esto permite que el alcance del proyecto cambie constantemente en función de la retroalimentación provista por los stakeholders.

Valores de Scrum

Scrum se construye sobre 5 pilares, sus valores. Estos valores dan dirección al *Scrum Team* con respecto a su trabajo, acciones y comportamiento:



1. ***Foco*.** Los miembros de un equipo se enfocan en un conjunto acotado de características por vez. Esto permite que al final de cada *Sprint* se entregue un producto de calidad, reduciendo a su vez el *time to market.*
2. ***Coraje*.** Los *Scrum Teams* trabajan como verdaderos equipos, apoyándose entre compañeros, pudiendo así asumir compromisos desafiantes que les permiten crecer como profesionales y como equipo.
3. ***Apertura*.** En Scrum se privilegia la transparencia y la discusión abierta de los problemas. No hay agendas ocultas ni triangulación de conflictos. La sinceridad se agradece y la información está disponible para todos, todo el tiempo.
4. ***Compromiso*.** Los miembros de un equipo tienen mayor control sobre sus actividades, por eso se espera de su parte el compromiso profesional para convertir las metas en logros.
5. ***Respeto*.** Debido a que los miembros de un *Scrum Team* trabajan de forma conjunta, compartiendo éxitos y fracasos, se fomenta el respeto mutuo.

El Scrum Team: los roles y sus responsabilidades

La unidad fundamental de Scrum es un pequeño equipo de personas, conocido como *Scrum Team*, que consta de tres roles, los cuales fueron elegidos para dividir de manera clara y simple diferentes perspectivas y responsabilidades, estos roles son el *Scrum Master*, el *Product Owner* y los *Developers*. Dentro de este equipo no hay subequipos ni jerarquías, sino que es una unidad cohesionada de profesionales enfocados en un objetivo a la vez.

Los desarrolladores se encuentran apoyados en dos roles: el *Scrum Master* y el *Product Owner*. El *Scrum Master* es quien vela por la utilización de Scrum, la remoción de impedimentos y asiste al equipo a que logre su mayor nivel de performance posible. El *Product Owner* es quien representa al negocio, stakeholders, clientes y usuarios finales, y es quien tiene la responsabilidad de conducir al equipo de desarrollo hacia el producto adecuado.

En pocas palabras, Scrum requiere un *Scrum Master* para fomentar un entorno donde:

1. Un *Product Owner* ordene el trabajo de un problema complejo en *un Product Backlog.*
2. El *Scrum Team* convierta una selección del trabajo en un Increment de valor durante un *Sprint*.
3. El *Scrum Team* y sus interesados inspeccionen los resultados y se adapten para el próximo Sprint.
4. Se repita el proceso.

Product Owner

El *Product Owner* es responsable de maximizar el valor del producto resultante del trabajo del *Scrum Team*, y es el responsable del éxito del producto desde el punto de vista de los stakeholders. La forma en que esto se hace puede variar ampliamente entre organizaciones, *Scrum Teams* e individuos.

El *Product Owner* también es responsable de la gestión efectiva del *Product Backlog*, lo que incluye:

- Desarrollar y comunicar explícitamente el objetivo del producto.
- Crear y comunicar claramente los elementos del *Product Backlog*.
- Ordenar los elementos del *Product Backlog*.
- Asegurarse de que el *Product Backlog* sea transparente, visible y se entienda.

Para que los *Product Owners* tengan éxito, toda la organización debe respetar sus decisiones. Estas decisiones son visibles en el contenido y orden del *Product Backlog*, y a través del *Increment* inspeccionable en la *Sprint Review*.

Teniendo esto en cuenta, sus principales responsabilidades son:

- Determinar la visión del producto, recolectar los requerimientos, determinar y conocer en detalle las características funcionales de alto y de bajo nivel.
- Gestionar las expectativas de los stakeholders mediante la comprensión completa de la problemática de negocio y su descomposición hasta llegar al nivel de requerimientos funcionales.
- Generar y mantener el plan de entregas, las fechas y contenidos de cada entrega.
- Maximizar la rentabilidad del producto a través de la priorización, reordenando los elementos del *Product Backlog* para que el equipo de desarrollo construya con mayor anticipación las características o funcionalidades más requeridas por el mercado o los stakeholders.
- Determinar las prioridades de cada una de las características y cambiarlas según avanza el proyecto, acompañando así los cambios en el negocio.
- Aceptar o rechazar el producto construido durante el *Sprint* y proveer *feedback* para su evolución.
- Participar de la revisión del *Sprint* junto a los miembros del Equipo de Desarrollo para obtener *feedback* de los stakeholders.

Developers o Equipo de Desarrollo

El equipo de desarrollo está formado por todos los individuos necesarios para la construcción del producto en cuestión.

El mismo se caracteriza por ser *autoorganizado*. Esto significa que no existe un líder externo que asigne las tareas ni que determine la forma en la que serán resueltos los problemas. Es el mismo equipo quien determina la forma en que realizará el trabajo y cómo resolverá cada problemática que se presente.

También una de sus características es la *multifuncionalidad*, ya que cada uno de los developers del equipo debe poseer las habilidades necesarias para realizar el trabajo requerido. Lo que se espera de un developer es que no solo realice las tareas en las cuales se especializa, sino también todo lo que esté a su alcance para colaborar con el éxito del equipo.

A su vez, se caracterizan por ser *de reducido tamaño*, lo cual les permite seguir siendo ágiles. Generalmente un *Scrum Team* consta de no más de 10 personas, teniendo en cuenta los 3 roles.

Las habilidades específicas que necesitan los *Developers* suelen ser amplias y varían según el ámbito de trabajo. De igual modo, los *Developers* son siempre responsables de:

- Crear un plan para el *Sprint*, el *Sprint Backlog*.
- Inculcar calidad al adherirse a una *Definition of Done*, o Definición de Terminado.
- Adaptar su plan cada día hacia el objetivo del *Sprint*.
- Responsabilizarse mutuamente como profesionales.

Las tres responsabilidades fundamentales de los developers son 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 conjunto determinado de características en el tiempo que dura el mismo y, finalmente, ser responsables por la entrega del producto terminado al finalizar cada *Sprint*.

Scrum Master

Los *Scrum Master* son responsables de lograr la efectividad del *Scrum Team*, son líderes que sirven al equipo y a la organización en general. Lo hacen apoyando al *Scrum Team* en la mejora de sus prácticas, ayudando a todos a comprender la teoría y la práctica de Scrum.

El *Scrum Master* sirve al *Scrum Team* de varias maneras, que incluyen:

- Guiar a los miembros del equipo en ser autogestionados y multifuncionales.
- Ayudar al *Scrum Team* a enfocarse en crear *Increments* de alto valor que cumplan con la Definición de Terminado.
- Procurar la eliminación de impedimentos para el progreso del *Scrum Team*.
- Asegurarse de que todos los eventos de Scrum se lleven a cabo, sean positivos y productivos.

El *Scrum Master* sirve al *Product Owner* de varias maneras, que incluyen:

- Ayudar a encontrar técnicas para una definición efectiva de Objetivos del Producto y la gestión del *Product Backlog*.
- Ayudar al *Scrum Team* a comprender la necesidad de tener elementos del *Product Backlog* claros y concisos.
- Ayudar a establecer una planificación empírica de productos para un entorno complejo.
- Facilitar la colaboración de los interesados según se solicite o necesite.

El *Scrum Master* sirve a la organización de varias maneras, que incluyen:

- Liderar, capacitar y guiar a la organización en su adopción de Scrum.
- Planificar y asesorar implementaciones de Scrum dentro de la organización.
- Ayudar a los empleados e interesados a comprender y aplicar un enfoque empírico para el trabajo complejo.
- Eliminar las barreras entre los interesados y los miembros de los *Scrum Teams*.

Se espera que el *Scrum Master* acompañe al equipo de trabajo en su día a día y garantice que todos, incluyendo al *Product Owner*, comprendan y utilicen Scrum de forma correcta.

Bajo estas consideraciones, las principales responsabilidades del *Scrum Master* son:

- Velar por el correcto uso y evolución de Scrum.
- Facilitar el uso de Scrum a medida que avanza el tiempo. Esto incluye la responsabilidad de que todos asistan a tiempo a las *Daily Meetings*, *Reviews* y *Retrospectives*.
- Asegurar que el equipo de desarrollo sea multifuncional y eficiente.
- Proteger al equipo de desarrollo de distracciones y trabas externas al proyecto.
- Detectar, monitorear y facilitar la remoción de los impedimentos que puedan surgir con respecto al proyecto y a la metodología.
- Asegurar la cooperación y comunicación dentro del equipo.
- Detectar problemas y conflictos interpersonales dentro del equipo de trabajo.

Stakeholders

Es el conjunto de personas que no forman parte directa del *Scrum Team*, pero que deben ser tenidos en cuenta por ser personas interesadas en el desarrollo, tales como directores, gerentes, personal de la empresa dueña del producto, clientes, etc.

Es importante diferenciar a los usuarios, aquellos que utilizarán el producto a desarrollar en su día a día, de los clientes, los destinatarios finales del producto, quienes constituyen el público objetivo.

Es el *Product Owner* el que se mantiene en contacto con ellos para conocer en todo momento sus inquietudes y necesidades, y para mantener el *Product Backlog* actualizado y listo para comenzar el siguiente *Sprint*.

Artefactos o elementos de Scrum

El proceso de Scrum posee una mínima cantidad necesaria de elementos formales para poder llevar adelante un proyecto de desarrollo, los artefactos de Scrum, que representan trabajo o valor. Estos están diseñados para maximizar la transparencia de la información clave.

Cada artefacto contiene un compromiso para garantizar que proporcione información que mejore la transparencia y el enfoque frente al cual se pueda medir el progreso:

- Para el *Product Backlog*, el compromiso es el Objetivo del Producto.
- Para el *Sprint Backlog*, el compromiso es el Objetivo del Sprint.
- Para el *Increment*, el compromiso es la Definición de Terminado.

Estos compromisos existen para reforzar los valores de Scrum para el Scrum Team y sus interesados.

Product Backlog

El primer paso en la estimación y planificación ágil es la creación del Product Backlog, el cual debe ser dividido en una serie de Objetivos para trabajar. Estos objetivos suelen ser *User Stories* o historias de usuario, donde cada historia es un requerimiento de nuestro proyecto visto desde el punto de vista de un stakeholder; estas serán desarrolladas en profundidad en el *Anexo I – User Stories*.

El *Product Backlog* es básicamente un listado de ítems (*Product Backlog Ítems*, PBIs o características del producto a construir) mantenido y priorizado por el *Product Owner*. Es importante que exista una clara priorización, ya que es esta priorización la que determinará el orden en el que el equipo de desarrollo transformará las características/ítems en un producto funcional acabado.

Esta priorización es responsabilidad exclusiva del *Product Owner* y, aunque el equipo de desarrollo pueda hacer sugerencias o recomendaciones, es el *Product Owner* quien tiene la última palabra sobre la prioridad final de los ítems del *Product Backlog*, teniendo en cuenta el contexto de negocio, el producto mismo y el mercado en el que está inserto. A continuación, se presentan dos formas de priorizar los *Product Backlog Ítems*.

Priorización por valor de negocio de cada PBI

Una forma de priorizar los ítems del *Product Backlog* es según su valor de negocio. Podemos entender el valor de negocio como 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

Un enfoque diferente de medir la prioridad de un determinado ítem del *Backlog* es a través del cálculo del beneficio económico que se obtendría en función de la inversión que se debe realizar. Esto, si bien es una simple fórmula matemática ($ROI = \frac{Valor\ de\ Negocio}{Costo}$, donde el costo representa el esfuerzo necesario para la construcción de una determinada característica de un producto y el valor de negocio es el rédito económico obtenido por su incorporación), tiene implícita la problemática de conocer el valor económico que se ganaría por incorporar una determinada característica a un producto.

Prioridades según la importancia y el riesgo de cada PBI

Ya sea que los ítems del *Backlog* se prioricen por valor de negocio o por ROI, éstos pueden verse complementariamente afectados por el nivel de riesgo asociado a cada uno de ellos.

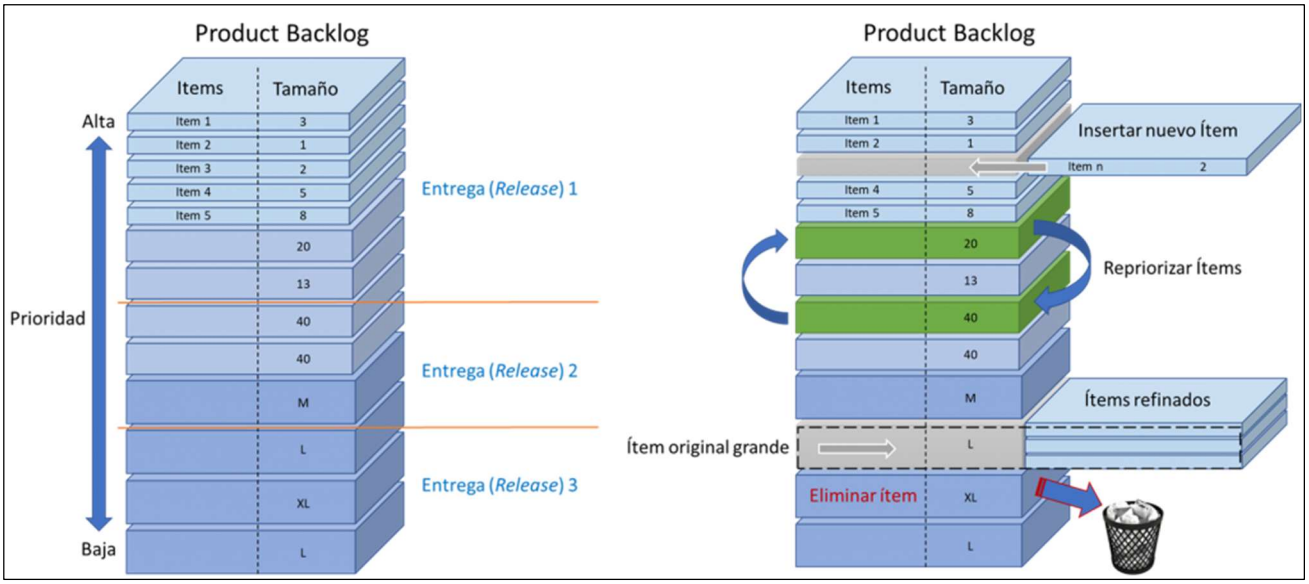
De esta manera, deberíamos aprovechar la construcción iterativa y evolutiva de Scrum para mitigar riesgos en forma implícita: construyendo primero aquellas características con mayor riesgo asociado y dejando las que poseen menor riesgo para etapas posteriores.

Se recomienda que los PBIs de baja importancia y alto riesgo sean evitados, por ejemplo, transfiriéndolos o eliminándolos del alcance.

Compromiso: *Product Goal u Objetivo del Producto*

El Objetivo del Producto describe un estado futuro del producto que puede servir como un objetivo para que el Scrum Team planifique, es el objetivo a largo plazo de este equipo. Este *Product Goal* está en el *Product Backlog*, y el resto del *Product Backlog* emerge para definir *qué* cumplirá con el Objetivo del Producto.

> *"Un producto es un vehículo para entregar valor. Tiene un límite claro, personas interesadas conocidas, usuarios o clientes bien definidos. Un producto puede ser un servicio, un producto físico o algo más abstracto".*



Representación del Product Backlog.[8]

[8] *Product Backlog y Sprint Backlog:* https://muyagile.com/product-backlog-y-sprint-backlog/

Alcance y manejo de contingencias

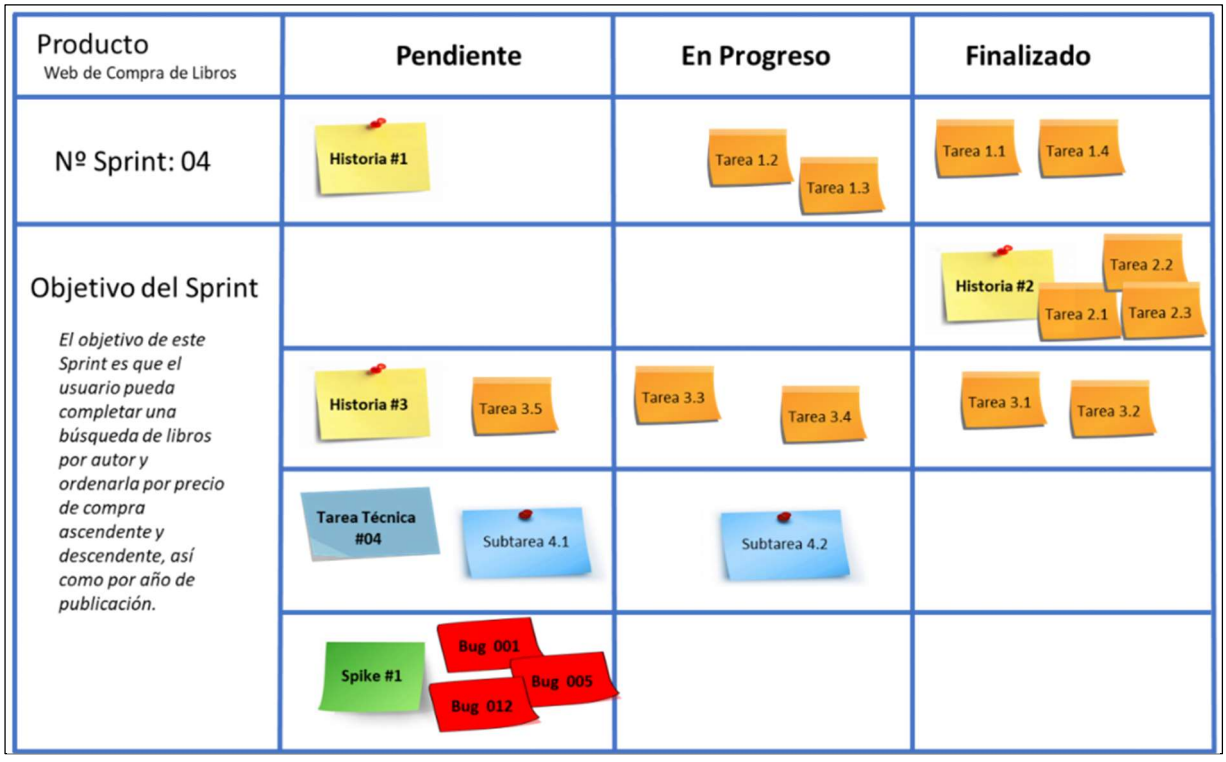
Debido a que asumimos que no nos es posible conocer de manera anticipada y con un nivel muy fino de detalle todas las características del producto que pretendemos construir, sino que es un viaje de descubrimiento que emprendemos junto con el cliente, este *Backlog* es un elemento vivo que muta a través del tiempo, al ritmo que vamos aprendiendo sobre el mismo con las entregas iterativas y el feedback frecuente.

Aprovechando que el alcance es variable y que todo lo que se debe realizar está priorizado en el *Product Backlog* según su impacto en el negocio, se utilizan las características menos prioritarias como la contingencia del proyecto frente a imprevistos. Esto quiere decir que, al respetar tiempo y costo, el alcance de menor prioridad sería el que pagaría el precio de retrasos o desvíos. Para que este enfoque sea eficaz, es fundamental la labor del *Product Owner* y su habilidad para facilitar el descubrimiento de las prioridades por parte de todos los involucrados.

Sprint Backlog

El *Sprint Backlog* se compone del *Sprint Goal* (*el por qué*), el conjunto de elementos del *Product Backlog* seleccionados para el *Sprint* (*el qué*), así como un plan de acción para entregar el Increment (*el cómo*). Es decir, es el conjunto de PBIs que fueron seleccionados para trabajar en ellos durante un cierto *Sprint*, conjuntamente con las tareas que el equipo de desarrollo ha identificado que debe realizar para poder crear un incremento funcional potencialmente entregable al finalizar el *Sprint*.

El *Sprint Backlog* es un plan realizado por y para los *Developers*. Es una imagen visible y en tiempo real del trabajo que los Developers planean realizar durante el *Sprint* para lograr el Objetivo del Sprint. En consecuencia, el *Sprint Backlog* se actualiza a lo largo del *Sprint* a medida que se aprende. Debe tener suficientes detalles para que puedan inspeccionar su progreso en la *Daily Scrum*.



Ejemplo de un Sprint Backlog en un Task Board[9] del tipo *To Do – In Progress – Done*.

Compromiso: *Sprint Goal u Objetivo del Sprint*

El Objetivo del Sprint es el único propósito del Sprint. Si bien el *Sprint Goal* es un compromiso de los *Developers*, proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo. Este

[9] *Task Board – Agile Alliance:* https://www.agilealliance.org/glossary/taskboard/

objetivo también crea coherencia y enfoque, lo que alienta al *Scrum Team* a trabajar en conjunto en lugar de en iniciativas separadas.

Es creado durante la *Sprint Planning* y es agregado al *Sprint Backlog*. A lo largo del *Sprint*, los *Developers* trabajan con el *Sprint Goal* en mente, y si el trabajo resulta ser diferente de lo que esperaban, colaboran con el *Product Owner* para negociar el alcance del *Sprint Backlog* dentro del *Sprint* sin afectar el *Sprint Goal*.

Increment

El resultado de cada *Sprint* debe ser un incremento funcional potencialmente entregable, por lo que un Increment es un peldaño concreto hacia el *Product Goal*.

Incremento funcional porque es una característica funcional nueva o modificada de un producto que está siendo construido de manera evolutiva. El producto crece con cada *Sprint*.

Potencialmente entregable porque cada una de estas características se encuentra lo suficientemente validada y verificada como para poder ser desplegada en producción o entregada a usuarios finales si así el negocio lo permite o el cliente lo requiere.

Cada Increment se suma a todos los *Increments* anteriores y se verifica minuciosamente, lo que garantiza que todos los *Increments* funcionen juntos. Para proporcionar valor, el Increment debe ser utilizable.

El trabajo no puede considerarse parte de un Increment a menos que cumpla con la Definición de Terminado.

Compromiso: *Definition of Done o Definición de Terminado*

La Definición de Terminado es una descripción formal del estado del Increment cuando cumple con las medidas de calidad requeridas para el producto. En el momento en que un PBI cumple con la Definición de Terminado, tiene lugar un *Increment*.

La dinámica de trabajo

Scrum es un proceso de desarrollo incremental e iterativo, lo cual significa que el producto se construye en incrementos funcionales entregados en períodos cortos para obtener feedback frecuente.

Una de las decisiones que debemos tomar al comenzar un proyecto es la duración de los *Sprints*, los cuales generalmente duran entre 1 y 4 semanas. Luego, el objetivo será mantener esta duración constante a lo largo del desarrollo del producto, lo que implicará que la duración de las iteraciones no cambie una vez que sea establecida.

Podemos encontrar situaciones en donde el equipo de desarrollo se atrasa o adelanta. En estos casos, la variable de ajuste es el alcance del *Sprint*, esto es, en el caso de adelantarnos deberemos incrementar el alcance del Sprint agregando nuevos PBIs y reducirlo en el caso de retrasarnos.

Sprint Planning Meeting o Planificación de Sprint

Al comienzo de cada *Sprint* se realiza una reunión de planificación del *Sprint* donde son generados los acuerdos y compromisos entre el equipo de desarrollo y el *Product Owner* sobre el alcance del *Sprint*.

Esta reunión de planificación habitualmente se divide en dos partes con finalidades diferentes: una primera parte estratégica enfocada en el *qué*, y una segunda parte táctica cuyo hilo conductor es el *cómo*.

a. Primer parte: ¿Qué trabajo será realizado?

El *Product Owner* expone todos y cada uno de los PBIs que podrían formar parte del *Sprint*, mientras que el equipo de desarrollo realiza todas las preguntas que crea necesarias para conocer sus detalles y así corroborar o ajustar sus estimaciones.

Aun asumiendo que los PBIs ya han sido estimados con anterioridad, es posible que en esta reunión aparezcan PBIs que no habían sido estimados anteriormente.

El objetivo buscado durante esta parte de la reunión es identificar *qué* es lo que el equipo de desarrollo va a realizar durante el *Sprint*, todos aquellos PBIs que el equipo se comprometerá a transformar en un producto funcional potencialmente entregable.

El *Product Owner* y los desarrolladores deben participar de esta parte de la reunión como protagonistas principales, mientras que el *Scrum Master* facilita la reunión y asegura que cualquier stakeholder del proyecto que sea requerido para profundizar en detalles esté presente.

El equipo de desarrollo utiliza su capacidad productiva (*Velocity*[10]), obtenida de los *Sprints* pasados, para conocer hasta cuánto trabajo podría comprometerse a realizar. Esto determinaría en un principio cuáles serían los PBIs comprometidos en este *Sprint*.

Cada uno de los ítems del *Product Backlog* debe ser discutido para entender cuáles son sus criterios de aceptación y así conocer en detalle qué se esperará de cada uno. De esta manera, el equipo de desarrollo discute con el *Product Owner* sobre cada PBI y genera un compromiso de entrega para aquellos PBIs que considera suficientemente claros como para comenzar a trabajar y que además podrían formar parte del alcance del Sprint que está comenzado. A esto se lo conoce como planificación basada en compromisos o *Commitment-based Planning*.

Al finalizar esta primera parte de la reunión, tanto el *Product Owner* como los stakeholders involucrados se retirarán, dejando así al *Scrum Master* y a los desarrolladores para que den comienzo a la segunda parte de la reunión.

b. Segunda parte: ¿Cómo será realizado el trabajo?

En esta etapa los desarrolladores determinan la forma en la que llevarán adelante el trabajo. Esto implica la definición inicial de un diseño de alto nivel, el cual será refinado durante el *Sprint* mismo y la identificación de las actividades que el equipo en su conjunto tendrá que llevar a cabo.

Se espera que el diseño sea emergente, es decir, que surja de la necesidad del equipo de desarrollo a medida que éste avance en el conocimiento del negocio. Es por ello que no es necesario realizar un diseño completo y acabado de lo que será realizado durante el *Sprint*. En cambio, se buscará un acuerdo de alto nivel que será bajado a detalle durante la ejecución de la iteración.

Esto mismo sucede con las actividades del *Sprint*, es decir que no es necesario enumerar por completo todas las actividades que serán realizadas durante la iteración ya que muchas aparecerán a medida que avancemos. Se debe tener en cuenta que en este punto los PBIs ya han sido estimados y el surgimiento de actividades durante el *Sprint* no habilita a incrementar las estimaciones de los PBIs, salvo excepciones donde la estimación inicial no había considerado la totalidad del esfuerzo necesario.

Adicionalmente, es recomendable que las actividades duren menos de un día. Esto permitirá detectar bloqueos o retrasos durante las reuniones diarias. Si bien el *Product Owner* no participa de esta reunión, debería ser contactado en el caso de que los desarrolladores necesiten respuestas a nuevas preguntas con la finalidad de clarificar dudas que puedan tener respecto de las necesidades.

Al finalizar esta reunión, el equipo genera un *Sprint Backlog* que representa el alcance del *Sprint* en cuestión. Este *Sprint Backlog* es el que se coloca en la pizarra de actividades del equipo.

Scrum Daily Meeting o Reunión Diaria

Uno de los beneficios de Scrum está dado por el incremento de la comunicación dentro del equipo de proyecto. Esto facilita la coordinación de acciones entre los miembros del equipo de desarrollo y el conocimiento "en vivo" de las dependencias de las actividades que realizan.

[10] *Velocity – Agile Alliance: https://www.agilealliance.org/glossary/velocity/*

Por otro lado, se requiere además aumentar y explicitar los compromisos asumidos entre los miembros del equipo de desarrollo y dar visibilidad a los impedimentos que surjan del trabajo que está siendo realizando y que muchas veces nos impiden lograr los objetivos.

Estos tres objetivos: incrementar la comunicación, explicitar los compromisos y dar visibilidad a los impedimentos, son logrados mediante las reuniones diarias o *Daily Scrum Meetings*. Su duración será como máximo de 15 minutos y de ser posible se realizarán de pie, para que los miembros del equipo no se relajen, ni se involucren en explicaciones que den más detalles de los necesarios.

A la reunión diaria acude el *Scrum Master* y el equipo de trabajo. En el caso de que sea necesario, se podrá requerir la presencia del *Product Owner* y de los stakeholders. Todos y cada uno de los miembros toman turnos para responder las siguientes tres preguntas, y de esa manera comunicarse entre ellos:

1. *¿Qué hice desde la última reunión?* Finalidad: verificar el cumplimiento de los compromisos contraídos por los miembros del equipo en función del cumplimiento del objetivo del Sprint.
2. *¿Qué voy a hacer hasta la próxima reunión?* Finalidad: generar nuevos compromisos hacia el futuro.
3. *¿Qué problemas o impedimentos tengo?* Finalidad: detectar y dar visibilidad a los impedimentos, los cuales no se resuelven en esta reunión, sino en posteriores.

No se trata de una reunión de reporte de avance o status. Por el contrario, es un espacio de estricta comunicación entre los miembros del equipo de desarrollo.

Como apoyo a la reunión, el equipo cuenta con la lista de tareas definidas en la *Sprint Planning*, y donde se actualizará el estado y el esfuerzo pendiente para cada tarea, así como las jornadas y horas que restan para finalizar el *Sprint*.

El encargado de dirigir la reunión es el *Scrum Master*, que se encarga de solucionar los problemas que el equipo no puede solucionar por sí mismo o que les resta tiempo.

Sprint Review o Revisión de Sprint

Reunión que se produce al finalizar el sprint y su objetivo es que el equipo presente al cliente los resultados completados en la iteración y obtenga feedback.

A esta reunión asistirán: el *Product Owner*, el Equipo de desarrollo, el *Scrum Master* y si es posible, alguno de los stakeholders del proyecto, ya que la presencia de estos aporta otra visión sobre el producto distinta a la del *Product Owner*. La duración de esta reunión, suele estar comprendida entre 1 y 4 horas según la duración del sprint.

En esta reunión, el cliente podrá comprobar si han sido desarrollados los objetivos que se plantearon y el equipo valorar si entendió de forma correcta cuales eran los requisitos solicitados por el cliente, siendo muy productiva, ya que al mostrar resultados en un corto periodo de tiempo (la duración del sprint), pueden realizarse las adaptaciones necesarias en el proyecto de manera objetiva y temprana, desde la primera iteración.

Toda la retroalimentación que los stakeholders aporten debe ser ingresada como PBIs en el *Product Backlog*. Para esto, los PBIs nuevos deben ser estimados y priorizados con respecto a todos los ya existentes en el *Product Backlog*.

En el caso de que una funcionalidad sea rechazada, el PBI correspondiente reingresa al *Product Backlog* con máxima prioridad, para ser tratado en el siguiente *Sprint*. La única excepción a esta regla es que el *Product Owner*, por decisión propia, prefiera dar mayor prioridad a otros.

Sprint Retrospective o Reunión de Retrospectiva

Esta reunión es la última del *Sprint*, y su objetivo es analizar cómo se ha trabajado durante el mismo, el por qué se han logrado o no los objetivos comprometidos en su inicio y si se han cumplido las expectativas del cliente. Su timeline es de unas dos a tres horas, según duración del *Sprint*.

Sirve para conocer:

- Si el equipo ha funcionado bien.
- Si hay aspectos en el sistema de trabajo que hay que mejorar.
- Qué hemos aprendido.
- Cuáles son los problemas que podrían impedir progresar adecuadamente.

Mediante el mecanismo de retrospección, el equipo reflexiona sobre la forma en la que realizó su trabajo y los acontecimientos que sucedieron en el *Sprint* que acaba de concluir para mejorar sus prácticas.

Esta reunión tiene lugar inmediatamente después de la reunión de revisión. Mientras que la reunión de revisión se destina a revisar el *"qué"*, la retrospectiva centra el proceso en el *"cómo"*.

Este tipo de actividad necesita un ambiente seguro donde el *Scrum Team* pueda expresarse libremente, sin censuras ni temores, por lo que se suele restringir solo a los *Developers* y al *Scrum Master*.

Valiéndose de técnicas de facilitación y análisis de causas raíces, se buscan tanto fortalezas como oportunidades de mejora. Luego, el *Scrum Team* decide por consenso cuáles serán las acciones de mejora a llevar a cabo en el siguiente *Sprint*. Estas acciones y sus impactos se revisarán en la próxima reunión de retrospectiva.

Backlog Grooming o Backlog Refinement

El refinamiento del *Product Backlog*[11] es una actividad constante a lo largo de todo el *Sprint*, la cual no tiene una posición fija en este, y a la cual se recomienda que los equipos dediquen al menos un 5% del tiempo de cada iteración. Su objetivo es profundizar en el entendimiento de los PBIs que se encuentran más allá del Sprint actual y así dividirlos en PBIs más pequeños, si lo requieren, y estimarlos. Idealmente se revisan y detallan aquellos que potencialmente se encuentren involucrados en los próximos dos o tres *Sprints*.

Otro objetivo importante que se debe perseguir en esta reunión es la detección de riesgos implícitos en los PBIs que se estén analizando, y en función de ellos revisar y ajustar las prioridades del *Product Backlog*.

En ella colabora todo el equipo (*Scrum Master*, Equipo de desarrollo y *Product Owner*), y se trabaja de cara al *Sprint Planning* de la siguiente iteración, estimando componentes del *Product Backlog*, para tener claros los requisitos de trabajo en el próximo sprint.

En el caso de que se realice como una reunión, la responsabilidad de convocarla es del *Product Owner*, entre una y dos veces por *Sprint*, facilitadas por el *Scrum Master*.

Durante el *Sprint* en curso, el *Product Owner* trabajará sobre el *Product Backlog* del proyecto, añadiendo requisitos nuevos, modificando los que ya estaban presentes e incluso eliminado otros. Estas modificaciones son necesarias porque el cliente a lo largo del sprint puede tener nuevas inquietudes y porque el contexto del proyecto puede cambiar.

Gracias a esta reunión el cliente podrá obtener una reestimación de los costos para completar los objetivos, en caso de que estos se desvían de los iniciales; podrá replanificar el proyecto para obtener un nuevo calendario de entregas o incorporar nuevos recursos, y el equipo puede ajustar la velocidad de desarrollo a la complejidad de los nuevos objetivos, teniendo ya de base la experiencia adquirida durante el *Sprint*.

[11] *Backlog Refinement – Agile Alliance:* https://www.agilealliance.org/glossary/backlog-grooming/

Bibliografía y material de consulta adicional

[1] Acercamiento a las Metodologías Ágiles: https://medium.com/acercamiento-a-las-metodolog%C3%ADas-%C3%A1giles

[2] Agile Alliance: https://www.agilealliance.org/

[3] Agile Manifesto: https://agilemanifesto.org/

[4] Alaimo, Diego Martín. *Proyectos ágiles con Scrum: flexibilidad, aprendizaje, innovación y colaboración en contextos complejos*. Kleer. 2013.

[5] Asignatura "*Metodologías ágiles en el desarrollo de softwarte*" – UTN FRRO: https://sites.google.com/view/agilesutnfrro

[6] Cohn, Mike. *User Stories Applied for Agile Software Development*. Pearson Education, Inc. 2009.

[7] Gestión ágil vs gestión tradicional de proyectos ¿cómo elegir?: https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir

[8] Project Management Institute: https://www.pmi.org/

[9] Ries, Eric. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Busineses.* Random House. 2011.

[10] Scaccia, Candela. Santa Clara, Mariana. *Proyecto 050-19 – Estudio de las Metodologías Ágiles y su vínculo con la Ingeniería Industrial.* 2020.

[11] Schwaber, Ken. Sutherland, Jeff. *La Guía Definitiva de Scrum: Las Reglas de Juego*. 2020. https://scrumguides.org/

[12] Scrum.org: https://www.scrum.org/

[13] Stellman, Andrew. Greene, Jennifer. *Learning Agile: Understanding Scrum, XP, Lean, and Kanban.* O'Reilley. 2015.

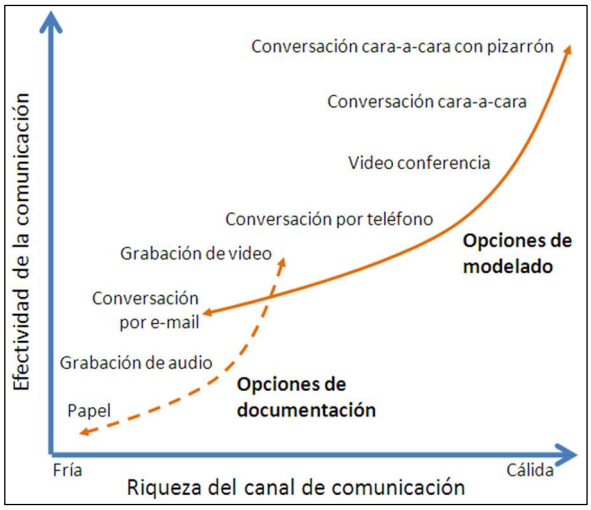
[14] Success in Disruptive Times – Project Management Institute: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf

[15] The Agile Coach – Atlassian: https://www.atlassian.com/agile

Anexo I – User Stories y conceptos relacionados

Las Historias de Usuario surgieron en eXtremme Programming (XP) en 1998 como una respuesta a una situación habitual en los proyectos de desarrollo de software: los clientes o especialistas de negocio se comunican con los equipos de desarrollo a través de extensos documentos conocidos como especificaciones funcionales. A su vez, las especificaciones funcionales son la documentación de supuestos y están sujetas a interpretaciones, lo que da lugar a malos entendidos y a que el software construido no se corresponda con la realidad esperada.

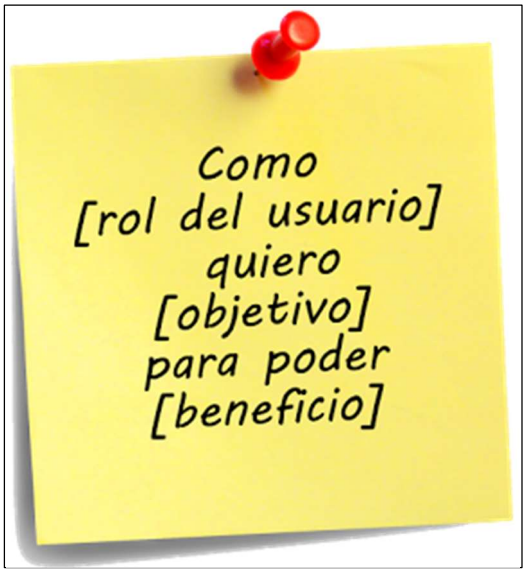
El uso de estas especificaciones detalladas como medio de comunicación no suele conducir a resultados satisfactorios ya que el contenido suele abarcar una porción muy acotada del espectro de la comunicación humana[12], siendo este uno de los motivos del 6to principio de las metodologías ágiles.



Para tener una comunicación sólida y completa es necesario el contacto cara a cara entre los interlocutores. En un esfuerzo orientado a 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 y no un remplazo.

Componentes de una Historia de Usuario

En 2001 Ron Jeffries propuso el modelo de las 3 C's[13], según el cual una *User Story* se compone de 3 elementos:



- ***Card:*** Toda historia de usuario debe poder describirse en una ficha de papel pequeña, como puede ser un *Post-It*, que dé una forma tangible y duradera a lo que de otro modo sería solo una abstracción. Además, si una Historia de Usuario no puede describirse en ese tamaño, es una señal de que estamos traspasando las fronteras y comunicando demasiada información que debería compartirse de otro modo, como cara a cara.
- ***Conversation:*** Toda historia de usuario debe surgir de una conversación con el *Product Owner*. Una comunicación cara a cara en la que se intercambia no solo información sino también pensamientos, opiniones y sentimientos, la cual tiene lugar en diferentes momentos y lugares durante el proyecto con los diferentes stakeholders.
- ***Confirmation:*** La confirmación de que se han alcanzado los objetivos entorno a los cuales giraba la conversación; esto implica que la *user story* deba estar lo suficientemente explicada para que el equipo de desarrollo sepa qué es lo que debe construir y qué es lo que el *Product Owner* espera. Esto se conoce también como Criterios de Aceptación.

---

[12] De acuerdo con Albert Mehrabian la comunicación humana se compone en un 7% del contenido y un 93% de comunicación no verbal, 38% tono de voz y 55% lenguaje corporal: https://en.wikipedia.org/wiki/Albert_Mehrabian
[13] The Three C's – Agile Alliance: https://www.agilealliance.org/glossary/three-cs/

Redacción de una Historia de Usuario

La siguiente forma de redactar Historias de Usuarios, conocida como formato *Connextra* o *Role-Feature-Reason*, fue inventada en 2001. Esta forma de redactar es también sugerida por Mike Cohn en su libro *User Stories Applied for Agile Software Development*[14].

"Como (rol) necesito (funcionalidad) para (beneficio)"

Ejemplo:

- **Como** *un cliente del banco*
- **Necesito** *retirar dinero del cajero automático*
- **Para** *no estar limitado por los turnos y horarios bancarios de la caja.*

Los beneficios de este tipo de redacción son, principalmente:

- La redacción en primera persona de la Historia de Usuario invita a quien la lee a ponerse en el lugar del usuario.
- Tener esta estructura para redactar la Historia de Usuario ayuda al *Product Owner* a priorizar, ya que le permite comprender fácilmente cuál es la funcionalidad, quien se beneficia y cuál es el valor de la misma.
- Conocer el propósito de una funcionalidad permite al equipo de desarrollo plantear alternativas que cumplan con el mismo propósito en el caso de que el costo de la funcionalidad solicitada sea alto o su construcción no sea viable.
- Logra que las personas intervinientes en la conversación presten atención no solo al *qué* debe hacer el producto a desarrollar, sino también a pensar en *para quién* lo hace y en la búsqueda de *qué objetivos.*

Características de una Historia de Usuario

El acrónimo INVEST surge en 2003 como un *checklist* o lista de criterios para evaluar la calidad de una *User Stories*[15], y constituye una serie de 6 características que se espera que toda Historia de Usuario posea.

- ***Independent.*** Deben ser independientes de forma tal que no se superpongan funcionalidades y que puedan planificarse y desarrollarse en cualquier orden. Esta característica puede no cumplirse para todas las historias de usuario, de igual modo, el objetivo que se debe perseguir es cuestionar en cada caso si se hizo todo lo posible para que ésta sea independiente del resto.
- ***Negotiable.*** Una buena Historia de Usuario es Negociable. No es un contrato explícito, sino que el alcance y los criterios de aceptación de las Historias podría ser variable: pueden incrementarse o reducirse a medida que el desarrollo avanza, en función del feedback de los stakeholders y de la performance del Equipo.
- ***Valuable.*** Una Historia de Usuario debe ser valorable o cuantificable por el *Product Owner*. En general, esta característica representa un desafío a la hora de dividir Historias de Usuario.
- ***Estimable.*** Una Historia de Usuario debe ser estimable. Mike Cohn, identifica tres razones principales por las cuales una Historia de Usuario no podría estimarse:
    - La Historia de Usuario es demasiado grande. En este caso la solución sería dividir la Historia de Usuario en historias más pequeñas que sean estimables.
    - Falta de conocimiento funcional. En este caso la Historia de Usuario vuelve al Product Owner para bajar en detalle la historia o conversar con el Equipo de Desarrollo.
    - Falta de conocimiento técnico. En estos casos el Equipo de Desarrollo puede dividir la historia en un *timebox* conocido como *spike* que le permita investigar la solución y proveer una estimación más certera.

---

[14] *User Stories Applied for Agile Software Development:* http://athena.ecs.csus.edu/~buckley/CSc191/User-Stories-Applied-Mike-Cohn.pdf

[15] *INVEST in Good stories, and SMART Tasks – Exploring Extreme Programming:* https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

- ***Small.*** Toda *User Story* debe ser lo suficientemente pequeña de forma tal que permita ser estimada por el Equipo de Desarrollo, con una duración generalmente de no más de 2 semanas de trabajo de un solo desarrollador. Las descripciones de las Historias de Usuario también deberían ser pequeñas, escribirlas en fichas pequeñas o *Post-It* ayuda a que esto suceda.
- ***Testable.*** Una buena Historia de Usuario es Verificable. Se espera que el *Product Owner* no solo pueda describir la funcionalidad que necesita, sino que también logre probarla. Algunos Equipos acostumbran solicitar los criterios de aceptación antes de desarrollar la Historia de Usuario.

Redacción de pruebas de aceptación de User Stories

El modelo *Given-When-Then* se utiliza para guiar la redacción de *acceptance tests* o pruebas de aceptación de las User Stories[16].

"Dado (cierto contexto) cuando (alguna acción es llevada adelante) entonces (debería obtenerse un conjunto de consecuencias observables)"

Ejemplo:

- **Dado** *que cuento con saldo en mi cuenta bancaria y no realice retiros recientemente*
- **Cuando** *intento retirar una cantidad de dinero menor al límite de extracción diario del cajero*
- **Entonces** *el retiro debe efectuarse sin errores ni advertencias.*

Definition of Ready[17]

La definición de listo es el conjunto de características que una *User Story* debe cumplir para que el Equipo de Desarrollo pueda comprometerse a su entrega, es decir, incluirla en un *Sprint Backlog*.

Una definición típica de listo se da cuando la Historia de Usuario cumple con los criterios *INVEST* y todos sus pre requisitos están resueltos.

La definición de listo evita que se comience a trabajar en características que no tienen criterios de finalización claramente definidos, lo cual se suele traducir en costosas discusiones o reelaboraciones, y, a su vez, proporciona al equipo un acuerdo explícito que le permite "rechazar" la aceptación de características mal definidas.

Definition of Done[18]

La definición de terminado es el conjunto de características que un Incremento o una *User Story* debe cumplir para que el equipo de desarrollo considere que ha culminado de trabajar en ella.

Un típico criterio de terminado se da cuando todos las *acceptance test* se superan correctamente, cuando todos los archivos fuentes están en el repositorio de código fuente y el build se ejecutó exitosamente. Podría considerarse el visto bueno del *Product Owner* sobre la funcionalidad construida antes de llegar a la *Review Meeting*.

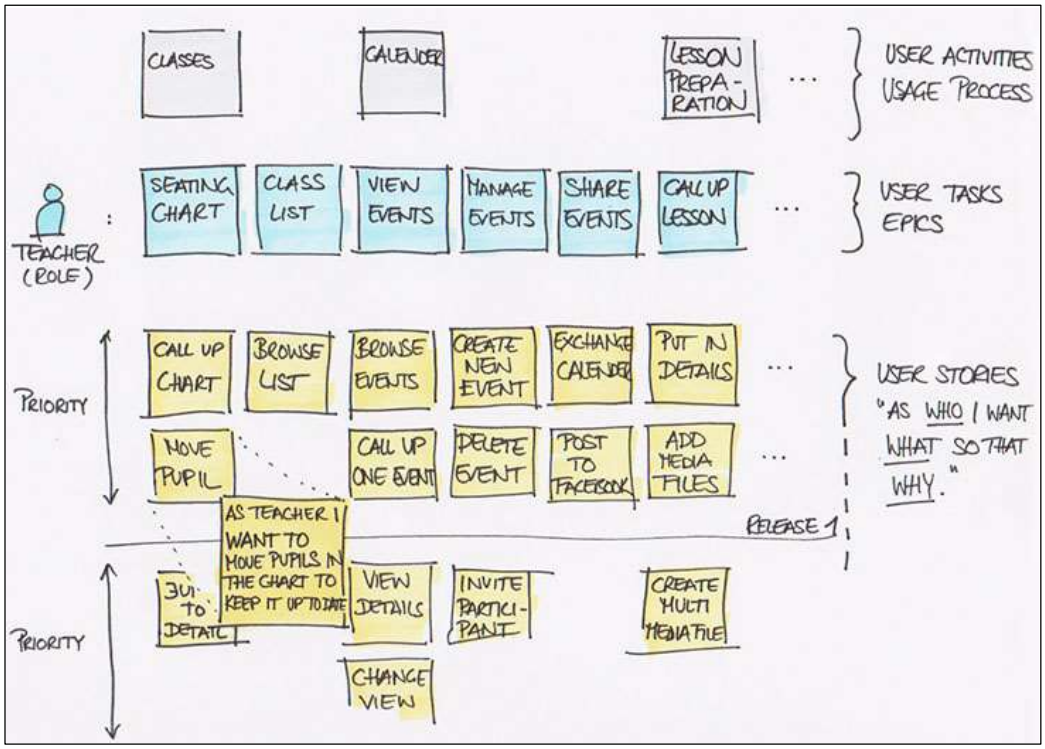
La definición de terminado proporciona una lista de verificación o *checklist* que guía de manera útil las actividades previas a la implementación: discusión, estimación, diseño. A su vez, la definición de terminado limita el costo de reelaboración una vez que una característica ha sido aceptada como terminada, y el hecho de tener un contrato explícito limita el riesgo de malentendidos y conflictos entre el equipo de desarrollo y el cliente o propietario del producto.

User Story Mapping

El *User Story Mapping*, desarrollado por Jeff Patton, es una técnica que permite establecer en una cuadrícula todas las posibles tareas que el usuario de un producto o servicio haría; es decir, el recorrido

[16] *Given-When-Then – Agile Alliance: https://www.agilealliance.org/glossary/gwt/*
[17] *Definition of Ready – Agile Alliance: https://www.agilealliance.org/glossary/definition-of-ready/*
[18] *Definition of Done – Agile Alliance: https://www.agilealliance.org/glossary/definition-of-done/*

de los usuarios durante las etapas del producto o servicio. Permite la visualización de todas las tareas que el usuario completa cuando usa un producto, y es una forma de organizar las *User Stories*.



Epic

Los *Epics* son grandes cantidades de trabajo que se pueden desglosar en una serie de *User Stories.* No existe una forma estándar de redactarlas, aunque algunos equipos utilizan los formatos usados para las historias de usuario o utilizan frases cortas.

Los *Epics* permiten realizar un seguimiento de las ideas grandes y poco definidas del *Backlog*, sin la necesidad de sobrepoblarlo con varios elementos. Pueden representar ideas vagas en el *Backlog* hasta que el equipo identifique la necesidad de entregar el resultado que ese ítem permite. En ese momento, el equipo puede dividir la *Epic* en historias de usuarios más pequeñas durante la *Product Backlog Refinemet.*

Las épicas permiten establecer una jerarquía para los elementos del Backlog, en el cual una épica representa una idea original, a menudo estrechamente relacionada con un resultado en particular. Las historias de usuario asociadas con esa *Epic* representan los diversos aspectos de la solución que necesita entregar, o las opciones que tiene para satisfacer esa necesidad.

Theme

Un *Theme* se utiliza para agrupar *User Stories* que tratan el mismo tema. El concepto de *Theme* se usa con frecuencia para agrupar historias de usuario que se identificaron por separado y se puede usar como un tipo de filtro de decisión para decidir qué historias agrupar en un *Sprint* o entrega en particular. Por lo general, los temas no se utilizan como un nivel en una jerarquía de trabajos pendientes.

Task

Las tareas o *Tasks* son subdivisiones de las *User Stories*, y suelen ser de alto contenido técnico. En ellas se detallan las tareas que los desarrolladores deben realizar para alcanzar la definición de terminado de la historia de usuario en cuestión.