INGENIERÍA y CALIDAD DE SOFTWARE

EL PROCESO DE SOFTWARE

La elaboración de software es un proceso reiterativo de aprendizaje social, y el resultado es la reunión de conocimiento recabado, depurado y organizado a medida que se realiza el proceso.
Desde un punto de vista técnico, el **proceso de software** se define como un conjunto estructurado de actividades, acciones y tareas que se requieren a fin de desarrollar y construir software de alta calidad.
¿"Proceso" es sinónimo de "ingeniería de software"? La respuesta seria "si y no". Un proceso de software define el enfoque que se adopta mientras que se hace ingeniería sobre el software. Pero la ingeniería de software también incluye las tecnologías que pueblan el proceso: **métodos técnicos** y **herramientas automatizadas**.
Además, la ingeniería de software es llevada a cabo por personas creativas y preparadas que deben adaptar un proceso maduro de software a fin de que resulte apropiado para los productos que construyen y para las demandas de su mercado.

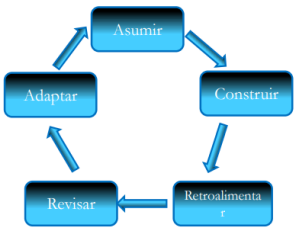
Cuando hablamos de **proceso** nos referimos a una colección de actividades de trabajo, acciones y tareas que se realizan cuando va a crearse algún producto terminado, donde cada actividad se encuentra dentro de una estructura o modelo que define su relación tanto con el proceso, como con el resto de las actividades.
Siguiendo con esto, un **proceso de software** será el conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.
Debemos tener en cuenta ciertas consideraciones.

- Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse, pero siempre deben incluir **productos**, **roles**, **responsabilidades** y **condiciones**.
- El proceso debe ser explícitamente modelado si va a ser administrado.
- Un proceso de desarrollo se puede aplicar con varios ciclos de vida, y esto dependerá de la estrategia que se tome. El ciclo elegido debe permitir seleccionar los artefactos a producir, definir roles y actividades, y modelar conceptos.
- Las personas utilizan herramientas y equipos para llevar a cabo los procedimientos que componen el proceso de desarrollo.

Procesos Definidos vs Proceso Empíricos

Hay proyectos que son bastante sencillos y relativamente predecibles, donde se pueden aplicar herramientas racionales de planificación y toma de decisiones. Otros proyectos que son más complejos o impredecibles requieren un enfoque diferente que se base más en la autoorganización y la innovación.
La mayoría de los proyectos de desarrollo de software se consideran de naturaleza compleja e impredecible debido a la convergencia de tres factores: personas, requerimientos y tecnología.

Tradicionalmente, una vez comienza un proyecto, se crea un paquete de requerimientos y luego se aprueban. El gerente del proyecto asume que esta aprobación da como resultado un conjunto fijo de requerimientos y que ahora puede comenzar la planificación. El director del proyecto estima cuanto tiempo llevara completar los requerimientos y crea el plan del proyecto. Este plan predice que el mismo estará terminado en una fecha determinada, y esa fecha se comunica al cliente.
Este tipo de procesos se definen como **proceso definidos**. En estos, se asume que podemos repetir el mismo proceso una y otra vez, indefinidamente, y obtener los mismo resultados en todas las repeticiones. Sin embargo, el defecto fundamental de este enfoque es que el plan se basa en la suposición de que los requerimientos son fijos y no cambiaran; y como sabemos, este nunca es el caso, ya que los requerimientos nunca son fijos, y siempre cambian. Cuando estos cambian, el plan se ve afectado, y como resultado, la fecha de finalización también debería cambiar. Desafortunadamente, no siempre es posible cambiarla y el equipo debe entregar en la fecha en que se comprometieron. Esto lleva a una gran crisis y el proyecto empieza a salirse de control.



Es por esta razón que surge otro tipo de proceso, conocido como **proceso empírico**, el cual promueve un enfoque ágil impulsado por el valor, basado justamente por el empirismo que cambia toda la mentalidad. De esta forma, se asumen proceso complicados con variables cambiantes, es decir, se asume desde el principio que los requerimientos que existen por adelantado no son fijos y que cambiaran, y cuando se repite el proceso se podrá llegar a obtener resultados diferentes.

Se dice que son procesos agiles porque acentúan la maniobrabilidad y la adaptabilidad, y esta mentalidad ágil supone que debe entregar en una fecha determinada, por lo que este enfoque fija el tiempo y los recursos, y deja los requerimientos sin determinar. Cuanto se tiene una cantidad fija de tiempo en la que no se está seguro de poder cumplir todos los requerimientos, la reacción natural será priorizar los mismos, y terminar primero aquellos que agregan más valor al cliente. Es por esto que son proceso impulsados por valor, donde el principal factor de aprendizaje es la experiencia.

CICLO DE VIDA

Todo esfuerzo de desarrollo de software atraviesa un ciclo de vida, que consiste de todas las activades desde el momento en que inicia una versión 1.0 de un sistema, hasta la versión en la cual toma su ultimo respiro en la máquina del consumidor final.

Un ciclo de vida de un proyecto software es un **representación de un proceso**, y el mismo grafica una **descripción del proceso** desde una perspectiva particular. De esta forma, existen distintos modelos de ciclo de vida, donde un modelo de ciclo de vida es un modelo prescriptivo de lo que debe ocurrir entre el primer destello del sistema hasta su ultimo respiro.

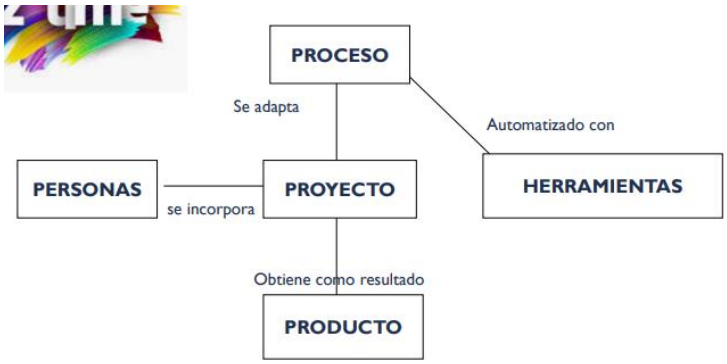
El propósito principal del modelo de ciclo de vida es establecer un orden en el cual el proyecto especifica, prototipa, diseña, implementa, revisa, testea y realiza otras actividades. Además, establece el criterio para determinar cuándo proceder de una tarea a la siguiente.

El modelo de ciclo de vida más conocido es el ciclo de vida **en cascada**, el cual, sin embargo, tiene ciertas debilidades. Existen otros tipos de ciclos de vida, y en muchos casos son mejores opciones que el modelo de cascada, como lo son el **iterativo/incremental** o el **recursivo**.

CICLO DE VIDA	DESCRIPCIÓN	VENTAJAS	DESVENTAJAS
Cascada o Secuencial	toma las actividades fundamentales del proceso, como especificación, desarrollo, validación, evolución, y las representa como fases separadas del mismo: especificación de requerimientos, diseño de software, implementación, pruebas, etc.	✓Desarrollo dirigido por un plan ✓Cada etapa genera documentación para realizar un monitoreo constante contra el plan. ✓Muy útil cuando los requerimientos son claros y es poco probable un cambio drástico durante el desarrollo	○La documentación puede ser burocrática y excesiva ○En etapas finales puede ser que haya que hacer re trabajo por cambios en requerimientos o fallas de diseño
Iterativo	Vincula las actividades de especificación, desarrollo y validación, y el sistema se desarrolla como una serie de versiones (incrementos) donde cada una añade funcionalidades a la versión anterior.	✓La especificación, desarrollo y validación están entrelazadas en lugar de separadas y aisladas. ✓Rápida retroalimentación a través de las actividades. ✓Muy útiles para sistemas de requerimientos cambiantes ✓Más fácil y menos costoso implementar cambios. ✓Cada iteración genera funcionalidad para el cliente	○Los incrementos progresivos tienden a degradar la estructura del sistema.
Recursivo	Se inicia con algo en forma completa, como una subrutina que se llama a sí misma e inicia nuevamente. Se presenta un prototipo que va mejorando con cada vuelta.	✓Se generan productos independientes de la implementación, que pueden ser reusables en sistemas de características similares.	○Puede ser más costoso en tiempo y dinero readaptarlos para reutilizarlos para los diferentes proyectos. ○La tecnología puede ser obsoleta. ○Pueden carecer de mantenimiento o documentación.

Al definir el plan maestro para el proyecto, el modelo de ciclo de vida que se elige tiene una influencia mucho mayor en el éxito del proyecto que cualquier otra decisión que se tome. Un modelo de ciclo de vida apropiado puede optimizar el proyecto y ayudar a asegurar que cada paso nos mueve más cerca del objetivo.
Dependiendo del ciclo de vida que se elija se puede mejorar la velocidad de desarrollo, la calidad, el control y seguimiento del proyecto, y se puede minimizar los gastos generales, la exposición al riesgo, y además, se puede mejorar la relación con los clientes.
Un modelo de ciclo de vida incorrecto puede ser una fuente constante de retraso en el trabajo, trabajo repetitivo, trabajo innecesario y frustración. Lo mismo ocurrirá cuando no se elija un modelo de ciclo de vida.

Las cuatro P del Desarrollo de Software



- **Proceso:** cuando hablamos de proceso nos referimos al **proceso de desarrollo de software**, y lo definimos como el **conjunto completo de actividades necesarias para transformar los requerimientos de un usuario en un sistema software (producto)**. De esta forma, es el conjunto estándar de actividades (tareas agrupadas con características comunes) con un objetivo.
- **Proyecto:** es un **elemento organizativo** a través del cual **se gestiona el desarrollo del software**. El resultado de un proyecto es una version del producto. Es un **esfuerzo temporal** que se lleva a cabo para crear un producto, servicio o resultado único.
  El proyecto establece **fechas definidas de inicio y finalización**, una **especificación clara del objetivo** y **alcance del sistema**, y un **presupuesto** establecido. A través del ciclo de vida, un equipo de proyecto debe preocuparse del **cambio**, las **iteración** y el **patrón organizativo** dentro del cual se conduce el proyecto.
  Un proyecto es una **instancia única de ejecución** que organiza; es **particular**, es decir, tiene sus respectivas personas, clientes, técnicas y productos; es **temporal**, es decir, tiene una fecha de inicio y fin; y **obtiene resultados únicos**, un proyecto es único y diferente cada vez que lo instanciamos, y genera una versión única del software, por lo que no se puede masificar su producción.
- **Producto:** son **"artefactos" que se crean durante la vida del proyecto**, como los modelos, código fuente, ejecutables y documentación. Es el **resultado de la ejecución de las actividades de desarrollo del sistema**.
  Por ello el producto **no es únicamente el código**, sino que hace referencia al sistema entero y no solo al código que se entrega, incluye documentación de requerimientos, análisis y diseño del sistema, resultados de las pruebas y medidas de productividad.
- **Personas:** involucra a los **principales autores del proyecto**, incluyendo analistas, diseñadores, arquitectos, desarrolladores, tester, y el personal de gestión que les da soporte, además de también los usuarios, clientes y otros interesados (stakeholders).
  Existen personas que están implicadas en el desarrollo de un producto de software durante todo su ciclo de vida, y otras serán aquellas que financian el producto, lo planifican, lo desarrollan, lo gestionan, lo prueban, lo utilizan, o se benefician de el de alguna forma.
- **Herramientas:** son todos aquellos artefactos que proporcionan un apoyo automatizado en el proceso.

La meta de todo **proyecto** de software es producir un **producto** de software, y por ello el sistema deseado está constituido por un conjunto de productos finales, que son más que el código fuente y los ejecutables, ya que incluyen distinto tipo de documentación.
Además, es clave el **proceso** mediante el cual los proyectos producen productos de manera efectiva. En cada proceso se define un conjunto de tareas específicas, y para cada tarea se identifica un procedimiento que define la forma de ejecutarla, y serán el vehículo de comunicación entre usuarios y desarrolladores, es decir, entre las **personas** que estarán involucradas en el proyecto.

PROYECTO DE SOFTWARE

Ya vimos que un **proyecto** es un elemento que nos permite organizar como será la gestión del desarrollo del software, y cuyo resultado o salida será una versión del producto.
El proyecto incluye todo el esfuerzo temporal que se lleva a cabo para crear un producto, servicio, o cualquier otro tipo de resultado que se quiera obtener. El mismo cuenta con ciertas características:

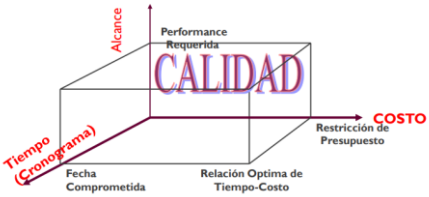
- **Orientación a Objetivos:** los proyectos están dirigidos a obtener resultados y ello se refleja a través de objetivos. Estos objetivos guiaran al proyecto, y para esto deben ser **no ambiguos**, es decir, deben ser claros, dado que son los responsables de guiar el desarrollo del proyecto; **medibles**, con el fin de poder determinar el avance del proyecto; y por sobre todas las cosas, **alcanzable**, no alcanza con que sea claro, sino que debe poder ser realizado por el equipo.
- **Duración Limitada:** los proyectos son temporarios, es decir, tienen un principio y un fin, donde este último estará marcado cuando se alcanzan los objetivos del proyecto. Por ejemplo, una línea de producción no es un proyecto, ya que nunca finaliza.
- **Tareas Interrelacionadas Basadas en Esfuerzos y Recursos:** esto se debe a la complejidad sistémica de los distintos problemas que pueden surgir en el proyecto, donde surgirán distintas actividades que deberán ser realizadas con el esfuerzo y los recursos disponibles.
- **Únicos:** a pesar de que puedan existir proyectos que sean similares o que cuenten con cierta semejanza, cada uno de ellos tendrá características que los harán únicos, ya sea por el equipo que lo desarrolla, por el cronograma, por el framework utilizado, etc.

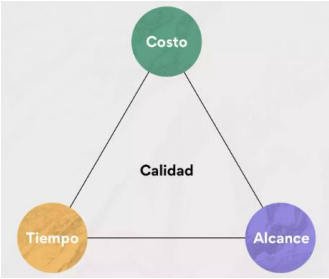
ADMINISTRACIÓN DE PROYECTOS

Cuando hablamos de **administrar un proyecto**, nos referimos a la posibilidad de tener el trabajo hecho y finalizado, teniendo en cuenta factores como lo son **el tiempo**, el **presupuesto acordado**, y habiendo satisfecho los **requerimientos**.
Como una definición formal, podemos decir que la **Administración de Proyectos** es la aplicación de conocimientos, habilidades, herramientas y técnicas en las actividades del proyecto, para satisfacer los requerimientos del mismo.
De esta forma, la administración de proyectos incluirá actividades como la **identificación de los requerimientos**, el **establecimiento de objetivos** de forma que sean claros, medibles y alcanzables, y la **adaptación de las especificaciones, planes y el enfoque** a los diferentes intereses de los stakeholders.

Triple Restricción

Para quienes se encarga de realizar esta administración de los proyectos, es muy complicado mantener la productividad de su equipo con muy poco tiempo, un presupuesto insuficiente, y un alcance increíblemente grande.
Con estas tres variables, **Costo**, **Tiempo** y **Alcance**, se forma lo que se conoce como triple restricción. En la misma, ninguna de las variables se puede cambiar sin compensarlo en los otros dos puntos, por lo que el trabajo de quien lidera los proyectos será equilibrar los tres elementos para mantener su proyecto dentro del presupuesto y el plazo, y al mismo tiempo, cumplir con las especificaciones del alcance del proyecto.





- **Alcance**. Son los requerimientos del proyecto, es decir los límites o el ámbito sobre el cual se va a mover el mismo. Esta variable es la que primero se negocia con el cliente.
- **Tiempo**. Hace referencia al calendario, cuáles serán las fechas especificadas para realizar las entregas que determinarán el avance del proyecto.
- **Costo**. Son los recursos que se verán implicados en el desarrollo del proyecto. Esto incluye equipamiento, infraestructura, equipos, salarios, entre otros.

Como dijimos, si se cambia una de las variables de la triple restricción, las otras deben ser modificadas para poder mantener el proyecto siguiendo con las condiciones establecidas. Si se llega a romper esta triple restricción, es decir, si uno de los puntos se mueve sin modificar uno o ambos de los restantes junto con él, la **calidad** del proyecto se verá **afectada**.
El trabajo del líder del proyecto será equilibrar los tres puntos para lograr la mayor calidad posible a la vez que se mantiene dentro del presupuesto, los plazos, y cumple con las especificaciones del proyecto.

En cuanto a cómo es la relación de cambio entre las distintas variables, podemos decir que el **alcance** es **directamente proporcional** tanto al **tiempo** como al **costo**, ya que si el alcance aumenta en un proyecto, necesariamente deberán aumentar el tiempo y los recursos necesarios para poder abordarlo. Sin embargo, en muchos casos no es posible aumentar el presupuesto o extender los plazos, por lo que, en esta situación, lo que se deberá hacer es **priorizar**, de forma que se defina cuáles son las actividades más importantes y más urgentes que se deben llevar a cabo, y destinar los recursos disponibles en dichas actividades, para poder completarlas en el plazo acordado.

Por otro lado, la relación entre el **costo** y el **tiempo** es **inversamente proporcional**, es decir, que si uno debe ser reducido, el otro deberá extenderse.
Como vemos, es imposible modificar una de las variables sin tener la obligación de cambiar al menos una de las restantes, ya que si queremos ofrecer un producto que cuente con la mayor calidad posible, debemos hallar la forma de equilibrar estos tres elementos.
Esta **triple restricción** estará **equilibrada** cuando el **alcance** sea **igual** a la **combinación** de los **costos** y el **tiempo**.

Existe una gran diferencia cuando trabajamos con procesos definidos o tradicionales y con procesos empíricos o agiles, en términos de esta triple restricción.
En **procesos tradicionales**, esta triple restricción siempre va a estar basada en el **alcance**. Es decir, a la hora de iniciar un nuevo proyecto, tendremos un alcance definido, con los distintos requerimientos que debemos llevar a cabo y realizar para poder concretar el producto, y en base a dicho alcance, es que plantearemos las otras dos variables que son el tiempo y el costo, es decir, se verá que es lo que hay que hacer, y luego veremos cuanto tiempo nos podrá llevar realizar dichas actividades, y con que costo (recursos, personal, etc.).
Por otro lado, cuando trabajamos con **procesos agiles**, lo que tendremos definido o fijo será el **tiempo y el costo**, y en base a esos elementos es que iremos planteando los distintos requerimientos o actividades que se llevaran a cabo (el alcance). Es decir, dejamos de poner el foco en que es lo que se deberá hacer, y nos basamos en el presupuesto y el calendario con el que contamos.
De esta forma, decimos que un *enfoque tradicional* estará **dirigido por un plan**, ya que nos basamos en cual es el alcance que queremos obtener; mientras que en un *enfoque ágil* estará **dirigido por valor** que el producto entregue al cliente.

Líder de Proyecto o Líder de Equipo
En un enfoque tradicional, el líder de proyecto realiza la toma de decisiones en su totalidad sin la inclusión de la opinión del equipo. El líder asigna las tareas que deben realizar los integrantes del equipo, y además, maneja todas las relaciones con todos los stakeholders del proyecto (clientes, gerencias, contratistas, stakeholders en general).
Usualmente los profesionales técnicos no disponen de habilidades necesarias para tratar con la gente, por lo que es necesario una figura de líder de proyecto con las siguientes aptitudes:
1. **Motivación**: habilidad para alentar al personal técnico a producir a su máxima capacidad.
2. **Organización**: habilidad para adaptar el proceso existente (o inventar nuevos), para lograr que el concepto inicial se transforme en el producto final.
3. **Innovación**: habilidad para alentar a las personas a crear y sentirse creativas.
4. **Empatía**.
5. **Comunicación**.
6. **Liderazgo**.

También cuenta con habilidades duras, relacionadas a los conocimientos del producto, técnicas y herramientas. Dado que el líder de proyecto tiene un enfoque basado en la resolución de problemas, también son características propias las siguientes:
1. **Resolución de problemas**: identifica conflictos técnicos y organizativos, estructura la solución o motiva a otros profesionales para que la desarrollen, aplica lecciones aprendidas en otros proyectos y adapta el proceso de resolución ante inconvenientes.
2. **Identidad administrativa**: tiene la confianza de asumir el control cuando es necesario.
3. **Logro**: recompensa las iniciativas y los logros, incentivando al equipo a correr los riesgos de manera controlada.
4. **Influencia**: define la estructura del equipo en función de la retroalimentación que él mismo suministra mediante palabras y gestos.

Entre las responsabilidades del líder de proyecto se encuentran, en las nuevas metodologías:
1. Definir el **alcance** del proyecto.
2. Identificar a los **involucrados**, **recursos** y **presupuesto**.
3. Detallar las **tareas**, estimar los tiempos y requerimientos.
4. Identificar y evaluar **riesgos**.
5. Preparar **planes de contingencia**.
6. Controlar hitos y participar en las **revisiones** de las fases del proyecto.

7. Administrar el **proceso de control de cambios**.
8. Producir **reportes** de estado.

Equipo de Proyecto
Es un grupo de personas comprometidas con alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.
Entre sus características se encuentran:
1. Cuentan con conocimientos, habilidades técnicas, motivación, experiencias y diversas personalidades. Además, deben tener un sentimiento de mejora continua y aprendizaje mutuo.
2. Usualmente son un grupo pequeño, esto permiten lograr una comunicación efectiva entre los miembros del equipo.
3. Trabajan en forma conjunto con espíritu de grupo, convirtiéndose en un equipo cohesivo donde prima la sinergia de este. Para eso, se debe lograr un equipo maduro y estable, es decir, que no se agrega o cambia un integrante constantemente.
4. Sentido de responsabilidad como una unidad y se focalizan en el cumplimiento de las metas grupales.

En equipos cohesivos:
1. El grupo establece sus propios estándares de calidad.
2. Los individuos aprenden de los demás y se apoyan mutuamente.
3. El conocimiento se comparte.
4. Prevalece la mejora continua.

Stakeholders
Son los interesados del proyecto, incluye el equipo de proyecto, el equipo de dirección, el líder de proyecto y el patrocinador.

PLAN DE PROYECTO
El plan de proyecto, (o también llamado plan de desarrollo de software en el ámbito del software), es un artefacto de gestión que cumple la función de "hoja de ruta" de un proyecto, en la cual se documentan todas las decisiones que se toman a lo largo del desarrollo; en términos de: el proceso, la división de tareas a realizar, asignación de responsabilidades, equipos de trabajo, calendario de desarrollo, actividades de soporte (planes de contingencia, mecanismos para control de cambios, evaluar calidad y valorar riesgos).
En un plan de proyecto se establece qué se va a hacer (alcance del proyecto), cuándo se va a hacer (calendario), cómo se va a hacer (actividades o tareas para cubrir el alcance) y quién lo va a hacer (responsables de cubrir las actividades).
Responder a estas preguntas, implica las siguientes actividades:

- **Definición del Alcance del Proyecto.** El alcance de un proyecto, junto a su objetivo, responden al qué se está desarrollando. Es importante diferenciar el alcance de un producto y el del proyecto en sí. El **alcance de un producto** define qué **funcionalidades debe realizar el software** para satisfacer los requerimientos del cliente y se mide contra la Especificación de Requerimientos (ERS) (objetivos del producto), en cambio, el **alcance del proyecto** define **todo el trabajo y solo el trabajo** necesario que se debe realizar para cumplir con el desarrollo de este y poder entregar el producto o servicio con todas las características y funciones especificadas. El alcance del proyecto se mide contra el Plan de Proyecto (objetivos del proyecto).
  Tanto el alcance como el objetivo de un proyecto puede ser modificado a lo largo del proyecto, pero esto no necesariamente tiene que cambiar el alcance del producto. Por otra parte, el alcance de un producto sí condiciona los alcances de un proyecto.
  Es importante aclarar que el alcance del proyecto es menor que el alcance del producto.

- **Definición de Proceso y Ciclo de Vida.** Al definir el proceso que se va a utilizar, se responde al cómo se va a desarrollar el proyecto. Es importante tener en cuenta la información del contexto del desarrollo: objetivos, alcances, personas y recursos disponibles, clientes, forma de trabajo de personas, etc.
  También se debe tener en cuenta la complejidad del software a desarrollar (alcances y objetivos) para así poder adaptar el proceso definido que se elija al proyecto en el cual se está trabajando.
  El ciclo de vida del proyecto es lo que define cuánto de cada tarea se debe hacer y en qué momento del desarrollo del proyecto hacerlo.

El ciclo de vida de un proyecto define qué trabajo técnico debería realizarse en cada fase; **quién** debería estar involucrado en cada fase; **cómo controlar** y **aprobar** cada fase; **cómo** deben **generarse** los **entregables**; **cómo revisar**, verificar y **validar** el producto.
En metodologías tradicionales, se permite elegir cualquier tipo de ciclo de vida (cascada, iterativo e incremental, espiral, etc.), en cambio en metodologías ágiles esto no es posible (si o si tiene que ser iterativo e incremental).

- **Estimación.** En un proyecto de desarrollo de software, se trata de estimar para predecir el valor de un elemento relacionado con el proyecto o con el producto, por ejemplo, el tiempo que va a llevar, qué costo va a tener el sistema, cuántas personas necesito para realizar el desarrollo, etc. En un enfoque tradicional, esta estimación de valores es realizada únicamente por el líder de proyecto. En la metodología tradicional, existe un orden definido y recomendado para realizar las estimaciones:
  - *Tamaño:* para estimar el tamaño de un proyecto se utilizan los requerimientos tomados en la etapa de requisitos. Esta estimación es una de las más importantes a realizar, ya que el aumento o disminución del tamaño del producto, afecta de gran manera a otros aspectos. Esto es debido a que un aumento del tamaño trae consigo un aumento de complejidad del software, que puede repercutir de muchas maneras en el desarrollo. El tamaño de un producto se puede estimar en muchas escalas, por ejemplo: líneas de código, cantidad de requerimientos, puntos de función ajustados, etc.
  - *Esfuerzo:* refiere a la cantidad de horas persona lineales haciendo una actividad por vez, dentro del desarrollo del producto. Esta estimación intenta responder cuántas horas personas lineales serán necesarias para construir el producto, previamente estimado su tamaño.
  - *Calendario:* la estimación del calendario del proyecto permite responder al cuándo se van a realizar las tareas definidas. En el momento de la estimación, no es necesario realizar una calendarización muy específica a nivel de día, sino que se escala a tareas a realizar en, por ejemplo, un mes del año o incluso un trimestre. Es necesario realizar esta estimación para poder darle una idea al cliente de cuánto tiempo se va a demorar la finalización del proyecto, y como en los demás aspectos, es necesario ya tener una buena estimación del tamaño y del esfuerzo que se requiere, ya que el tiempo de duración del proyecto es proporcional a su crecimiento.
  - *Costos:* de forma similar a la estimación de la duración del proyecto, la estimación de costos depende directamente del tamaño y del esfuerzo que se necesite para el desarrollo del software, por eso es lo último que se estima. De la misma manera, la estimación de costos es muy útil para darle una idea al cliente de cuál es el costo aproximado del proyecto.

- **Gestión de Riesgos.** Un riesgo es la probabilidad de ocurrencia de una pérdida o daño que afecte de forma directa al proyecto y pueda comprometer el incumplimiento de su objetivo. Es de suma importancia identificar y definir los riesgos a los que está atado un proyecto, para así poder analizarlos, y realizar una cuantificación de estos, teniendo en cuenta la exposición del proyecto ante los mismos, para así definir a cuáles se debe atender y sobre cuáles se debe hacer un seguimiento.

1. **Identificación de riesgos**: se identifican los riesgos que suponen una mayor amenaza al proceso de ingeniería de software, al software a desarrollar o a la organización que lo desarrolla (al recibir los requerimientos).
2. **Análisis de riesgos**: para cada riesgo identificado se realiza un juicio acerca de la probabilidad y del impacto. No existe una forma certera de realizar esto. Se utilizan intervalos de probabilidad y clasificaciones de gravedad. El criterio dependerá de la experiencia de quien realice el análisis de riesgos.
3. **Planeación del riesgo**: para cada riesgo analizado, se definen estrategias para manejarlos. Estas estrategias consisten en considerar acciones a tomar para minimizar o evitar el impacto sobre el proyecto como consecuencia de la ocurrencia de la amenaza que representa el riesgo. Otras estrategias consisten en desarrollar planes de contingencia, el cual define un plan para enfrentar una situación controversial.
4. **Monitorización del riesgo y control**: proceso que permite determinar que las suposiciones acerca de los riesgos de producto, proceso y organización no han cambiado. Esto permite revalorar al riesgo en términos de posibles variaciones de su probabilidad e impacto. Este proceso se aplica en todas las etapas del proyecto.

- **Asignación de Recursos.** La asignación de responsabilidades a personas, permite asignar roles a los integrantes del equipo de trabajo. En el entorno de procesos definidos, estos roles se ven definidos en el concepto de trabajadores. Dentro de un mismo proyecto, una persona puede desenvolverse en más de un rol, ya que puede que el presupuesto del desarrollo de software no sea lo suficientemente grande para permitirse tener una persona distinta por rol, o bien, el proyecto no sea lo suficientemente grande o complejo para justificar dicha adición.

- **Programación de Proyectos.** En esta etapa se trata de definir en detalle, cada tarea que debe ser cumplida en el desarrollo. Se toma como base lo definido en la estimación del calendario realizada previamente, pero se refina al máximo detalle posible cada actividad.
  Cuando se habla de detallar las tareas, se hace referencia a descomponer el proceso de desarrollo en actividades refinadas a nivel de días, identificando quien la realiza, cuando debe realizarse y cuánto esfuerzo debe llevar en forma teórica

- **Definición de Controles.** En la planificación de los controles del desarrollo, se toma como base las métricas, y verifica que todo se está haciendo en concordancia con lo establecido. En esta planificación se definen reuniones periódicas, reportes o informes necesarios, etc.

- **Definición de Métricas.** Las métricas se definen como una medida numérica que aporta visibilidad sobre el avance o estado del proyecto, proceso o producto en un momento determinado del desarrollo. Al realizar la planificación del proyecto, es necesario definir de forma clara y no ambigua, cada una de las métricas asociadas a cada dominio (producto, proyecto o proceso) en conjunto con la forma de calcularlas. Además, es imprescindible que las métricas sean representativas para el entorno, es decir, que representen un aumento en la información y beneficien la practicidad del desarrollo y seguimiento.
  Las métricas se dividen en tres dominios, el cual, cada uno posee un foco distinto de medición:
  - **Métricas de Proceso:** son métricas estratégicas a nivel organizacional, orientadas a mejorar el proceso y tienen como objetivo generar indicadores que permitan mejorar los procesos de software a largo plazo. Son públicas y se despersonifican las mediciones de personas, áreas o proyectos particulares, para obtener un número aislado y tener una métrica sobre el proceso de la organización en general, como un todo.
    Ejemplo: desviación organizacional de estimaciones, esfuerzo realizado, tiempos de planificación promedio por proyecto, errores previos a releases por proyecto.
  - **Métricas de Proyecto:** están enfocadas a los recursos que se dedican al proyecto, como costos, esfuerzos, estimaciones y tiempo. Son responsabilidad del Líder de Proyecto y permiten al equipo adaptar el desarrollo de los proyectos y de las actividades técnicas. Estas métricas son privadas de ese proyecto y solo son visibles para los involucrados en el mismo.
    Se utilizan para mejorar la planificación del desarrollo, generando ajustes que eviten retrasos, reduzcan riesgos potenciales y por lo tanto, problemas.
    Además, se utilizan para evaluar la calidad de los productos en todo momento y en caso de ser necesario, modificar el enfoque para mejorar la calidad, minimizando defectos, retrabajo y por ende el costo total del proyecto.
    Ejemplo: eficiencia de estimación por proyecto, fechas de entrega reales vs programadas, cantidad de cambios y sus características.
  - **Métricas de Producto:** están enfocadas en lo que se construye, son responsabilidad del equipo de desarrollo y de testing y son particulares de ese producto.
    Se utilizan con propósitos técnicos y tienen como objetivo generar indicadores en tiempo real de la eficacia del análisis, el diseño, la estructura del código, la efectividad de los casos de prueba y calidad del software a construir.
    Ejemplo: cantidad de líneas de código por producto, promedio de métodos por clase, cantidad de casos de uso por complejidad.

  Métricas Básicas para un Proyecto de Software. Son las mínimas e imprescindibles que uno tiene que utilizar si tiene la intención de desarrollar una cultura de medición:
  - **Tamaño** del producto: asociada a métrica de producto.
  - **Esfuerzo**: asociada a métrica de proyecto.
  - **Calendario**: asociada a métrica de proyecto.
  - **Defectos**: asociada a métrica de producto.

  Estas métricas están relacionadas con la triple restricción en un proyecto (esfuerzo, alcance y tiempo).

Monitoreo y Control
El control de proyectos se refiere a comparar el progreso con respecto al plan, con el objetivo de verificar si estamos bien encaminados, de no ser así generar medidas preventivas que nos permitan volver a lo esperado, si ya se han detectado desviaciones deberán tomarse acciones correctivas.
El objetivo del monitoreo es determinar el estado del proyecto:

- ✓ **Proyecto bajo control**: se están alcanzando los hitos del proyecto a tiempo con los recursos estimados y con un nivel de calidad esperado. (Se cumplen Estándares y compromisos de planificación).
- ✓ **Proyecto fuera de control**: se deberá replanificar y renegociar el plan.

Es importante aclarar que los **proyectos se atrasan de a un día por vez**. Es decir los proyectos se atrasan por horas y se recuperan con días. Recuperar un proyecto atrasado depende del momento dentro del ciclo de vida en el cual estoy.

Factores para el éxito de un proyecto

- Monitoreo & Feedback
- Misión/objetivos claros
- Comunicación

Factores para el fracaso de un proyecto

- Fallas de toma de requerimientos
- Fallas al definir el problema
- Planificación basada en datos insuficientes
- Planificación realizada por grupo de planificaciones
- No hay seguimiento del plan de proyecto
- Mala planificación de recursos
- Estimaciones basadas en supuestos sin consultar datos históricos
- Nadie estaba a cargo

MANIFIESTO AGILE
Cuando hablamos de procesos empíricos vimos como estos procesos estaban guiados por una mentalidad y un enfoque agile, donde se asumen procesos cambiantes. Este enfoque agile surge de la necesidad de una entrega de mejores proyectos de software.
Agile surge de un grupo de desarrolladores de software independientes que se juntaron en 2001, y de esa reunión surgió el **Manifiesto Ágil**. Este grupo concluyo que descubren mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo. También, a través de este trabajo han llegado a los siguientes **valores agiles**:

- **Individuos e Interacciones** sobre Procesos y Herramientas

En metodologías agiles estoy centrado sobre los individuos, y por lo tanto los roles son intercambiables a diferencia de las metodologías tradicionales.

- **Software Funcionando** sobre Documentación Extensiva

Las metodologías agiles entrega software funcionando cada un rango de tiempo, siempre lo más corto posible. Si bien la documentación es importante, se priorizará entregar un producto que le entregue valor al cliente y le sea de utilidad.

- **Colaborar con el Cliente** sobre Negociación Contractual

Hay que estar dispuestos al cambio y cerca del cliente para predecirlo. Existen requerimientos que surgirán de la colaboración constante con el cliente, donde aparecen distintas alternativas que generan cambios en el producto.

- **Responder al Cambio** sobre Seguir un Plan

Los nuevos requerimientos pueden tener un mayor valor para el cliente que los iniciales, por lo que si se plantean cambias una vez comenzado el proyecto, deben ser recibidos y se incluirán dentro del alcance.

Estos cuatro valores agiles nos indican que, a pesar de que hay cierto valor en los ítems mencionados en la derecha, se valoraran más los ítems de la izquierda.

En ese entonces los proyectos de software estaban mayormente administrados utilizando un modelo de ciclo de vida en cascada, donde se realizaban distintas actividades o fases del proyecto, como análisis, diseño, desarrollo, testing y despliegue se hacían en una secuencia lineal y consecutiva, y donde se debía completar un fase para poder continuar con la siguiente. Esta metodología de trabajo no funcionaba, y entre un 20% a 50% de los proyectos de software fallaban.

Es por eso que con este manifiesto ágil, este grupo de personas busca sentar las bases de una nueva ideología de trabajo a la hora de desarrollar software, y para esto propuso **12 principios del manifiesto agile**, los cuales son:

1. Nuestra mayor prioridad es **satisfacer al cliente** mediante la entrega temprana y continua de software con valor.
2. **Aceptamos** que los **requerimientos 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 auto-organizados**.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación **ajustar** y perfeccionar **su comportamiento** en consecuencia.

AGIL

Cuando hablamos de agilismo no hablamos ni de una metodología ni de un proceso. Ágil es una **ideología** con un conjunto definido de principios que guían el desarrollo del producto.

Existen distintos valores que se siguen dentro de **equipos** que se consideran **agiles**, como por ejemplo seguir una *planificación* que sea *continua*, de multinivel; que los equipos sean *auto-organizados*, facultados y completos; que se realicen *entregas frecuentes*, iterativas y que estén priorizadas de acuerdo a lo que se considera de mayor valor para el cliente; que se utilicen *prácticas de ingeniería disciplinadas*; que se siga una *integración continua* del producto; y que se realice *testing* de manera *concurrente*.

El agilismo entonces, termina siendo un balance entre ningún proceso y demasiado proceso. La diferencia inmediata con otro tipo de procesos o metodologías, es que hay una exigencia de llevar una menor cantidad de documentación, aunque este punto no sea el más importante.

- Los métodos agiles son adaptables en lugar de predictivos.
- Los métodos agiles son orientados a la gente en lugar de orientados al proceso.

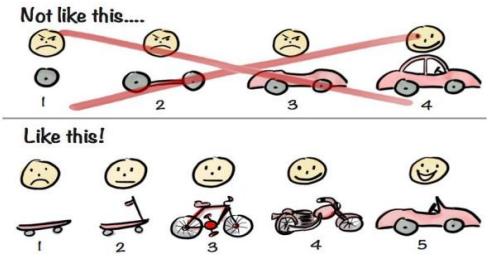
Estas son características fundamentales de la ideología ágil, y es de lo que venimos hablando. Un proceso ágil siempre deberá adaptarse a los cambios que vayan surgiendo a lo largo de un proyecto, ya que es precisamente para lo que están hechos, para poder responder ante problemas complejos y situaciones cambiantes, donde siempre lo más importante va a ser ofrecer el máximo valor al producto según los requerimientos del cliente. Y si el cliente quiere que el producto tenga funcionalidades o características que no fueron mencionadas o que involucren cambios en el plan, deben ser aceptadas y priorizadas correctamente.

Algunos de los frameworks agiles más importantes son Scrum, ATDD, FDD, Crystal, XP, entre otros.

Cuando es aplicable AGILE

Las principales razones por las cuales es un beneficio adoptar una ideología agile en un proyecto, es que nos permite realizar una entrega de producto que se encuentre funcionando y que le de valor al cliente de una manera más rápida y continua, nos ofrece una mejor administración de los cambios que surgen a lo largo del proceso, nos permite incrementar la productividad del equipo, y aumenta la visibilidad del proyecto.

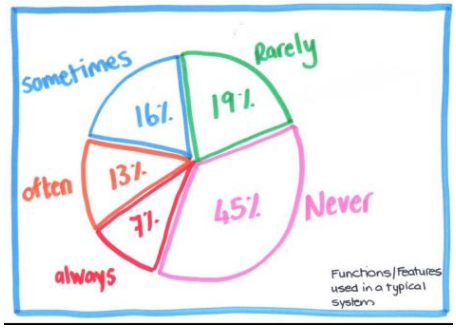
Agile se utilice mayormente en proyectos que involucren proceso complejos donde sabemos que los requerimientos pueden estar constantemente cambiando, o donde sabemos que pueden ingresar nuevas funcionalidades sobre la marcha. Esta ideología nos permite contar con un plan y una arquitectura que sean robustos y que puedan ser fácilmente adaptables en caso de que esto sea necesario.
Debemos tener en cuenta que ser agile no es ser indisciplinado, un mal uso del agilismo es no contar con un plan o con una arquitectura en nuestro proyecto, pero tampoco debemos llegar al otro extremo de contar con un plan y una arquitectura extremadamente grande por adelantado, ya estaríamos volviendo a los proyectos tradicionales, que es lo que queremos evitar.



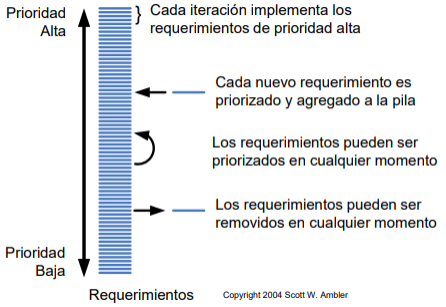
Agile no es realizar un producto donde cada entrega es una parte separada del mismo, agile es entregar siempre productos que sean funcionales para el cliente y que cumplan de alguna forma con el objetivo final por el cual se está construyendo. De esta forma, luego de cada entrega el cliente ya podra comenzar a utilizar el producto, aunque no se encuentre finalizado.

REQUERIMIENTOS AGILES

Como estuvimos hablando hasta recien, el agilismo lo que busca es la entrega rapida de un producto que ofrezca valor al cliente, es decir, entregar producto que el cliente pueda utilizar y que cumpla con lo que este pida. Es por eso, que en procesos agiles, se usa el **valor** para construir el producto correcto.
Cuando surge un problema, una nueva necesidad o una oportunidad dentro del proyecto, el equipo encargado de resolverla debe buscar cual es el camino mas efectivo para poder resolverla, y de esta forma generar una solucion que pueda ser entregable al cliente.
Para esto, se usan historias y distintos modelos para mostrar que es lo que se va a construir, y se debe poder determinar que es “solo lo suficiente” que se debe realizar para cumplir con los requerimientos del cliente.



Esto ultimo es muy importante, ya que la gran mayoria de los productos exitosos que encontramos en el mercado tienen también un desperdicio que es muy grande en cuanto a las funcionalidades que tiene. En la imagen podemos ver como el 45% de las funcionalidades de un sistema promedio no son utilizadas nunca por el cliente, es decir, se esta gastando tiempo en construir funcionalidades y features que el cliente jamas va a utilizar, mientras podriamos estar utilizando ese tiempo en realizar otras actividades que si entreguen valor.
Es por eso que es importante saber captar que es lo que el cliente espera del producto, y poder priorizar según cual será el aporte de valor para el cliente.



En el proyecto contaremos con una pila de requerimientos, que incluyen todas las funcionalidades que el cliente quiere que su producto tenga. Al trabajar con una **gestion agil de los requerimientos**, esto significa que en esta pila pueden estar constantemente ingresando requerimientos, pueden salir requerimientos que se detectaron que ya no seran necesarios, o también puede ser que los requerimientos que se encontraban ya en la pila, cambien.
Es por eso que la pila se ordenara por **prioridad**. Esta priorizacion de los requerimientos se encontra en función de, cuales son los requerimientos que mas **valor** le daran al cliente, cuales son los requerimientos que están definidos con un grado de **granularidad mas bajo**, es decir, de los cuales tenemos mas información y detalle de lo que se debe realizar, y de esta forma, en cada iteracion se tomaran los requerimientos que se encuentren en lo mas alto de la pila, ya que estos son aquellos que mayor prioridad tienen, en termino de los dos factores mencionados (valor y detalle o granularidad).
Los requerimientos pueden ser priorizados en cualquier momento de la vida del proyecto, es decir, podemos tener requerimientos que se encontraban mas abajo en la pila debido a que no se contaba con información suficiente sobre ellos, o porque no eran los que mas valor podian ofrecer al cliente, y por algún tipo de cambio, ya sea porque se realizo una reunion con el cliente y se pudo obtener mas información sobre ellos, o porque el cliente decidio que quiere este cambio para la proxima entrega, estos requerimientos pueden ser priorizados y escalar dentro de la pila, ya que ahora su prioridad es mas alta.

En cualquier momento pueden ingresar nuevos requerimientos, y cuando esto ocurra, los mismo deberan ser priorizados, y se agregaran a la pila en función de dicha prioridad. De la misma forma, también se puede remover requermientos en cualquier momento, ya que no son necesarios.

Cabe aclarar que cuando hablamos de **valor** para el cliente, lo asociamos con la **utilidad**, **beneficio** o **satisfaccion** que le ofrece a los usuarios finales cada funcionalidad completa que le entregmos. El valor será la obtencion de un beneficio ya sea tangible o intangible.

Existen distintos **tipos de requerimientos**:

- **Requerimiento de Negocio.** Disminuir un X% de tiempo invertido en proceso manuales relacionados con atencion al cliente.
- **Requerimiento de Usuario.** Realizar consultas en línea del estado de cuenta de los clientes.
- **Requerimiento Funcional.** Generar reporte de saldos de cuenta. Recibir notificaciones por mail.
- **Requerimiento No Funcional.** Formato del reporte PDF. Cumplir niveles de seguridad para credenciales de usuarios según la ley bancaria 9999XX.
- **Requerimiento de Implementación.** Servidores en la nube.

De todos estos tipos de requerimientos, tantos los Requerimientos de Negocio como los Requerimientos de Usuario van a pertenecer al **dominio del problema**, mientras que el resto de los requerimientos pertenecerán al **dominio de la solución**.

De esta forma, como un resumen de lo explicado, algunas de las actividades que se deben hacer en un proyecto con ideología ágil es **entender** cuál es la **necesidad** o necesidades que el cliente tiene con respecto al producto que quiere obtener, y cuál es el **negocio** para el cual se estará construyendo el producto. Una vez que conocemos esto, deberemos encargarnos de descubrir cual será la **solución más efectiva** para poder resolver este problema o estas necesidades que tiene el cliente, de forma que podamos ofrecer un producto con valor, y esto se deberá hacer de forma colaborativa, en un equipo, detectando cuales son los distintos requerimientos que irán a la pila de requerimientos, cada uno de estos con una prioridad especifica. Luego, junto a un **equipo motivado y competente** con las habilidades para poder **construir el producto** especificado, en cada iteración se tomarán los requerimientos cuya prioridad sea la más alta y se encargarán de resolverlos utilizando las herramientas necesarias, de forma que finalmente se pueda **entregar** de manera **frecuente** a los stakeholders **productos** que les puedan ofrecer cierto **valor** y cumpla con los requerimientos y las necesidades planteadas en un inicio.
Se debe tener en cuenta que:

- La única constante son los cambios
- Los stakeholders son todos los que están
- Siempre se cumple que “El usuario dice lo que quiere cuando recibe lo que pidio”
- No hay tecnicas ni herramientas que sirvan para todos los casos
- Lo importante no es entregar una salida o un requerimiento, lo importante es entregar un resultado, una solucion de valor.

Principios Agiles relacionados a Requerimientos Agiles

1- La prioridad es satisfsacer al cliente a través de releases tempranos y frecuentes (de 2 semanas a 1 mes).
2- Recibir cambios de requerimientos aun en etapas finales.
4- Tecnicos y no tecnicos trabajando juntos todo el proyecto.
6- El medio de comunicación por excelencia es el cara a cara.
11- Las mejores arquitecturas, diseño y requerimientos emergen de equipos auto-organizados.

USER STORIES

Una user story contiene una descripcion corta de una funcionalidad valuada por un usuario o cliente de un sistema. Se llaman “stories” porque se supone que quien la describe esta contando una historia, y de esta forma, lo se escribe en la tarjeta no es lo mas importa, sino la conversacion entre los clientes y analistas o desarrolladores acerca de la historia.
La parte mas dificil de construir un sistema de software es decidir precisamente que construir.

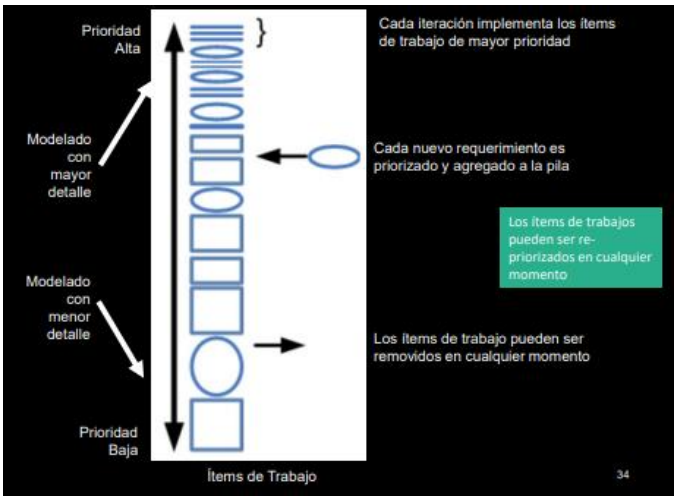
- Ninguna otra parte del trabajo conceptual es tan dificil como establecer los requerimientos tecnicos detallados.
- Ninguna otra parte del trabajo afectara tanto al sistema resultante si se hace incorrectamente.
- Ninguna otra parte es tan dificil de rectificar mas adelante.

Las user stories son:
- Una necesidad del usuario.
- Una descripcion de una funcionalidad del producto.
- Un item de planificacion.
- Un token para una conversacion.
- Un mecanismo para diferir una conversacion.

Estrctura de una User Story
COMO **<< rol de usuario >>** *QUIERO* **<< actividad >>** *PARA* **<< valor de negocio >>**

- **Rol de Usuario:** representa quien esta realizando la accion o quien recibe el valor de la actividad.
- **Actividad:** representa la acción que realizara el sistema, y usualmente se utiliza la frase verbal de la user.
- **Valor de Negocio:** comunica porque es necesario la actividad, es decir, de qué forma agrega valor al negocio y cuál es el objetivo de dicha actividad.

Componentes de una User Story
1. **Tarjeta:** es una descripcion detallada de la historia, utilizada para planificar y como recordatorio. Expresa el valor que se quiere conseguir desde el punto de vista del rol de usuario.
2. **Conversacion:** son las discusiones acerca de la historia que sirven para desarrollar los detalles de la misma, es la parte más importante de la user. Se puede dar en cualquier momento, lo más habitual es que se produzca durante el refinamiento del backlog o la sprint planning.
3. **Confirmación:** son las pruebas que expresan y documentan los detalles, y que pueden usarse para determinar cuándo una historia está completa. Es un acuerdo que refleja que todas las personas implicadas (PO y Dev Team) entienden cuáles son los elementos, valor y resultado esperado de la historia.



Cuando hablamos de requerimientos agiles hablamos de una pila de requerimientos, formada justamente por distintos requerimientos que implicaban funcionalidades a incorporar en el producto.
En el contexto de Scrum, esta pila de requerimientos se convierte en un **product backlog**, el cual estará formado por distintos ítems, que en general son historias de usuario. En este sentido, el **product owner** es el encargado de priorizar las historias de usuario, lo que funciona de la misma manera que lo explicado en la pila de requerimientos.
Hay requerimientos con **mayor grado de detalle y una prioridad más alta** y serán los que se implementarán en la siguiente iteración. Cada nuevo requerimiento que ingresa al product backlog es priorizado y agregado a la pila. Los ítems de trabajo (user stories en este caso) pueden ser re-priorizados en cualquier momento, y también pueden ser removidos.

Criterios de Aceptación de una User Story
Los criterios de aceptación de una US son **condiciones de satisfaccion** ubicadas en el sistema, y deifnen limites para dicha US. Estos ayuda que el PO responda lo que necesita para que la US **provea valor** (RF minimos). Son las condiciones que deben cumplirse para determinar que la historia fue desarrollada completamente, ya que de lo contrario, los desarrolladores no tendrian un parametro para definir si la misma fue cumplimantada o no.
Ayudan a que el equipo tenga una vision compartida de la US, ayudan a desarrolladores y testers a derivar las pruebas, y ayudan a los desarrolladores a saber cuando parar de agregar funcionalidad en una US.
Para que los criterios sean buenos deben definir una intencion, no una solucion; son independientes de la implementación; y son relativamente de alto nivel, ya que no es necesario que se escriba cada detalle.

Detalles de una User Story
Los detalles como “El encabezado de la columna se nombra ‘Saldo’” o “Deberia usarse una lista desplegable en lugar de un checkbox” se obtienen de **conversaciones** entre el dueño del producto y el equipo. Otra forma es que el equipo capture estos detalles de dos lugares: **documentación interna** de los equipos, y **pruebas de aceptación** automatizadas.

Pruebas de Aceptacion de una User Story
Las pruebas de aceptación expresan detalles resultante de las conversaciones entre clientes y el equipo; suelen usarse para completar detalles de la US y se ven como un proceso de dos pasos:
1. Identificarlas al dorso de la US.
2. Diseñar las pruebas completas que se utilizan para demostrar que la historia se ha hecho correctamente y se ha codificado en forma completa.

Las mismas deben escribirse antes que la programacion empiece (las escribe el cliente o PO, o al menos especifica que pruebas se utilizaran para determinar si la historia ha sido desarrollada correctamente).

Definition of Ready. Expresa cuando un requerimiento esta listo para poder ingresar en una sprint. El DoR deja en claro cuando el equipo comprende que la historia esta lo suficientemente bien definida para poder ingresar en la sprint y comenzar a ser desarrollada. Es decir, la US ya se encuentra con sus criterios y pruebas de aceptación con el suficiente nivel de detalle requerido como para poder entrar en una sprint.

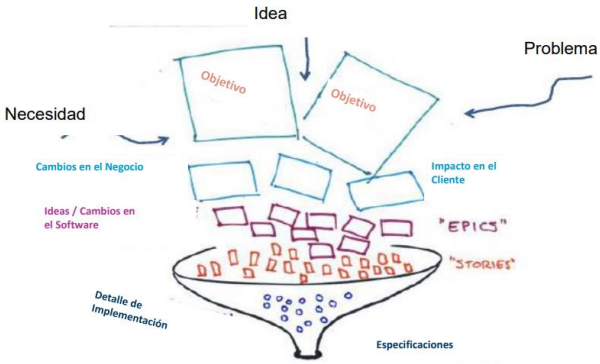
Definition of Done. Expresa cuando una funcionalidad del producto este lista para salir de una sprint. Es decir, el DoD representa cuando una historia de usuario esta lo suficientemente construida, desarrollada y testeada como para poder ofrecer valor al cliente. Deja en claro una serie de estandares de calidad, que deben ser definidos por el equipo, que debe requerir una US para poder ser incluida en un release.

INVEST Model
El acronimo INVEST nos ayuda a caracterizar una buena historia de usuario:
- **Independent:** las historias son mas faciles de trabajar si son independientes, en el sentido en que no queremos que se superpongan en concepto, y puedan ser calendarizables e implementables en cualquier orden.
- **Negotiable:** esto no refiere a una contrato explicito para cada feature, sino que refiere a que los detalles sean creados en conjunto entre el cliente y el equipo durante el desarrollo. Una buena historia de usuario captura la esencia, no los detalles. Es decir, la US debe incluir el “que” (se debe hacer), no el “como” (se debe hacer).
- **Valuable:** no nos interesa si la US entrega valor a nadie, esta debe ser valiosa para el cliente. Los desarrolladores pueden tener preocupaciones (legitimas), pero las US están enmarcadas de una manera que hace que el cliente las perciba como importantes.
- **Estimable:** no se necesita una estimación exacta, pero si lo suficiente como para ayudar al cliente a rankear y calendarizar la implementación de la historia basado en costos. Que una US sea estimable tiene relación con que sea negociable, ya que es difícil estimar una historia que no entendemos. También se debe tener en cuenta que las historias muy grandes son más difíciles de estimar; y también que la estimación va a variar dependiendo de la experiencia de los integrantes del equipo, y de la experiencia del equipo trabajando juntos.
- **Small:** las historias usualmente representan como mucho unas pocas semanas de trabajo de una persona. Por encima de este tamaño, puede ser un poco complicado saber cuál es el alcance de la historia. Historias más pequeñas tienden a tener una estimación más precisa. Además, siempre se busca que la historia pueda ser completada en una iteración, ya que si es más larga, debería ser dividida seguramente.
- **Testeable:** escribir una US lleva consigo una promesa implícita de que se entiende lo que se quiere lo suficientemente bien como para escribir un testeo para dicha historia. Muchos equipos reportan que al requerir al cliente test antes de la implementación de la historia, el equipo es más productivo. La testeabilidad siempre fue una característica de un buen requerimiento, y escribir los test nos ayuda si el objetivo es conocido.

Algo mas sobre las User Stories
- No son especificacion detalladas de requerimientos (como los CU).
- Son expresiones de intencion, “es necesario que algo como esto …”.
- No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
- Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.
- Junto con el código, sirven de entrada a la documentación que se desarrolla incrementalmente después.

Niveles de Abstracción
- **Spikes.** Es un tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una US u otra faceta del proyecto. Se clasifican en **tecnicas** y **funcionales**, y se pueden utilizar con distintos objetivos:

- Una inversión básica para familiarizar el equipo con una nueva tecnología o dominio.
- Analizar un comportamiento de una historia compleja y poder así dividirla en piezas manejables.
- Ganar confianza frente a riesgos tecnológicos, investigando o prototipado para ganar confianza.
- Frente a riesgos funcionales, donde no está claro cómo el sistema debe resolver la interacción con el usuario para alcanzar el beneficio esperado.



- ✓ **Spikes Técnicas:** son utilizadas para investigar enfoques técnicos en el dominio de la solución, por ejemplo, evaluar performance potencial, decisión de hacer o comparar, o evaluar la implementación de cierta tecnología. Sirven para cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.
- ✓ **Spikes Funcionales:** son utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema. Usualmente son mejores evaluadas con prototipos para obtener una realimentación de los usuarios o los involucrados.

Existen historias que requieren de ambos tipos de spikes, y en ese caso el equipo puede crear dos spikes, una que sea un spike técnico y por otro lado un spike funcional, cada una para resolver los distintos aspectos planteados.

Las spikes deben ser **estimables**, **demostrables** y **aceptables**. Además, se debe tener en cuenta que las spikes son la excepción y no la regla, es decir, toda historia va a tener un grado de incertidumbre y de riesgo, y el objetivo del equipo va a ser aprender a aceptar y resolver cierta incertidumbre en cada iteración, es por esto que las spikes deben dejarse para incógnitas más críticas y grandes, y se las debe tratar como la última opción.

También debemos saber que el spike se debe implementar en una iteración separada de las historias resultantes, salvo que el spike sea pequeño y sencillo y sea probable encontrar una solución rápida, en cuyo caso la spike y la historia pueden incluirse en la misma interacción.

- **Épicas.** Una épica captura un largo cuerpo de trabajo. Esto es esencialmente una historia de usuario larga que puede ser dividida en un número de historias de usuario más pequeño. Una épica es un gran conjunto de requerimientos de usuario y de features del producto. Una épica puede tomar varias sprints para ser completada.

- **Theme.** Un tema es un conjunto de historias de usuario relacionadas con algún aspecto en común, cómo por ejemplo, ser del mismo área funcional.

Se debe dejar en claro que, debemos diferir el análisis detallado tan tarde como sea posible, lo que es justo antes de que el trabajo comience; hasta entonces se deben capturar los requerimientos en forma de user stories; y debemos entender que las user stories no son requerimientos de software y no necesitan ser descripciones exhaustivas de la funcionalidad del sistema.

Para que las user stories sean útiles para el equipo, estás deben ir **de un paso a la vez** (evitar la palabra "y"), deben usar **palabras claras** en los criterios de aceptación, no se debe olvidar la parte que no es visible: la **conversación**, se deben escribir desde la **perspectiva del usuario** y **no** se debe **forzar** todo para escribirlo como user story.

ESTIMACIONES DE SOFTWARE

Una estimación es una predicción que tiene como objetivo predecir esfuerzo tiempo y costos que llevará completar una determinada tarea.

En cuanto a las estimaciones debemos tener en cuenta que,

- Por definición una estimación **no es precisa**, la mayor cantidad de veces nos equivocamos al estimar.
- **Estimar no es planear** y planear no es estimar.
- Las estimaciones son la **base de los planes** pero los planes no tienen que ser lo mismo que lo estimado.
- Mientras **mayor diferencia** haya entre lo estimado y lo planeado **mayor será el riesgo**.
- Las estimaciones **no son compromisos**.

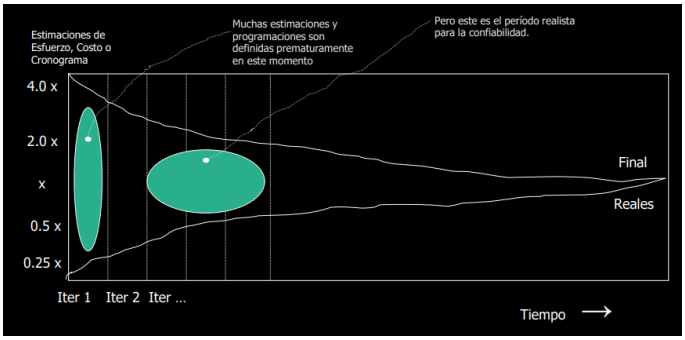
Los momentos más apropiados para estimar son al inicio del proyecto, luego de las especificaciones de requerimientos, o luego del diseño. Además, debemos notar que la extinción es una de las actividades más complejas en el desarrollo de software, luego de la definición del mismo.

Para Que Estimamos

Las dos principales razones por las cuales se llevan adelante las estimaciones son para *predecir completitud* y para *administrar riesgos*.

Antes de que el proyecto comience el líder del proyecto y el equipo de desarrollo deben estimar el trabajo que habrá que realizar, los recursos que se requerirán y el tiempo que transcurrirá desde el principio hasta el final. Es por eso que las estimaciones requieren de experiencia, acceso a una buena información histórica, es decir, a métricas, y de valor para comprometerse con predicciones cuantitativas cuando la información cualitativa es todo lo que existe.

Siempre se desea saber al inicio del proyecto cuál será el costo y el tiempo que llevará pero como aún no se conoce con precisión cuál será el alcance del producto resulta complejo predecir con exactitud estos aspectos, y es por eso que a medida que avanza el proyecto las estimaciones serán más certeras, dado que se cuenta con una mayor información sobre la cual realizar las estimaciones



Al inicio el desarrollo hay una estimación errónea de 2 a cuatro veces más, y esto se debe a distintos motivos. Al inicio del proyecto tenemos poca información y demasiada incertidumbre, lo que aumenta el riesgo de generar estimaciones erróneas. Además, las estimaciones son de naturaleza optimista, las estimaciones deben ser recalculadas conforme se avanza con el proyecto.

Errores en las Estimaciones

- Información imprecisa acerca del software a estimar o acerca de la capacidad para realizar el proyecto.
- Demasiado caos en el proyecto, es decir, está mal definido.
- Existe imprecisión generada por el proceso de estimación.
- Se omiten actividades necesarias para la estimación del proyecto, tales como requerimientos faltantes, licencias, reuniones, revisiones, etc.

Técnicas Fundamentales de Estimación

Contar. Contar es la técnicas fundamental de las estimaciones, pero la pregunta es, ¿que contar?

- En etapas tempranas: requerimientos, características, casos de uso, user stories, etc.
- En la mitad del proyecto: puntos de función, pedidos de cambio, páginas web, reportes, pantallas, tablas, etc.
- Mas avanzado el proyecto: defectos, clases, tareas, etc.

Se debe contar aquello que esté muy relacionado con el tamaño del software que se está estimando.

Se debe buscar aquello que esté disponible lo más pronto en el proyecto.

Se debe entender lo que se cuenta, es decir, si se contrasta con datos históricos se debe asegurar que se utilizan las mismas presunciones que se utilizaron en el pasado.

Lo que se cuenta debe requerir poco esfuerzo.

Métodos Utilizados

Lo más recomendado es utilizar distintos métodos de estimación y luego contrastar los resultados. Todos los métodos incluyen un “factor de ajuste” que permite que el método cierre (calibraciones).

La calibración se realiza contra la industria la organización o el mismo proyecto con etapas anteriores. De todas estas opciones, compararse contra la industria es la más riesgosa porque depende del contexto.

Las estimaciones no son una promesa ni certeza, ya que se asumen muchas variables desconocidas y elementos no tenidos en cuenta (tiempo de reuniones, tiempo de armado del set de pruebas, tiempo de revisiones, etc.).

Basados en la Experiencia

- ✓ **Datos Históricos.** Se deben recolectar los datos básicos del desarrollo de los proyectos, como el tamaño, esfuerzo, tiempo, defectos, entre otros, de modo de ir generando una base de conocimiento que sea de utilidad para futuras estimaciones. Con esto se busca una alternativa en la cual el conocimiento de cada individuo se transfiere a la organización y no estamos atados a una única persona.
  - Debemos dejar de lado el optimismo de pensar que no se van a tener los problemas que se enfrentaron en el pasado.
  - La productividad de una organización es un atributo que no tendrá gran variación entre los proyectos.

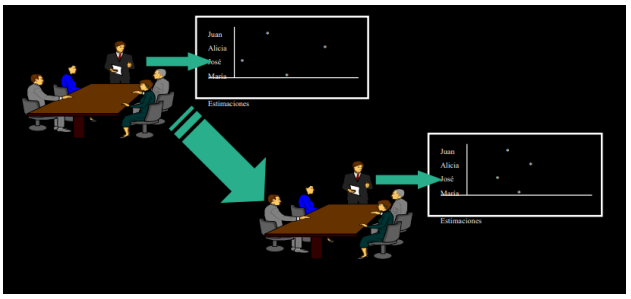
- Los datos históricos se deben utilizar para evitar discusiones entre desarrolladores y clientes.
- A la hora de recolectar datos históricos se debe tener en cuenta que se haga de la misma manera en todos los proyectos.
  - Tamaño: ¿Cómo se cuenta el código rehusado?
  - Esfuerzo: ¿Qué unidad se utiliza?¿Como se cuenta el tiempo en el proyecto?
  - Tiempo: ¿Cuándo comenzó?¿Cuándo tenemos aprobado el presupuesto?
  - Defectos: ¿Cuántos defectos se originaron en el diseño?

- *Industry Data:* datos de otras organizaciones que aportan al mercado y desarrollan productos con algún grado de semejanza y que permite una comparación básica. (En general están es estándares).
- *Historical Data:* datos de la organización de proyectos que se desarrollaron y ya se cerraron.(Se guardaron datos-estimaciones).
- *Project Data:* datos del proyecto pero de etapas anteriores a la que se está estimando.(Cada vez que se abre una fase estimarla).

✓ **Juicio Experto.** Es el método más utilizado en la práctica y se calcula que alrededor del 75% de las organizaciones utilizan este método de estimación, pero el problema se presenta cuando se utiliza como única técnica de estimación el juicio experto puro.

❖ *Juicio Experto Puro:* En esta técnica un experto estudia las especificaciones y hace su estimación. Para esto se basa fundamentalmente en sus propios conocimientos, y no es beneficioso utilizar solo esta técnica debido a que si desaparece el experto la empresa deja de estimar.
El juicio del experto se estructura de la siguiente manera:
- Debe tener tareas de una granularidad aceptable.
- Se debe utilizar el método **"optimista, pesimista y habitual"** y su fórmula = **(o + 4h + p)/6**.
- Se debe usar un checklist y un criterio definido para asegurar la cobertura, esto tanto para asegurarnos que estamos estimando correctamente, y también para tener cobertura completa de lo que vamos a estimar.

❖ *Juicio Experto: Wideband Delphi:* es similar al anterior pero ahora se trata de un grupo de personas que se reúne con el objetivo de determinar lo que costará el desarrollo tanto en esfuerzo como en duración. Este tipo de estimaciones en grupo suelen ser mejores que las individuales.
Para esto se siguen una serie de pasos:



1. Se dan las especificaciones a un grupo de expertos.
2. Se les reúne para que discutan tanto del producto como la estimación.
3. Los expertos remiten sus estimaciones individuales al coordinador.
4. Cada estimador recibe información sobre su estimación y las ajenas pero de forma anónima.
5. Se reúnen de nuevo para discutir las estimaciones.
6. Cada uno revisa su propia estimación y las envía al coordinador.
7. Se repite el proceso hasta que la estimación converge de forma razonable. Si no converge luego de sucesivas iteraciones, se utiliza el enfoque optimista-pesimista-habitual, o bien la desviación estándar entre observaciones.

✓ **Analogía.** Se compara con situaciones anteriores para poder estimar. Para que funcione debe determinarse de antemano cuales son los factores relevantes a tener en cuenta para buscar proyectos parecidos (ejemplo: cantidad de casos de uso por complejidad, cantidad de usuarios, tipos de tecnologías, equipos de trabajo asignado, etc.) Es el método que mayor error tiene.
Factores:
- Tamaño: ¿Mayor o menor?
- Complejidad: ¿Más complejo de lo usual?
- Usuarios: Si hay más usuarios habrá más complicaciones
- Otros factores:
  - Sistema operativo, entornos (la primera vez más).
  - Hardware, ¿Es la primera vez que se va a utilizar?
  - Personal del proyecto, ¿nuevos en la organización?

Basados exclusivamente en los Recursos. En la estimación consiste en ver de cuanto personal y durante cuánto tiempo se dispone de él, haciendo las estimaciones exclusivamente en función de este factor. En la realización: "El trabajo se expande hasta consumir todos los recursos disponibles, independientemente si se alcanzan o no los objetivos."

Basados exclusivamente en el Mercado
- Lo importante es conseguir el contrato.
- El precio se fija en función de lo que creemos que está dispuesto a pagar el cliente.
- Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta.
- Peligroso si es el único método utilizado.
- Actualmente en desuso dado que se utilizaba con clientes cautivos.

Basados en los Componentes del Producto o en el Proceso de Desarrollo. Cuando hablamos de producto, hacemos referencia a módulos y sus respectivas funciones, mientras que por procesos el resultado es la WBS (descomposición del proyecto en partes elementales). Dos técnicas:
- **Bottom-up**. Se descompone el proyecto en las unidades lo menores posibles. Se estima cada unidad y se calcula el coste total.
- **Top-Down**. Se ve todo el proyecto, se descompone en grandes bloques o fases. Se estima el coste de cada componente.

Métodos Algorítmicos. En base al producto se realiza la ejecución de algunos algoritmos que contemplan algunas características del producto tales como tamaño, complejidad, conocimiento del dominio, etc. (factores multiplicadores), esto genera como resultado una estimación del esfuerzo.
Que sea un cálculo matemático no quiere decir que el resultado sea certero.

(TERMINAN LOS MÉTODOS DE ESTIMACIÓN)

Actividades Omitidas
- Una de la fuentes de error más común en las estimaciones es omitir actividades necesarias para las estimación del proyecto.
  - Requerimientos faltantes.
  - Actividades de desarrollo faltantes (documentación técnica, participación en revisiones, creación de datos para el testing, mantenimiento de producto en previas versiones).
  - Actividades generales. (días por enfermada, licencias, cursos, reuniones de compañía).
- **SOLUCIÓN:** Uso de buffers

"Nunca tenga temor que estimaciones creadas por desarrolladores sean demasiado pesimistas, dado que los desarrolladores siempre generan cronogramas demasiado optimistas".

ESTIMACIONES AGILES
Cuando realizamos estimaciones no debemos ser dogmáticos sobre nada, sino que la clave del existo es ser pragmático.
Algunas cosas que debemos saber sobre las estimaciones son que si estas se utilizan como **compromisos**, pueden ser **muy peligrosas** y perjudiciales para cualquier organización, ya que se comprometen a realizar ciertas tareas basándose en aspectos que posiblemente lleven a una mala estimación, y por lo tanto no puedan cumplir. Lo más beneficioso en las estimaciones es el "**proceso de hacerlas**". La estimación podría servir como una gran **respuesta temprana** sobre si el trabajo planificado es **factible** o no, y también puede servir como una **gran protección al equipo**.

Cuando hablamos de ESTIMACIÓN ÁGIL las features/stories son estimadas usando una medida de tamaño relativo conocida como **story points (SP)**. Se debe dejar en claro que los SP no son una medida basada en el tiempo, sino que, como dijimos, es una medida relativa.
La utilización de una **estimación relativa** nos ofrece beneficios, debido a que las personas no saben estimar en términos absolutos, y son mejores comparando cosas, además de que comparar es generalmente más rápido. Con este tipo de estimaciones se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual, y también se emplea mejor el tiempo de análisis de las stories.

Estimación de Tamaño, Esfuerzo y Tiempo
La palabra **tamaño** refiere a cuan grande o pequeño es algo. En este caso, el tamaño es una medida de la cantidad de trabajo necesaria para producir una feature/story.
El tamaño nos indica cuan compleja es la story, cuanto trabajo es requerido para hacer o completar una story, y cuan grande es.

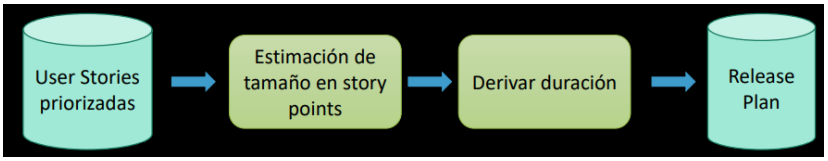
Por otro lado, las estimaciones basadas en **tiempo** son más propensas a errores debido a varias razones. Entre ellas, las habilidades, el conocimiento, las asunciones, la experiencia, la familiaridad con los dominios de aplicación/negocio.

Debemos tener en cuenta que **tamaño NO ES esfuerzo**.
Que medidas usamos entonces para medir el tamaño, algunas de ellas pueden ser:
- Tamaño por números: 1 a 10
- Talles de remeras: S, M, L, XL, XXL
- Serie 2n : 1, 2,4, 8, 16, 32, 64, etc.
- Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.

Una vez que elegimos una medida o escala, no se cambia, ya que si se cambia debemos cambiar el metro patrón.

STORY POINTS
Un story point es una medida especifica (del equipo) de complejidad, riesgo y esfuerzo, y es lo la analogía de un kilo en el sistema de medición de peso. El story point da idea del "peso" de cada story y decide cuan grande (compleja) es. La complejidad de una story tiende a incrementarse exponencialmente, por eso la escala más utilizada suele ser la de fibonacci.

Velocidad (Velocity). La velocidad es una medida o métrica del progreso de un equipo. Se calcula sumando el número de story points (asignados a cada US) que el equipo **completa** durante la **iteración**. Cabe recalcar que únicamente se consideraran los puntos de las historias completadas completamente, y no se cuenta nada de las parcialmente completadas. La velocidad corrige los errores de estimación.
La velocidad no se estima, se calcula; solo cuenta el trabajo completado; no es un objetivo a alcanzar; no es comparable entre equipos; no es comparable entre proyectos; sirve para observar empíricamente la capacidad de trabajo de un equipo pero no para estimar los objetivos.
Si utilizamos scrum, la velocidad va a ser más baja si se corren ciclos de testing y se encuentran bugs que deben ser fixeados.



Estimación de un Proyecto
La duración de un proyecto no se estima, sino que se deriva tomando el número total de story points de las US y dividiéndolo por la velocidad del equipo.
La velocidad nos ayuda a determinar un horizonte de planificación apropiado, y la estimación en SP separa completamente la estimación de esfuerzo de la estimación de la duración.

Poker Estimation (Propuesta de Método de Estimación Ágil)
Este método es una técnica de estimación popular entre los practicantes de las metodologías agiles, y fue publicado por Mike Cohn. Resulta de la combinación de distintos métodos de estimación: opinión de experto, analogía y desagregación.
En la misma, los participantes en poker planning son desarrolladores, ya que se dice que las personas más competentes para resolver una tarea deben ser quienes las estiman. El PO participa en la planificación pero no estima. Existe un moderador, que es frecuentemente el dueño del producto o el analista, pero de todas formas, este puede ser cualquiera ya que no hay privilegios.
Para esta técnica se suele utilizar la escala de fibonacci, ya que la misma crece en forma exponencial, al igual que lo hace la complejidad en el software. Estos valores se colocan en cartas para realizar la estimación, la cual seguirá una serie de pasos.

Como prerrequisitos en este método de estimación agile se debe contar con una lista de features/story que van a ser estimadas, y cada estimador debe tener un mazo de cartas con estos valores.
1. Lo primero que se debe hacer es elegir un **base story** o **historia canónica**, que será utilizada para comparar con las demas stories a la hora de estimar.

1.1 Elegida la canónica, se lee a todo el equipo la historia que se va a estimar.
1.2 Los estimadores discuten la historia, haciendo preguntas al PO si es necesario en base al posible desconocimiento de ciertos aspectos, o porque tienen dudas sobre la misma.
1.3 Cada estimador selecciona una carta con el numero que estima sobre dicha story y la coloca boca abajo en la mesa.
1.4 Cuando todos pusieron las cartas, se exponen al mismo tiempo.
1.5 Si todos los estimadores seleccionaron el mismo valor, será ese el valor estimado. Sino, se debe comenzar una discusion sobre los resultados, poniendo atencion en los valores extremos. Luego de esta discusion, se vuelve a seleccionar una carta y se repite el proceso desde ese punto, hasta llegar a un acuerdo.

2. Se toma la proxima story, se discuto con el PO.
3. Cada estimador asigna un valor por comparacion contra la base story, y se vuelve a realizar todo el proceso.

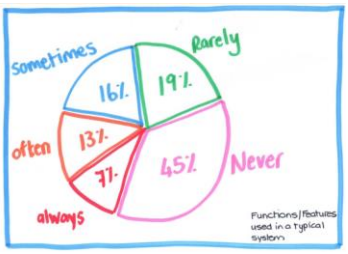
No es una definición concreta de cada valor, pero mas o menos se podria decodificar cada uno de los valores de las estimaciones de la siguiente manera:

- 0: Quizás usted no tenga idea de su producto o funcionalidad en este punto.
- ½ , 1: funcionalidad pequeña (usualmente cosmética).
- 2-3: funcionalidad pequeña a mediana. Es lo que queremos. ☺
- 5: Funcionalidad media. Es lo que queremos. ☺
- 8: Funcionalidad grande, de todas formas lo podemos hacer, pero hay que preguntarse si no se puede partir o dividir en algo más pequeño. No es lo mejor, pero todavía. ☺
- 13: Alguien puede explicar por qué no lo podemos dividir?
- 20: Cuál es la razón de negocio que justifica semejante story y más fuerte aún, por qué no se puede dividir?
- 40: no hay forma de hacer esto en un sprint.
- 100: confirmación de que está algo muy mal. Mejor ni arrancar.

GESTIÓN DE PRODUCTOS

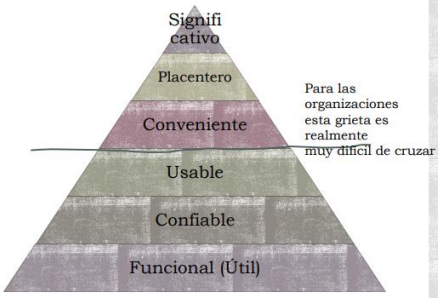
Existen 4 razones por las cuales se crean nuevos productos:

- **Para satisfacer a los clientes.** Un cliente viene con una propuesta de producto y un equipo de trabajo se encarga de la planificación, diseño y construcción de dicho producto siguiendo los requerimientos que el cliente quiere.
- **Para tener muchos usuarios logueados.** Se comienza el desarrollo de un producto únicamente con el objetivo que la plataforma tenga muchas visitas y atraiga a la mayor cantidad de usuarios posible.
- **Para obtener mucho dinero.** No hace falta hacer mucha aclaración, se crea un producto para obtener un beneficio económico, ya sea por medio de suscripciones, un pago mensual, etc.
- **Realizar una gran visión, cambiar el mundo.** Se crea un producto con el objetivo de, mas que obtener dinero o tener muchos usuarios, realizar un cambio en algún aspecto del mundo, ya sea más en un ámbito ecológico-ambiental, o puede ser relacionado a crear una revolución digital (como lo fue whatsapp).



A la hora de crear un producto, debemos siempre pensar en quienes van a utilizarlo, y si realmente todas las funcionalidades con las que cuenta el producto van a aportar valor al usuario que las utilice. Cómo se ve en el gráfico en muchos productos existen funcionalidades que no son utilizadas nunca por el usuario. Es por esto que a la hora de definir una nueva funcionalidad para agregar en una aplicación debemos pensar si realmente vale la pena agregarla.

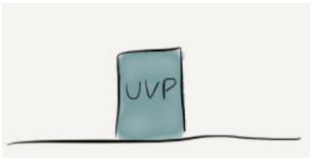
Evolución de los Productos de Software



Cuando vamos a desarrollar un software la mayoría de las veces se empieza a construir un producto sobre la base de lo funcional, que haga lo que tiene que hacer (las tareas que debe realizar), luego si se quiere avanzar más, el software debe ser confiable donde los usuarios no corran peligro usando el producto y los resultados que este producto me da yo pueda depender de ellos sin ningún problema. Luego, la usabilidad tiene que ver con aspectos de la disciplina UX/UI, donde el usuario podrá lograr sus objetivos productivamente usando este producto, estando a gusto con su uso. Tiene que ser productivo, no es lo mismo que la utilidad. Por ejemplo, si tarda 12 horas en realizar una operación donde el anterior tarda 4 horas no es productivo para mí y no mejora la calidad de vida al usuario.

Existe una línea en la pirámide que separa la usabilidad de la conveniencia. La mayoría de los productos no se ubican por encima de esta línea, debido a que resulta complejo que un software sea conveniente para el desempeño de tareas de ciertas personas (es decir, que el producto nos haga sentir que lo necesitamos porque nos produce un cambio, una facilidad para cumplir con un objetivo), y que estas no sean “esclavos” del software, limitándose a cumplir con las características de funcional, confiable y/o usable. Placentero y significativo para todo el mundo, algo que nos cambie la forma de usar algo o de vivir.

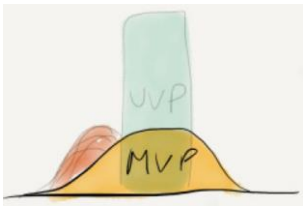
MVPs, MVFs, MMFs



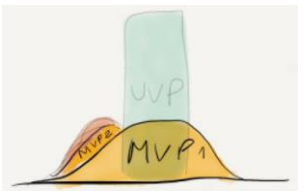
Una forma para poder entender qué significa cada 1 de estos conceptos es utilizar el modelo de dinosaurio.
El primer paso se encuentra en entender que un producto nuevo tiene una **hipótesis** de **valor único**. Este es el área donde el producto o el servicio va a ser único.

El siguiente paso es crear el **Minimum Viable Product (MVP)**, lo que significa producto mínimo viable para **poder testear esta hipótesis**. Esto se centra en la propuesta de valor única, pero normalmente también proporciona un poco de funciones “Table stakes” solo para asegurarse de que sea viable como producto.

Este MVP también es una hipótesis por lo que podría ser lo suficientemente bueno para **encontrar un mercado** o no. Se muestra el caso en el que cada cliente potencial con el que interactúas te dice: “Esto es genial pero para poder usarlo necesito X” y X es diferente para cada cliente o usuario. Esto muestra que aún no se encuentra un mercado para el producto.
Si por el contrario ves cada vez más respuestas apuntanto al mismo X entonces tiene sentido revisar la hipótesis de Cliente/Problema/Solución.
Básicamente estás ejecutando un pívot, se está construyendo un **MVP2** centrado en la **nueva hipótesis** basada en el aprendizaje reciente de desarrollo de clientes generado por el anterior MVP.

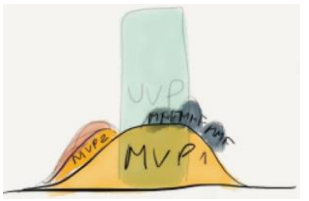


Supongamos que el MVP2 es exitoso y se está viendo una tracción real de los primeros usuarios. Se desea aumentar el crecimiento y se busca una penetración más profunda en los primeros usuarios además de atraer nuevos clientes, algunos de ellos más allá de la multitud de usuarios pioneros. En función del feedback recopilado y la investigación de la gestión de productos, se tiene un par de **áreas** que potencialmente pueden traer este **crecimiento**. Algunos de ellos, por cierto, amplían la propuesta de valor única y algunos hacen que el producto actual sea más robusto.



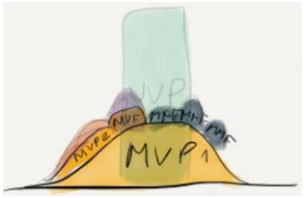
En el caso de areas con una fuerte indicación de valor se puede directamente definir un **Minimum Marketeable Features (MMF)**, es decir, características mínimas comercializables, para poder encontrar la pieza mínima que pueda empezar a traer el crecimiento. El **objetivo** del **MMF** es **aportar valor**. Se supone una alta certeza de que existe valor en esta área y que sabemos cuál debe ser el producto para proporcionar este valor. La razón para dividir una característica grande en MMF más pequeños es principalmente el tiempo de comercialización (Time to market) y la capacidad de aportar valor en muchas áreas. Una indicación de que se está **trabajando en MMFs** es que cuando al liberarse uno se **siente cómodo trabajando en el próximo MMF** en esa área.

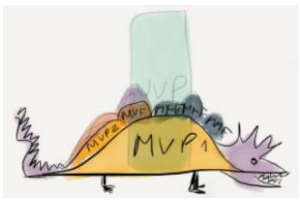
Si esto no es así y se desea esperar para ver si el primer MF no tuvo un buen impacto entonces se está de vuelta en la tierra de la hipótesis. Ahora la hipótesis se centra en una característica en lugar del producto y se tiene un alto potencial pero también un alta incertidumbre.
La forma de afrontarlo es crear una **función pionera** la forma de afrontarlo es crear **Minimum Viable Feature (MVF)**. La característica mínima que aún puede ser viable para uso real y aprendizaje de los usuarios reales.



Si la MVF resulta exitosa puede desarrollar más MMF en esa área para tomar ventaja (si esto tiene sentido). Si no es así, se puede cambiar otro enfoque hacia esa área de características o en algún momento buscar una ruta de crecimiento alternativa. Esencialmente, el **MVF es una versión mini del MVP**.
De esta forma se obtiene el modelo completo. Esencialmente el punto es que se puede **construir un producto el mercado desconocidos al intentar varios MVPs**.

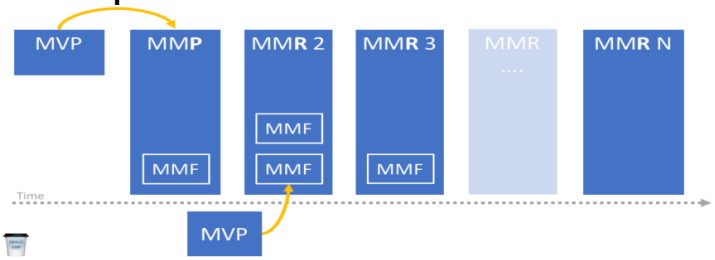
Cuando se logra ajustar el producto en el mercado de productos se combinan MMF y MVF según el nivel de incertidumbre del negocio y los requerimientos en las áreas en las que se está enfocando.
Si bien MVPs/MMFs/MVFs son atómicos desde una perspectiva empresarial, es decir, no se puede implementar y aprender de algo más pequeño, pueden ser bastante grandes desde la perspectiva de la implementación.
El dinosaurio se obtiene cortando cada una de esas piezas en pequeñas porciones destinadas a reducir el riesgo de la ejecución y de las tecnologías, estas piezas normalmente se denominan user stories, y pueden tener un valor comercial tangible o no. Para otros pueden ser importante proporcionar una temprana retroalimentación de decisiones a lo largo del camino.





Productos Mínimos para la Gestión de Productos
- **MVP – Minimal Viable Product – Producto Mínimo Viable**
- **MVF – Minimal Viable Feature – Característica Mínima Viable**
- **MRF – Minimal Release Feature – Característica Mínima de Release**
- **MMF – Minimal Marketable Feature – Característica Mínima Comercializable**

Relación entre MVP, MMF, MMP, MMR
- **MVP**
  - Versión de un **nuevo producto** creado con el **menor esfuerzo posible**
  - Dirigido a un **subconjunto de clientes potenciales**
  - Utilizado para obtener **aprendizaje validado**
  - Más cercano a los **prototipos que a una version real funcionando de un producto**.
- **MMF**
  - Es la **pieza más pequeña de funcionalidad** que puede ser liberada
  - Tiene valor tanto para la organización como para los usuarios.
  - Es parte de un MMR o MMP.



- **MMP**
  - Primer release de un MMR dirigido a **primeros usuarios** (early adopters),
  - Focalizado en características clave que satisfarán a este grupo clave.
- **MMR**
  - Release de un producto que tiene el conjunto de características más pequeño posible.
  - El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisfice sus necesidades actuales.
  - **MMP = MMR1**

MVP – Minimum Viable Product
Es un concepto de Lean Startup qué enfatiza el impacto del aprendizaje en el desarrollo de nuevos productos. Una premisa clave detrás de la idea del MVP es que usted produce un producto real que puede ofrecer a los clientes y observar su comportamiento real con el producto o servicio.
El MVP es la versión de un nuevo producto que permite a un equipo recopilar la cantidad máxima de aprendizaje validado sobre clientes con el menor esfuerzo. Este aprendizaje validado viene en forma de si sus clientes realmente comprarán su producto.
Ver lo que la gente realmente hace con respecto a un producto es mucho más confiable que preguntarle a la gente que harían.

El MVP tiene el valor suficiente para que las personas estén dispuestas a usarlo o comprarlo inicialmente. Además demuestra suficiente beneficio futuro para retener a los primeros usuarios y proporciona un ciclo de retroalimentación para guiar el desarrollo futuro.

MVF – Minimum Viable Feature
El MVF es una versión mini del MVP y representa una característica a pequeña escala que se puede construir e implementar rápidamente, utilizando recursos mínimos, para una población objetivo para probar la utilidad y adopción de la característica.

Un MVF de proporcionar un valor claro a los usuarios y ser fácil de usar. El MVF requiere recursos mínimos, los estándares de calidad de la industria y la producción deben guiar el diseño y la confiabilidad.
El grupo de usuarios para un MVF, son los primeros en adoptar, son los clientes leales que han compartido conocimientos anteriormente o los miembros de una junta asesora de clientes. Son usuarios flexibles y tolerantes.
Los resultados ayudarán a tomar decisiones estratégicas sobre el producto.

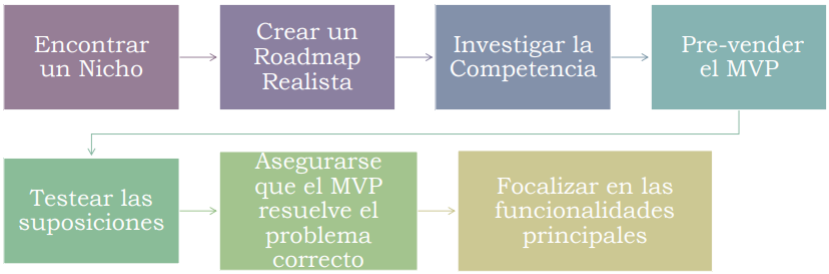
MVP – MMF – MMP: Errores Comunes

- Confundir a un MVP, que **se enfoca en el aprendizaje**, con un MMF o con un MMP, que ambos **se enfocan en ganar**.
- El riesgo de esto es entregar algo sin considerar si es lo correcto qué satisface las necesidades del cliente.
- Se enfatiza la parte **mínima** del MVP con exclusión de la parte **viable**. El producto entregado no es de calidad suficiente para proporcionar una evaluación precisa de si los clientes utilizarán el producto.
- Entregar lo que se consideran un MVP, y luego no hacer más cambios a ese producto, independientemente de los comentarios que reciban al respecto.

Valor vs Desperdicio

Lean Thinking define la creación de valor como proveer beneficios a los clientes, cualquier otra cosa es desperdicio.
La productividad de un Startup **no puede medirse en términos de cuánto se construye cada día**, por el contrario **se debe medir en términos de averiguar la cosa correcta a construir cada día**.

Preparar un MVP



SOFTWARE CONFIGURATION MANAGEMENT (SCM)

Cuando pensamos en software pensamos en un conjunto de programas, procedimientos, reglas, datos y la documentación que lo acompaña. Pensamos en una idea del software como información o conocimiento empaquetado a distintos niveles de abstracción, donde el nivel más bajo es el código.
El software es **información**:
- Información estructurada con propiedades lógicas y funcionales.
- Información creada y mantenida en varias formas y representaciones.
- Información confeccionada para ser procesada por computadora en su estado más desarrollado.

Los sistemas de software siempre cambian durante su desarrollo y uso. Conforme se hacen cambios al software, se crean nuevas versiones del sistema. Los sistemas pueden considerarse como un conjunto de versiones donde cada una de ellas debe mantenerse y gestionarse. Esto es necesario para no perder la trazabilidad de los cambios que se incorporan a cada versión.

Estos **cambios en el software** pueden tener distintos orígenes:
- Cambios del negocio y nuevos requerimientos
- Soporte de cambios de productos asociados
- Reorganización de las prioridades de la empresa por crecimiento
- Cambios en el presupuesto
- Defectos encontrados a corregir
- Oportunidades de mejor

Gestión de Configuración (SCM)

La administración o gestión de configuraciones es una disciplina de soporte, transversal a todo el proyecto con aplicación en diferentes disciplinas, y tiene el propósito de **mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida**.

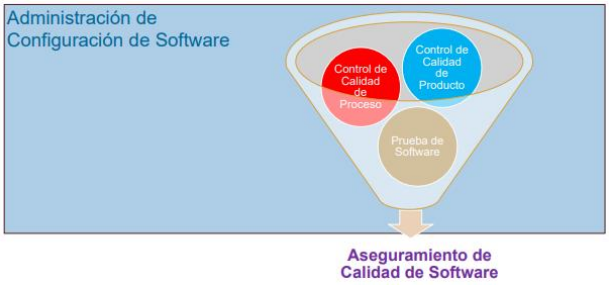
Involucra para la configuración:
- Identificarla en un momento dado
- Controlar sistemáticamente sus cambios
- Mantener su integridad y origen

La SCM es una disciplina que aplica **dirección y monitoreo** administrativo y técnico a:
- Identificar y documentar las características funcionales y técnicas de los ítems de configuración.
- Controlar los cambios de esas características.
- Registrar y reportar los cambios y su estado de implementación.
- Verificar correspondencia con los requerimientos.

Integridad del Producto
El software es un blanco móvil porque al ser intangible y fácilmente modificable la administración de configuración ayuda a que el producto permanezca inalterable en los términos que fue creado a lo largo del ciclo de vida. Se considera que un producto tiene integridad si:
- **Satisface las expectativas del usuario**. Considera los requerimientos funcionales
- **Puede ser fácilmente rastreado durante su ciclo de vida**. Puedo saber cómo llego a este punto cada pieza de software, es decir saber a qué requerimiento está asociado y saber dónde se analizó, donde se diseñó, donde se testeo, etc.
- **Satisface criterios de performance**. Satisface requerimientos no funcionales (performance, etc.)
- **Cumple expectativas de costo**. Para que esto sea posible es necesario planificar, por lo cual se evalúa la relación costo beneficio.

Problemas en el Manejo de Componentes
- Pérdida de un componente
- Pérdida de cambios (el componente que tengo no es el último)
- Sincronía fuente - objeto – ejecutable
- Regresión de fallas
- Doble mantenimiento
- Superposición de cambios
- Cambios no validados

¿Cuál es el problema que se está tratando de solucionar?



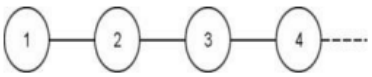
El software es muy fácil de modificar, en el sentido de que uno puede entrar al repositorio y con un solo click eliminar meses de trabajo. Entonces lo que buscamos con la gestión de configuración del software es tratar de que estas cosas no pasen, buscando que el producto de software sea íntegro y si fuera el caso de que se van a realizar cambios, podamos tener un control de qué cosas cambiaron en un determinado momento. Es por esta razón que se asocia el concepto de software con el concepto de versión, ya que cada ítem de configuración debe tener su versión. Y que todas las personas se enteren de los cambios.

Conceptos Clave para la Gestión de Configuración de Software

- **Ítem de Configuración de Software (SCI).** Se llama ítem de configuración (IC) a todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución a lo largo del ciclo de vida (ya sea del producto o proyecto, dependiendo del artefacto).
  Es decir, es cualquier aspecto asociado al producto de software (requerimientos, diseño, código, datos de prueba, documentos, etc.) que es necesario mantener. Son todos aquellos elementos que componen toda la información producida como parte del proceso de ingeniería de software, como ser programas de computadora (código fuente y ejecutables), documentos que describen los programas (documentos técnicos o de usuario), datos (de programa o externos).
  Los ítems, dependiendo de su naturaleza, pueden durar lo que dure el ciclo de vida del proyecto, producto o sprint.

Algunos ejemplos: Código Ejecutable, Documentos de Diseño, Plan de CM, Propuesta de Cambio, Versión, Planes de Iteración, Código Fuente, Requerimientos, Prototipo de Interfaz, etc.

- **Versión.** Una versión se define, desde el punto de vista de la evolución, como la forma particular de un artefacto en un instante o contexto dado. Es una instancia de un ítem de configuración que difiere de otras instancias de este ítem.
  La versión es un punto particular en el tiempo de ese ítem de configuración (es un estado). Las versiones se identifican unívocamente.



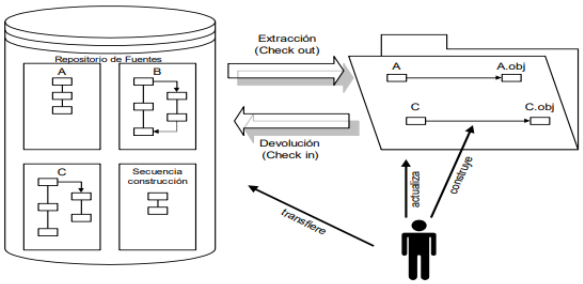
Evolución lineal de un ítem de configuración

  El **control de versiones** se refiere a la evolución de un único ítem de configuración (IC), o de cada IC por separado.
  La evolución puede representarse gráficamente en forma de grafo.
  La gestión de configuración permite a un usuario especificar configuraciones alternativas del sistema de software mediante la selección de las versiones adecuadas; esto se puede gestionar asociando atributos a cada versión (que pueden ser datos sencillos como un nro. de versión asociado a cada objeto. Cada versión de sw es una colección de elementos de configuración (ECS), como ser código fuente, documentos, datos).

- **Variante.** Una variante es una versión de un ítem de configuración (o de la configuración) que evoluciona por separado. Las variantes representan configuraciones alternativas. Un producto de software puede adoptar distintas formas (configuraciones) dependiendo del lugar donde se instale.
  Por ejemplo, dependiendo de la plataforma (máquina + S.O.) que la soporta, o de las funciones opcionales que haya de realizar o no.
  Las variantes de un ítem de configuración permiten satisfacer requerimientos en conflicto, como por ejemplo la compatibilidad del producto con diferente hardware o sistemas operativos. Si la diferencia que existe entre las instancias de un mismo ítem de configuración es muy pequeña, no se denomina versión, sino que se la conoce como variante.

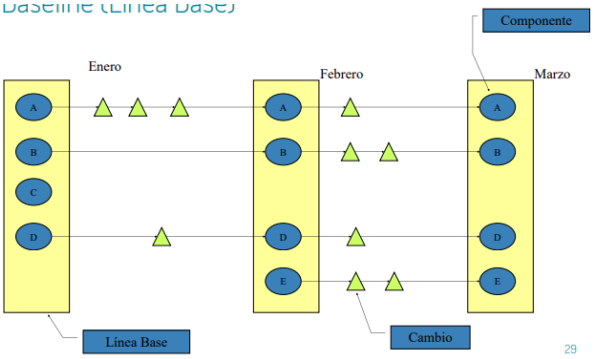
- **Configuración del Software.** Un conjunto de ítems de configuración con su correspondiente versión especifica en un momento determinado. Define una foto de los ítems de configuración en un momento de tiempo determinado.
  Una configuración es el conjunto de todos los componentes fuentes que son compilados, sus documentos y la información de la estructura que definen una versión del producto a entregar. La configuración de un software es la sumatoria de todos los ítems de configuración que tiene en un momento determinado, equivale a una instantánea o una foto de todos los ítems de configuración con su versión en un momento del tiempo.

- **Repositorio.** Es un contenedor de ítems de configuración, se encarga de mantener la historia de cada ítem con sus atributos y relaciones, además es usado para hacer evaluaciones de impacto de los cambios propuestos. Puede ser una o varias bases de datos.
  Tiene una estructura para mantener el orden y la integridad. El que tenga una estructura ayuda a la seguridad, los controles de acceso, las políticas de Backup y todo aquello que se aplica sobre un repositorio.



  - ✓ *Repositorio Centralizado:* está caracterizado porque un servidor contiene todos los archivos con sus versiones. Los administradores tienen un mayor control sobre el repositorio. Tiene como desventaja qué si falla el servidor, todo lo positivo se cae.
  - ✓ *Repositorio Descentralizado:* está caracterizado porque cada cliente tiene una copia exactamente igual del repositorio completo. Tiene como ventaja que se resuelve el problema de los servidores centralizados, debido a que si el servidor falla sólo es cuestión de “copiar y pegar”, además posibilita otros workflows no disponibles en el modelo centralizado. La desventaja que podemos mencionar es que es más complicado llevar un control sobre el mismo.

➢ **Línea Base.** Es un conjunto de ítems de configuración que han sido construidos y revisados formalmente, de manera que pueden ser tomados como referencia para demostrar que se ha alcanzado cierto nivel de madurez en ellos, y que sirve como base para desarrollos posteriores. Este conjunto de ítems debe tener una referencia única, y esto se hace a través de "tags".



Esto es lo mismo que decir que es una configuración de software que ha sido formalmente revisada y aprobada, que sirve como base para desarrollos. Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.

Se acuerdan parámetros para determinar qué se considera o qué debe cumplir los ítems de configuración para considerarse como línea base. Si se desea cambiar una línea base, existe un protocolo formal de control de cambios, dirigido por el Comité de Control de Cambios, que permite definir el procedimiento a seguir para manejar peticiones de cambios, y en caso de aceptarse, acordar nuevamente los parámetros y comunicar los cambios a todo el equipo, para que tomen la nueva línea base como modelo a seguir.

¿Para qué sirve? Fundamentalmente es para tener un punto de referencia, pero también sirve para hacer Rollback o saber qué se pone en producción. Nos permite saber cuál era la última situación estable en un momento de tiempo y cómo se fue evolucionando. Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.

Dos tipos de línea base:

- ✓ **Operacionales**: contiene una versión de producto cuyo código es ejecutable, han pasado por un control de calidad definido previamente. La primera línea base operacional corresponde con la primera Release. Es la línea base de productos que han pasado por un control de calidad definido previamente.
- ✓ **De especificación o documentación**: son las primeras línea base, dado que no cuentan con código. Podría contener el documento de especificación de requerimiento. Son de requerimientos, diseño.

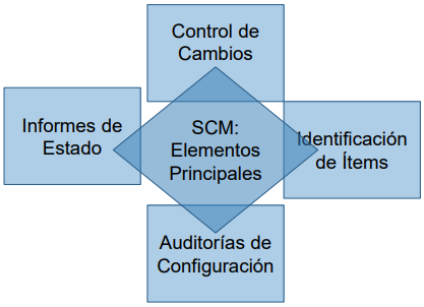
➢ **Ramas.** Es un conjunto de ítems de configuración con sus correspondientes versiones, que permiten bifurcar el desarrollo de un software, por varios motivos: *experimentación*, *resolución de errores de desarrollo*; *desarrollar un mismo software para diferentes plataformas*.

Integración de Ramas (Merge): Se da cuando se integran dos ramas, para fusionar la configuración de los ítems de configuración con sus correspondientes versiones, en cada una de ellas. Una buena práctica es mantener la rama principal como la versión estable, y fusionar los cambios hacia ella. En caso de no integrarse a la Main, las ramas deberían descartarse.

Actividades Fundamentales de la SCM

Estas actividades/elementos son las cosas que contienen las secciones de un plan de gestión de configuración. Recordemos que la gestión de configuración también se debe planificar.

Todas las actividades se realizan en ambas metodologías, menos las auditorías, ya que el concepto de ser auditado y controlado por alguien externo, no es compatible con una metodología ágil pura.



❖ **Identificación de Ítems de Configuración.** Esto implica una identificación unívoca para cada ítem, donde en el equipo se definirán convenciones y reglas de nombrado y versionado para todos ellos. También, se debe definir la estructura del repositorio, y la ubicación de los ítems de configuración dentro de esa estructura.

Además, implica una definición cuidadosa de los componentes de la línea base.

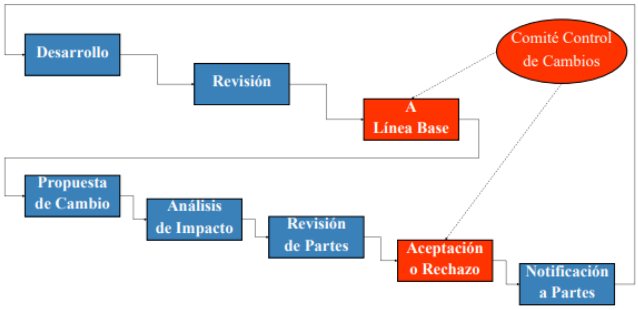
Esta identificación y documentación proveen un camino que une todas las etapas del ciclo de vida del software, lo que permite a los desarrolladores controlar y velar por la integridad del producto, como así también a los clientes evaluar esa integridad.

- **Ítems de Producto:** tienen el ciclo de vida más largo, y se mantienen mientras el producto exista. Ejemplo: documento de arquitectura, casos de uso, código, manual de usuario y de parametrización, casos de prueba, una ERS, los casos de prueba, la base de datos.

- **Ítems de Proyecto:** el plan de proyecto, el listado de defectos encontrados, tienen un ciclo de vida de proyecto. Un plan de iteración, un Burn-Down Chart, se mantiene durante una iteración. La duración impacta también en el esquema de nombrado, plan de proyecto. Ejemplo: Los ítems de configuración a nivel de proyecto, el plan de proyecto y el cronograma.
- **Ítems de Iteración:** Conocer el ciclo de vida de un ítem permite establecer la nomenclatura de este. Ejemplo: planes de iteración, cronogramas de iteración, reporte de defectos.

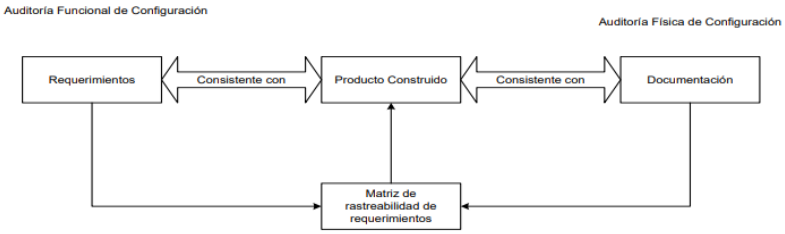
Luego de identificar los ítems se debe armar la estructura del repositorio con nombres representativos. Luego se vinculan los ítems de configuración con el lugar físico del repositorio donde se van a almacenar.

❖ **Control de Cambios.** Una vez definida la línea base, no es posible cambiarla sin antes pasar por un proceso formal de control de cambios. Es decir que el control se hace sobre los ítems de configuración que pertenecen a la **línea base**, ya que todos los trabajadores del software tienen a dichos ítems como referencia.
El control de cambios es un procedimiento formal que involucra diferentes actores y una evaluación del **impacto** del cambio.



Comité de Control de Cambios: esta formado por representantes de todas las áreas involucradas en el desarrollo (análisis, diseño, implementación, testing, otros). Se reúne para autorizar la creación y cambios sobre la línea base. Forman parte de dicho comité los interesados en evaluar y en enterarse del cambio, decidiendo si lo aceptan o no.
Se recibe una **propuesta de cambio** sobre una línea base determinada, no sobre un ítem. El comité de control de cambios realiza un **análisis de impacto** del cambio para evaluar el esfuerzo técnico, el impacto en la gestión de los recursos, los efectos secundarios y el impacto global sobre la funcionalidad y la arquitectura del producto. En caso de que se autorice la propuesta de cambio, se genera una **orden de cambio** que define lo que se va a realizar, las restricciones a tener en cuenta y los criterios para revisar y auditar. Luego de realizado el cambio, el comité vuelve a intervenir para **aprobar la modificación** de la línea base y marcarla como línea base nuevamente, es decir que realiza una revisión de partes. Finalmente se **notifica a los interesados** los cambios realizados sobre la línea base.

❖ **Auditorias de Configuración.** Son controles autorizados a realizar por el equipo de desarrollo en un momento determinado, donde un auditor externo al equipo (independiente y objetivo) analiza las líneas base, las cuales permiten "congelar" y analizar en un momento determinado cuál es el estado del software, y si se están cumpliendo todas las pautas que plantea esta disciplina de SCM.



La auditoría de configuración se hace mientras el producto se está construyendo y su objetivo es que el producto tenga calidad e integridad. Se entiende que lo más barato es prevenir.

- ✓ **Auditoría física de configuración (PCA).** Asegura que lo que está indicado para cada ICS en la línea base o en la actualización se ha alcanzado realmente.
- ✓ **Auditoría funcional de configuración (FCA).** Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de requerimientos.

Sirve a dos procesos básicos: la validación y la verificación

- **Validación:** el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto.
- **Verificación:** asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea base). Todas la funciones son llevadas a cabo con éxito y los test cases tengan status "ok" o bien consten como "problemas reportados" en la nota de release.

En *metodologías ágiles*, esta es la única actividad de la disciplina SCM que **no** se **soporta por la filosofía**. En estos casos se trabaja con integración continua (CI - Continuous Integration), un servidor donde se testea automáticamente el código y luego se integra en producción continuamente.

- **Registro e Informe de Estado.** Provee un mecanismo para mantener un registro de cómo evoluciona el sistema, y dónde está ubicado el software, comparado con lo que está publicado en la línea base. Esto permite mantener a todo el equipo informado sobre la última línea base, y que se trabaje en base a su última versión, y no sobre una versión obsoleta.
  Maneja mucha información y salidas por lo que se suele implementar dentro de procesos automáticos, e incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida.
  Permite responde a preguntas como:
  a. ¿Cuál es el estado del ítem?
  b. ¿La propuesta de cambio fue aceptada o rechazada por el comité?
  c. ¿Qué versión de ítem implementa un cambio de una propuesta de cambio aceptada?
  d. ¿Cuál es la diferencia entre dos versiones de un mismo ítem?

Plan de Gestión de Configuración
Este plan se debe confeccionar al inicio de un proyecto, y existen diferentes estándares donde se expresa cómo planificar:
- Reglas de nombrado de los CI
- Herramientas a utilizar para SCM
- Roles e integrantes del Comité
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de Auditoría

Consideraciones:
Debe hacerse **tempranamente**; se deben **definir los documentos** que se administrarán; **todos los productos** del proceso **deben administrarse**.

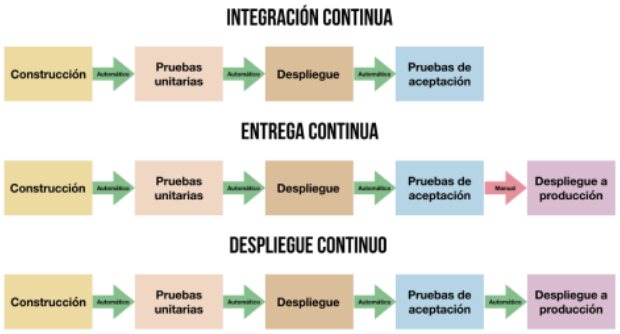
Gestión de Configuración de Software en AMBIENTES AGILES
En las metodologías ágiles la gestión de configuración cambia el enfoque, ya que la misma es de utilidad para los miembros del equipo de desarrollo y no viceversa.
En orden con la caracterización de los equipos ágiles, decimos que la gestión de configuración posibilita el seguimiento y la coordinación del desarrollo en lugar de controlar a los desarrolladores.
Los equipos ágiles son autoorganizados, por lo que las Auditorías de configuración no son una actividad propia de la gestión de configuración en los ambientes ágiles. Todos los procesos definidos establecidos en la gestión de configuración del enfoque tradicional son relativizados en agile. Por ejemplo, no existe un comité para el proceso forma de control de cambios.
Podemos decir que va de la mano con el manifiesto ágil porque recordemos que se buscaba estar siempre preparados para el cambio y justamente, la gestión de configuración busca responder a los cambios y mantener la integridad del producto.
La diferencia es que el manifiesto ágil se enfoca en los proyectos, mientras que la gestión de configuración de software se enfoca en el producto. Es decir, trasciende el proyecto.
Mientras el manifiesto ágil apunta a cómo trabajar en el contexto del proyecto de desarrollo y la gestión de configuración es una práctica específica del producto que garantiza la calidad del producto.

SCM en Agile
- Sirve a los practicantes (equipo de desarrollo) y no viceversa.
- Hace seguimiento y coordina el desarrollo en lugar de controlar a los desarrolladores.
- Responde a los cambios en lugar de tratar de evitarlos.
- Esforzarse por ser transparente y "sin fricción", automatizando tanto como sea posible.
- Coordinación y automatización frecuente y rápida.
- Eliminar el desperdicio - no agregar nada más que valor.
- Documentación Lean y Trazabilidad.
- Feedback continuo y visible sobre calidad, estabilidad e integridad.

Algunos Tips
- ✓ Es responsabilidad de todo el equipo.
- ✓ Automatizar lo más posible.
- ✓ Educar al equipo.
- ✓ Tareas de SCM embebidas en las demás tareas requeridas para alcanzar el objetivo del Sprint.

EVOLUCIÓN DE LA GESTIÓN DE CONFIGURACIÓN DE SOFTWARE

Integración, Entrega y Despliegue Continuo (CI, CD, CD)

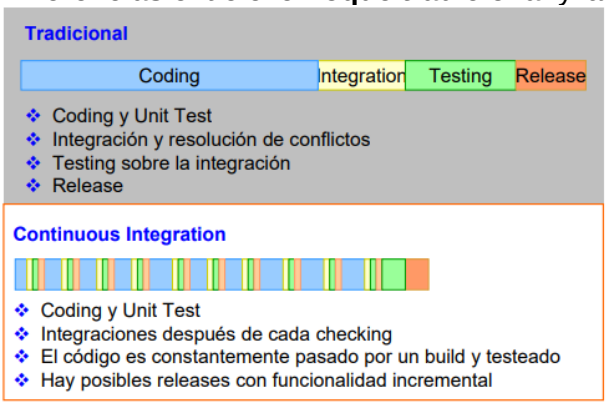


- **Continuous Integration (CI).** Continuous Integration es una práctica de desarrollo donde los miembros de un equipo integran su trabajo frecuentemente, generalmente una persona integra por lo menos diariamente.
  Cada integración es verificada por una herramienta de build automático (incluido un test) para detectar errores de integración tan pronto como sea posible.

  Esto funcionara si se cumple que:
  - Mantenga un repositorio para el código donde todos los del equipo tengan acceso a la última versión y a las previas.
  - Automatizar el proceso de build. Cualquiera puede lanzar un build.
  - Automatizar el testing (smoke test). Cualquiera puede lanzar la suite de test cases.
  - Todos tienen acceso al ejecutable que se cree que es el mejor ejecutable.
  - Se le dedica un tiempo de seteo y tiempo de mantenimiento.

Se busca un **build exitoso**, lo que no significa compilar todos los días, sino que implica una serie de pasos: las **fuentes** deben estar en el **repositorio** (check operation), **cada fuente se compila**, los **resultados** de los object files son **linkeados** y se los deja **listo para ejecucion**, se **arranca el sistema** y se corre una **suite de test cases**. Si todo esto se ejecuta sin errores ni intervención humana, entonces consideramos que hay un “build exitoso”.
Mientras más exhaustiva sea la suite de test cases mas valor genera la CI.
Es mejor tener mayor frecuencia en las integraciones. Con equipos que nunca lo trabajaron es una contradicción para su experiencia previa. Una clave para esto es la automatización

Diferencias entre el **enfoque tradicional** y la **CI**



- **Continuous Delivery (CD).** Es una disciplina de desarrollo de software en la que el software se construye de tal manera que puede ser liberado en producción en cualquier momento. Suma la automatización de las pruebas de aceptación y entonces el producto ya está listo para desplegarlo a producción.
  No implica necesariamente que se libere cada vez que hay un cambio, sólo ocurre si el responsable de negocio, el Product Owner de turno, decide pasar a producción. Es decir que hay un componente «humano» a la hora de tomar la decisión. Pero, en cualquier caso, la versión está lista de inmediato. Esto implica, entre otros, que se prioriza que la versión esté en un estado en el que pueda ser puesta en producción.
  La integración continua es prerrequisito de la entrega continua. Con esto se refiere a que los artefactos producidos en el servidor de integración continua son puestos en producción con solo un click. Hay que asegurarse de tener un despliegue automatizado, para que cada puesta en producción no lleve demasiado tiempo, lo que hará que las puestas puedan llegar a hacerse más seguidas, generando que lo que se despliegue no sean demasiados cambios y el riesgo de que algo se rompa disminuya. El software debe estar siempre en un estado de “entregable”, es decir, el software buildea, el código se compila y los tests pasan.

- **Continuous Deployment (CD).** Consiste en poner en el ambiente de producción del usuario final el producto. A diferencia de la entrega continua, en el despliegue continuo no existe la intervención humana para desplegar el producto en producción. Para esto, se utilizan pipelines que contienen una serie de pasos que deben ejecutarse en un orden determinado para que la instalación sea satisfactoria.
  El propósito de las estrategias es que sea transparente para el usuario que pusiste en producción una nueva versión del producto.
  Se recomienda hacer el despliegue a poco tiempo de haber trabajado con los cambios en el código, pues un error en producción un día después de haberlo hecho significa que todavía nos acordamos de lo que hicimos y será más fácil de resolver.

  Estrategias de Despliegue Continuo:

  - ✓ *Blue-Green Deployment:* es un modelo de lanzamiento de aplicaciones que transfiere gradualmente el tráfico de usuarios de una versión anterior de una aplicación o microservicio a una nueva versión casi idéntica, las cuales se ejecutan en producción.
    La versión anterior puede denominarse entorno azul, mientras que la nueva versión puede denominarse entorno verde. Una vez que el tráfico de producción se transfiere por completo de azul a verde, el azul puede quedar en espera en caso de reversión o retirada de producción y se actualiza para convertirse en la plantilla sobre la que se realiza la próxima actualización.

  - ✓ *Canary Deployment:* es la práctica de realizar lanzamientos por etapas. Primero implementamos una actualización de software para una pequeña parte de los usuarios, para que puedan probarla y proporcionar comentarios. Una vez aceptado el cambio, la actualización se extiende al resto de usuarios.
    Esta estrategia nos muestra cómo los usuarios interactúan con los cambios de la aplicación en el mundo real. Al igual que en Blue-Green Deployment, la estrategia Canary ofrece actualizaciones sin tiempo de inactividad y reversiones sencillas. A diferencia de Blue-Green, las implementaciones Canary son más fluidas y las fallas tienen un impacto limitado.

  - ✓ *A/B Testing:* hace referencia al proceso de experimentación aleatoria según el cual dos o más versiones de una misma variable (una página web, un elemento concreto de la página, etc.) se presentan a distintos segmentos de visitantes de un sitio web para determinar cuál de ellas reporta más beneficios a la empresa.

SCRUM

Scrum es un **marco de trabajo** ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.

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

1. Un propietario del producto (Product Owner) ordena el trabajo de un problema complejo en un Product Backlog.
2. El equipo de Scrum convierte una selección del trabajo en un Incremento de valor durante un Sprint.
3. El equipo de Scrum y sus partes interesadas (stakeholders) inspeccionan los resultados y realizan los ajustes necesarios para el próximo Sprint.
4. Repetir.

Scrum se basa en el empirismo y el pensamiento Lean. El empirismo afirma que el conocimiento proviene de la experiencia y la toma de decisiones basadas en lo que se observa. El pensamiento Lean reduce los desperdicios y se centra en lo esencial.

Scrum emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo.

Scrum involucra a grupos de personas que colectivamente tienen todas las habilidades y experiencia para hacer el trabajo y compartir o adquirir tales habilidades según sea necesario.

Scrum combina cuatro eventos formales para la inspección y adaptación dentro de un evento contenedor, el Sprint. Estos eventos funcionan porque implementan los pilares empíricos de Scrum: **transparencia**, **inspección** y **adaptación**.

El Equipo SCRUM

La unidad fundamental de Scrum es un pequeño equipo de personas, un equipo Scrum. El equipo Scrum consta de un **Scrum Master**, un propietario de producto (**Product Owner**) y **desarrolladores**. Dentro de un equipo de Scrum, no hay sub-equipos ni jerarquías. Es una unidad cohesionada de profesionales enfocada en un objetivo a la vez, el objetivo del producto.

Los equipos de Scrum son **multifuncionales**, lo que significa que los miembros tienen todas las habilidades necesarias para crear valor en cada Sprint. También son **autogestionados**, lo que significa que internamente deciden quién hace qué, cuándo y cómo.

El equipo de Scrum es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar un trabajo significativo dentro de un Sprint, por lo general 10 o menos personas. En general, hemos descubierto que los equipos más pequeños se comunican mejor y son más productivos. Si los equipos de Scrum se vuelven demasiado grandes, se debe considerar la posibilidad de reorganizarse en varios equipos Scrum cohesionados, cada uno centrado en el mismo producto. Por lo tanto, deben compartir el mismo objetivo de producto, trabajo pendiente del producto (Product Backlog) y propietario del producto (Product Owner).

El equipo Scrum es responsable de todas las actividades relacionadas con los productos, desde la colaboración, verificación, mantenimiento, operación, experimentación, investigación y desarrollo, y cualquier otra cosa que pueda ser necesaria. Están estructurados y empoderados por la organización para gestionar su propio trabajo.

Trabajar en Sprints a un ritmo sostenible mejora el enfoque y la consistencia del equipo de Scrum. Todo el equipo de Scrum es responsable de crear un incremento valioso y útil en cada Sprint.

- **Desarrolladores.** Los desarrolladores son las personas del equipo Scrum que se comprometen a crear cualquier aspecto de un Incremento útil (funcional) en cada Sprint. Las habilidades específicas que necesitan los desarrolladores son a menudo amplias y variarán con el dominio del trabajo. Sin embargo, los desarrolladores siempre son responsables de:
  - Crear un **plan** para el Sprint, el **Sprint Backlog**;
  - Inculcar la calidad adhiriéndose a una **definición de Hecho**;
  - **Adaptar** su plan cada día hacia el **Objetivo Sprint**;
  - **Responsabilizarse** mutuamente como profesionales.

- **Product Owner.** El Propietario del Producto es responsable de maximizar el valor del producto resultante del trabajo del equipo de Scrum. La forma en que esto se hace esto puede variar ampliamente entre organizaciones, equipos Scrum e individuos. El Propietario del Producto también es responsable de la gestión eficaz de la pila del producto (Product Backlog), que incluye:
  - Desarrollar y **comunicar** explícitamente el **Objetivo del Producto**;
  - Creación y **comunicación clara** de **elementos de trabajo pendiente** del producto;
  - Pedido de **artículos de trabajo pendiente** del producto;
  - Asegurarse de que el **trabajo pendiente** del producto sea **transparente**, **visible** y **comprendido**.

  El PO puede hacer el trabajo anterior o puede delegar la responsabilidad a otros. En cualquier caso, el propietario del producto sigue siendo responsable.

  Para que los POs tengan éxito, toda la organización debe respetar sus decisiones. Estas decisiones son visibles en el contenido y el orden del trabajo pendiente del producto, y a través del Incremento inspeccionable en la revisión de Sprint.

  El PO es una persona, no un comité. El PO puede representar las necesidades de muchas partes interesadas en el trabajo pendiente del producto. Aquellos que deseen cambiar el trabajo pendiente del producto pueden hacerlo tratando de negociar con criterio con el PO.

- **Scrum Master.** El Scrum Master es responsable de establecer Scrum tal como se define en la Guía de Scrum. Lo consigue ayudando a todos a comprender la teoría y la práctica de Scrum, tanto dentro del Equipo como en toda la organización. El Scrum Master es responsable de la efectividad del Scrum Team. Lo logra al permitir que el equipo Scrum mejore sus prácticas, dentro del marco de Scrum.

  Los Scrum Masters son verdaderos líderes que sirven al equipo Scrum y a toda la organización.

  El Scrum Master sirve al equipo de Scrum de varias maneras, incluyendo:
  - **Capacitar** a los miembros del equipo en autogestión y multifuncionalidad;
  - Ayudar al equipo de Scrum a **centrarse** en la creación de **incrementos de alto valor** que cumplan con la **definición de hecho**;

- Promover la **eliminación de los impedimentos** para el progreso del equipo Scrum;
- Asegurar de que todos los **eventos de Scrum se lleven a cabo**, sean positivos, productivos y que se respete el tiempo establecido (time-box) para cada uno de ellos.

El Scrum Master sirve al Product Owner de varias maneras, incluyendo:
- Ayudar a **encontrar técnicas** para una **definición** eficaz de los **objetivos del producto** y la gestión de los retrasos en el producto;
- Ayudar al equipo de Scrum a **comprender** la **necesidad** de elementos de **trabajo pendiente** de productos claros y concisos;
- Ayudar a establecer la **planificación empírica de productos** para un entorno complejo;
- Facilitar la **colaboración de las partes** interesadas según sea solicitado o necesario.

El Scrum Master sirve a la organización de varias maneras, incluyendo:
- **Liderar**, **capacitar** y **mentorizar** a la organización en su adopción de Scrum;
- Planificar y **asesorar** sobre la **implementación de Scrum** dentro de la organización;
- Ayudar a las personas y a las partes interesadas a **comprender** y **promulgar** un **enfoque empírico** para el trabajo complejo;
- **Eliminar las barreras** entre las partes interesadas y los equipos de Scrum.

Eventos de SCRUM

El Sprint es un contenedor para todos los eventos. Cada evento en Scrum es una oportunidad formal para inspeccionar y adaptar los artefactos de Scrum. Estos eventos están diseñados específicamente para permitir la transparencia necesaria. Si no se realizan los eventos según lo prescrito, se pierden oportunidades para inspeccionar y adaptarse. Los eventos se utilizan en Scrum para crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. De manera óptima, todos los eventos se llevan a cabo al mismo tiempo y lugar para reducir la complejidad.

Sprint. Son eventos de longitud fija de un mes o menos para crear consistencia. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior.
Todo el trabajo necesario para alcanzar el objetivo del producto, incluyendo la Planificación (Sprint Planning), Daily Scrums, Revisión del Sprint (Sprint Review) y la Retrospectiva (Sprint Retrospective), ocurren dentro del Sprints.
Durante el Sprint:
- **No** se hacen **cambios** que pongan en **peligro** el **Objetivo Sprint**;
- La **calidad no disminuye**;
- El **trabajo pendiente** del producto **se refina** según sea necesario;
- El **alcance se puede clarificar** y renegociar con el PO a medida que se aprende más.

Los Sprints permiten la previsibilidad al garantizar la inspección y adaptación del progreso hacia un objetivo del Producto, como mínimo, una vez al mes en el calendario. Cuando el horizonte de un Sprint es demasiado largo, el Objetivo de Sprint puede volverse obsoleto, la complejidad puede aumentar y el riesgo puede aumentar. Los Sprints más cortos se pueden emplear para generar más ciclos de aprendizaje y limitar el riesgo de coste y esfuerzo a un período de tiempo más pequeño. Cada Sprint puede considerarse un proyecto corto.
En entornos complejos, se desconoce lo que sucederá. Solo lo que ya ha sucedido se puede utilizar para la toma de decisiones con vistas a futuro.
Un Sprint podría ser cancelado si el Objetivo del Sprint se vuelve obsoleto. Solo el PO tiene la autoridad para cancelar el Sprint.

1. **Sprint Planning.** El Sprint Planning inicia el Sprint estableciendo el trabajo que se realizará para el mismo. Este plan resultante es creado por el trabajo colaborativo de todo el equipo de Scrum.
   El PO se asegura de que los asistentes estén preparados para discutir los elementos de trabajo pendiente de producto más importantes y cómo se asignan al objetivo del producto.
   Se abordan tres temas:
   Tema Uno: ¿Por qué este Sprint es valioso? El PO propone cómo el producto podría aumentar su valor y utilidad en el Sprint actual. A continuación, todo el equipo de Scrum colabora para definir un objetivo de Sprint que comunique por qué el Sprint es valioso para las partes interesadas. El Objetivo Sprint debe finalizarse antes del final de la Planificación de Sprint.

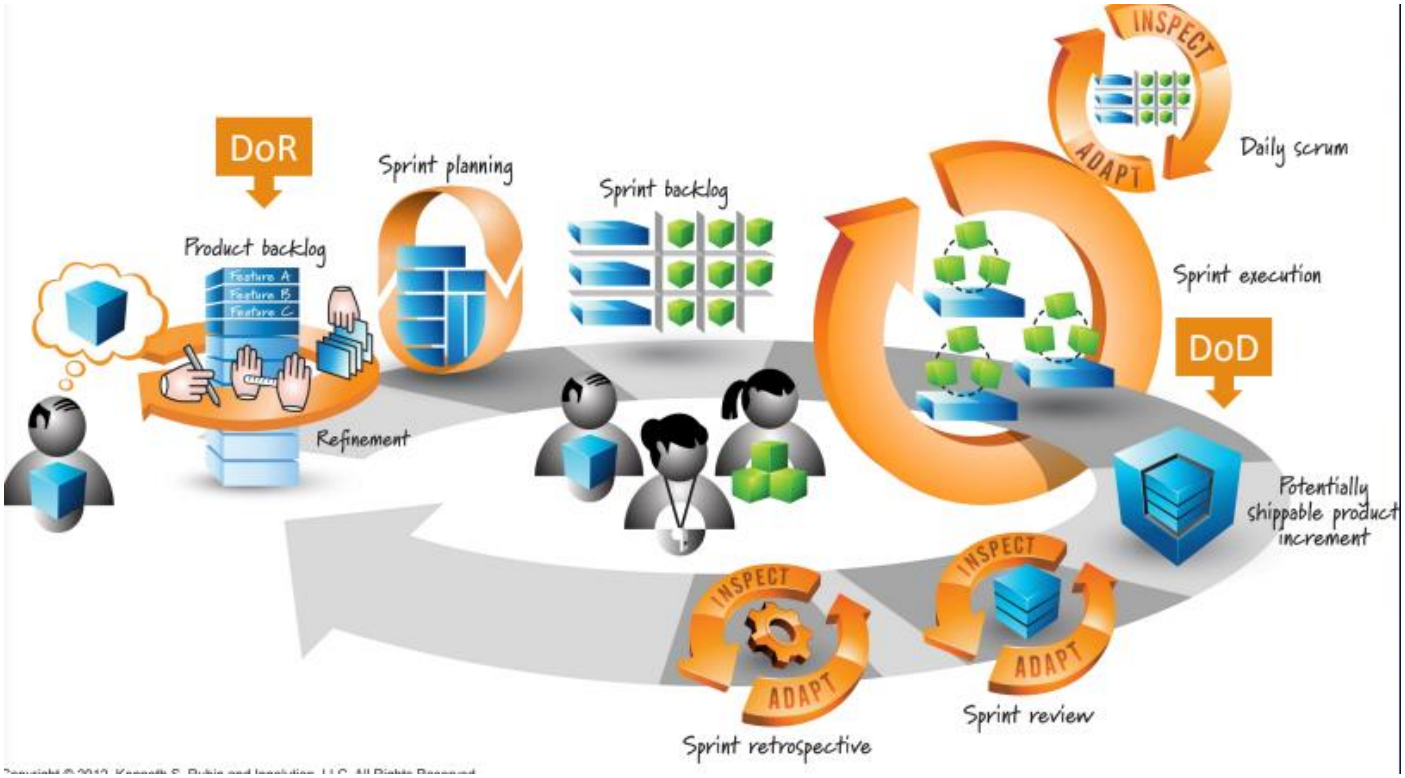
Tema dos: ¿Qué se puede hacer este Sprint? A través del debate con el PO, los desarrolladores seleccionan los elementos del Product Backlog para incluir en el Sprint actual. El equipo de Scrum puede refinar estos elementos durante este proceso, lo que aumenta la comprensión y confianza.
Seleccionar cuánto se puede completar dentro de un Sprint puede ser un desafío. Sin embargo, cuanto más sepan los desarrolladores sobre su rendimiento pasado, su capacidad futura y su definición de hecho, más seguro estarán en sus pronósticos de Sprint.
Tema Tres: ¿Cómo se realizará el trabajo elegido? Para cada elemento de trabajo pendiente de producto (Product Backlog item) seleccionado, los desarrolladores planifican el trabajo necesario para crear un incremento que cumpla con la definición de hecho. Esto se hace normalmente mediante la descomposición de elementos de trabajo pendiente (Product Backlog item) del producto en elementos de trabajo más pequeños que se puedan realizar en un día o menos. La forma de hacerlo es según la discreción de los propios desarrolladores. Nadie más les dice cómo convertir los elementos de trabajo pendiente del producto en incrementos de valor.
El objetivo de Sprint (Sprint Goal), los elementos de trabajo pendiente de producto seleccionados para el Sprint, más el plan para entregarlos se conocen conjuntamente como el trabajo pendiente de Sprint (**Sprint Backlog**).

2. **Daily Scrum.** El propósito del Daily Scrum es inspeccionar el progreso hacia el Objetivo Sprint y adaptar el Sprint Backlog según sea necesario, ajustando el próximo trabajo planeado.
   El Daily Scrum es un evento para los desarrolladores del equipo de Scrum. Para reducir la complejidad, se lleva a cabo al mismo tiempo y lugar todos los días laborables del Sprint. Si el propietario del producto o el Scrum Master están trabajando activamente en los elementos del Trabajo pendiente de Sprint, participan como desarrolladores.
   Los Scrums diarios (Daily Scrum) mejoran la comunicación, identifican impedimentos, promueven una rápida para la toma de decisiones, y en consecuencia, eliminan la necesidad de otras reuniones.
   El Daily Scrum no es la única vez que los desarrolladores pueden ajustar su plan. Frecuentemente se reúnen durante todo el día para debatir de forma más detalladas sobre la adaptación o replanificación del resto del trabajo del Sprint.

3. **Sprint Review.** El propósito de la revisión del Sprint es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. El equipo de Scrum presenta los resultados de su trabajo a las partes interesadas clave y se discute el progreso hacia el Objetivo de Producto.
   Durante el evento, el equipo de Scrum y las partes interesadas revisan lo que se logró en el Sprint y lo que ha cambiado en su entorno. En base a esta información, los asistentes colaboran en qué hacer a continuación. El trabajo pendiente del producto también se puede ajustar para satisfacer nuevas oportunidades. Sprint Review es una sesión de trabajo y el equipo de Scrum debe evitar limitarla a que se convierta en una simple presentación.

4. **Sprint Retrospective.** El propósito de la retrospectiva Sprint es planificar formas de aumentar la calidad y la eficacia.
   El equipo de Scrum inspecciona cómo fue el último Sprint con respecto a individuos, interacciones, procesos, herramientas y su definición de Hecho. Los elementos inspeccionados a menudo varían según el dominio del trabajo. Las suposiciones que los desviaron se identifican y se exploran sus orígenes. El equipo de Scrum analiza qué fue bien durante el Sprint, qué problemas encontró y cómo esos problemas fueron (o no fueron) resueltos.
   El equipo de Scrum identifica los cambios más útiles para mejorar su eficacia. Las mejoras más impactantes se abordan lo antes posible. Incluso se pueden agregar al Sprint Backlog para el próximo Sprint.
   La retrospectiva Sprint concluye el Sprint.

5. **Product Backlog Refinement.** No se considera un evento formal de scrum, pero se suele realizar en todas las implementaciones del marco de trabajo.
   El refinamiento de Backlog del producto es el acto de descomponer y definir aún más los elementos de trabajo pendiente del producto en artículos más pequeños y precisos. Esta es una actividad en curso para agregar detalles, como una descripción, un pedido y un tamaño. Los atributos a menudo varían con el dominio del trabajo.



Artefactos de SCRUM

Los artefactos de Scrum representan trabajo o valor. Están diseñados para maximizar la transparencia de la información clave. Por lo tanto, cada uno de los que los inspecciona tienen la misma base para la adaptación.

Cada artefacto contiene un compromiso para garantizar que proporciona información que mejora la transparencia y el enfoque con el que se puede medir el progreso:

- Para el **Product Backlog** es el **objetivo del producto**.
- Para el **Sprint Backlog** es el **objetivo del sprint**.
- Para el **Incremento** es la **Definición de Hecho**.

Estos compromisos existen para reforzar el empirismo y los valores de Scrum para el equipo de Scrum y sus partes interesadas.

- **Product Backlog.** El product backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente de trabajo emprendida por el equipo Scrum.
  Los elementos del product backlog que pueden ser hecho por el equipo de Scrum dentro de un Sprint se consideran listos para su selección en un evento de planificación de Sprint. Por lo general adquieren este grado de transparencia después de las actividades de refinación.
  Los desarrolladores que realizarán el trabajo son responsables del tamaño. El PO puede influir en los desarrolladores ayudándoles a entender y seleccionar mejores alternativas.
  Compromiso: Objetivo del producto (Product Goal)
  El objetivo del producto (Product Goal) describe un estado futuro del producto que puede servir como objetivo para el equipo Scrum contra el cual planificar. El objetivo del producto se encuentra en el Product Backlog. El resto del product backlog surge para definir "qué" cumplirá el objetivo del producto.
  El objetivo del producto es el objetivo a largo plazo para el equipo Scrum. Deben cumplir (o abandonar) un objetivo antes de asumir el siguiente.

- **Sprint Backlog.** El sprint backlog se compone del objetivo sprint (por qué), el conjunto de elementos de sprint backlog (qué), así como un plan accionable para entregar el incremento (cómo).
  El sprint backlog es un plan por y para los desarrolladores. Es una imagen muy visible y en tiempo real del trabajo que los desarrolladores planean realizar durante el Sprint para lograr el Objetivo Sprint. Por lo tanto, el Sprint Backlog se actualiza a lo largo del Sprint a medida que se aprende más. Debe tener suficientes detalles para que puedan inspeccionar su progreso en el Scrum Diario.
  Compromiso: Sprint Goal
  El Sprint Goal es el único objetivo para el Sprint. Aunque el objetivo de Sprint es un compromiso de los desarrolladores, proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo. El

Objetivo Sprint también crea coherencia y enfoque, animando al equipo de Scrum a trabajar juntos en lugar de en iniciativas separadas.
El objetivo de Sprint se crea durante el evento Sprint Planning y, a continuación, se agrega al sprint backlog.

- **Product Increment.** Un Incremento es un paso de hormigón hacia el Objetivo del Producto. Cada Incremento es aditivo a todos los Incrementos anteriores y verificado a fondo, asegurando que todos los Incrementos funcionen juntos. Para proporcionar el valor, el incremento debe ser utilizable.
  Se pueden crear varios incrementos dentro de un Sprint. La suma de los Incrementos se presenta en la sprint review apoyando así el empirismo. Sin embargo, un Incremento puede ser entregado a las partes interesadas antes del final del Sprint. La revisión de Sprint nunca debe considerarse una puerta para liberar valor.
  El trabajo no se puede considerar parte de un Incremento a menos que cumpla con la Definición de Hecho.
  Compromiso: Definición de Hecho (Definition of Done)
  La Definición de Hecho es una descripción formal del estado del Incremento cuando cumple con las medidas de calidad requeridas para el producto.
  En el momento en que un elemento de trabajo pendiente de producto cumple con la definición de hecho, se crea un incremento.
  La definición de Hecho crea transparencia al proporcionar a todos una comprensión compartida de qué trabajo se completó como parte del Incremento. Si un elemento de sprint backlog no cumple con la definición de hecho, no se puede liberar, ni siquiera presentar en la revisión de Sprint. En su lugar, vuelve al product backlog para su consideración futura.
  Los desarrolladores deben ajustarse a la definición de Hecho. Si hay varios equipos de Scrum trabajando juntos en un producto, deben definir y cumplir mutuamente con la misma definición de hecho.

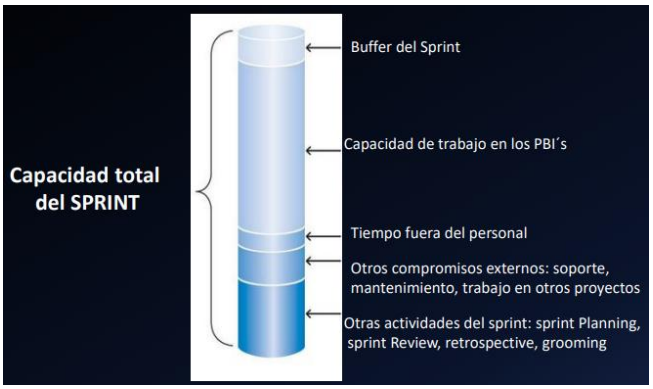
Timebox en SCRUM
Esto existe para que no se pierda más tiempo del necesario en las distintas tareas y para aprender a negociar en términos del alcance del producto y no del tiempo o los costos (relacionado con la triple restricción).
Si no se llega al final de un sprint, en vez de extender el tiempo, se entrega menos.
De esta manera, nos volvemos más confiables, con un ritmo de trabajo sostenible. Todas las ceremonias son Timebox. El refinamiento del Product Backlog al ser una actividad continua, no tiene definido un tiempo exacto, sino que el tiempo se expresa en términos porcentuales sobre la definición del Sprint.

Estos tiempos están pensados para una Sprint de duración de 1 mes, por lo que si la sprint es mas corta, el tiempo de duración de todas los eventos también lo será.

Definition of Ready (DoR). El DoR es un criterio que define cuando una US está lo suficientemente bien formulada como para poderse incluir en un Sprint.
Es el valor de negocio claramente expresado. Contiene los detalles suficientemente comprendidos por el equipo de forma tal que puedan tomar una decisión informada sobre si pueden completar el **ítem del product Backlog (PBI)**.
Se deben identificar todas las dependencias, y no debe haber dependencias externas que puedan impedir que el PBI se complete. El equipo debe ser asignado adecuadamente para completar el PBI.
El PBI se debe encontrar estimado y debe ser lo suficientemente pequeño como para ser completado en una sprint. Además, debe contar con criterios de aceptación claros y testeables, criterios de performance (si los hay), también claros y testeables, y el equipo comprende como mostrar el PBI en el sprint review.

Definition of Done (DoD). es un criterio que dice cuando una historia está lo suficientemente bien implementada como para entrar al Sprint Review y ser mostrada al PO, entonces él será el encargado de aprobar o no la US y en caso de que esté aprobada decidirá si desea ponerla en producción.

Para cumplir con el DoD el **diseño** debe estar **revisado**, el **código completo** (refactorizado, en formato estándar, comentado, en el repositorio e inspeccionado), la **documentación de usuario** debe estar **actualizada**, debe estar **probado** (prueba de unidad, integración y sistema hechas), debe contar con cero defectos conocidos, deben estar las **pruebas de aceptación realizada**, y debe encontrarse en los **servidores de producción**.



Capacidad del Equipo en una Sprint

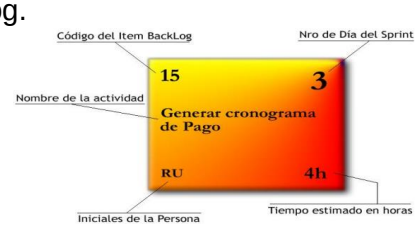
La capacidad es una métrica que utiliza SCRUM para determinar cuánta cantidad de trabajo puede comprometer un equipo para un determinado Sprint. La capacidad se estima y se mide.

Una de las cosas que se hace en el Sprint Planning es determinar la capacidad del equipo, y entonces, teniendo en cuenta la capacidad, se contrasta en cuántas US del Product Backlog se pueden incorporar en el Sprint Backlog.

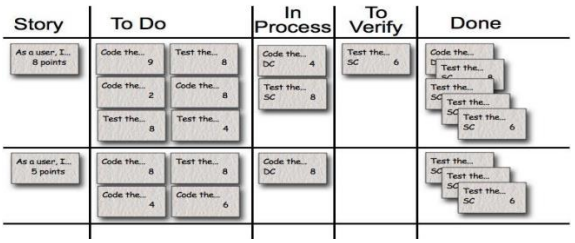
Para equipos más maduros, la capacidad puede ser estimada en Story Points y para equipos menos maduros se puede estimar en horas ideales.

Herramientas de SCRUM

- ✓ **Tarjeta de Tarea.** Es una tarjeta que cuenta con 5 partes:
  1. Código del Ítem Backlog: es un identificador del ítem en el Product Backlog.
  2. Nombre de la actividad: suele ser una frase verbal que define qué es lo que hay que hacer.
  3. Iniciales del responsable de realizar la tarea.
  4. Número de día del sprint.
  5. Estimación del tiempo para finalizar la tarea.



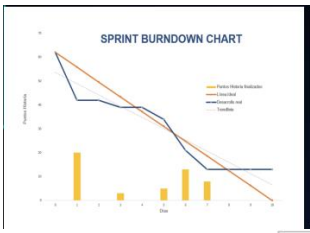
- ✓ **Tablero.** Tablero utilizado por el Equipo de Desarrollo en el cual se colocan las tarjetas de tareas a realizar durante el sprint. Se encuentra dividido en 5 secciones:
  1. **Story**: el nombre de la historia.
  2. **To D**o: aquí se encuentran las tareas por hacer en el Sprint.
  3. **Work In Process**: aquí se encuentran las tareas que se están realizando.
  4. **To Verify**: aquí se encuentran las tareas finalizadas que deben verificarse.
  5. **Done**: aquí se encuentran las tareas terminadas: finalizadas y verificadas.

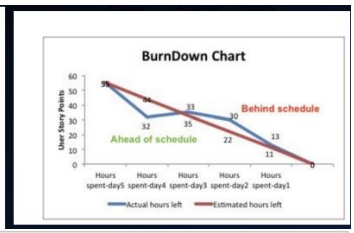


Lo ideal es que el nivel de granularidad de las tareas sea lo más fino posible, de tal forma que se detalle profundamente el avance del sprint, teniendo varias tareas en cada sección del tablero.

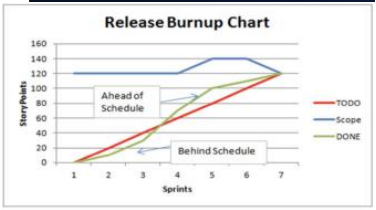
Gráficos del Backlog

Gráficos que brindan información acerca del progreso de un Sprint, de una release o del producto. El backlog de trabajo es la cantidad de trabajo que queda por ser realizado (en horas), mientras que la tendencia del backlog compara esta cantidad con el tiempo (medido en días).



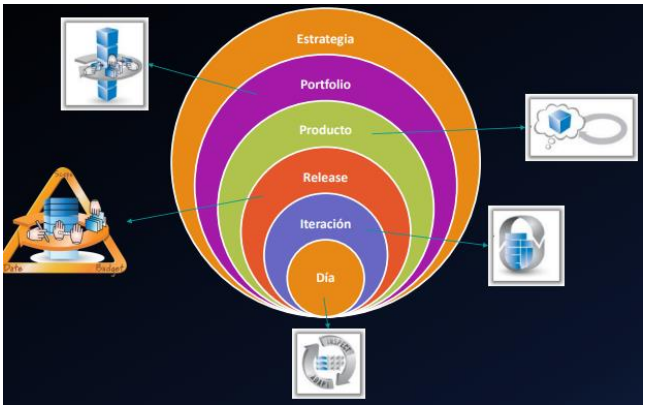


- ➢ **Sprint Burndown Chart.** Visualiza con una curva descendente cuánto trabajo queda para terminar. En el eje de las abscisas se coloca el tiempo que va desde el inicio del sprint. En el eje de las ordenadas se coloca los puntos de historia a realizar en el sprint.
- ➢ **Release Burnup Chart.** Visualiza con una curva ascendente cuánto trabajo se ha completado a lo largo de la release, es decir que visualiza los puntos de historia terminados en cada sprint.



Niveles de Planificación

Nivel	Horizonte	Quién	Foco	Entregable
Portfolio	1 año o más	Stakeholders y Product Owners	Administración de un Portfolio de Producto	Backlog de Portfolio
Producto	Arriba de varios meses o más	Product Owner y Stakeholders	Visión y evolución del producto a través del tiempo	Visión de Producto, Roadmap y características de alto nivel
Release	3 (o menos) a 9 meses	Equipo Scrum, Stakeholders	Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto	Plan de Release
Iteración	Cada iteración (de 1 semana a 1 mes)	Equipo Scrum	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Equipo Scrum (al menos los que trabajan en IPB)	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

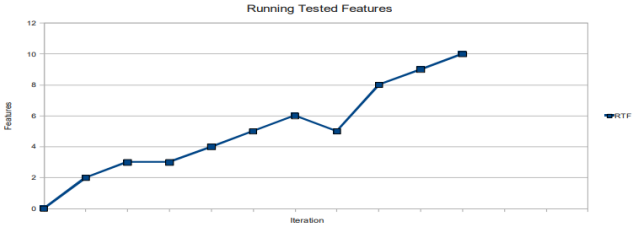


Métricas en Ambientes Agiles

Las métricas sirven para un equipo y no son extrapolables por lo que dice el agilismo a otros proyectos o equipos. En ágil hay un principio que habla específicamente de métricas (la mejor métrica de progreso es el software funcionando). Este principio apunta a que vamos a medir producto (no proceso ni proyecto). Esto es una reacción a proyectos tradicionales que tenían todas las métricas posibles cubriendo los tres enfoques, pero no teniendo avances sobre el producto (se perdía mucho tiempo y no se progresaba).
Se debe tener en cuenta que la medición es una salida, no una actividad. Solo se debe **medir lo que sea necesario y nada más**, es decir, lo que agregue valor para el cliente.

Existen dos principios agiles que guían la elección de las métricas:
- Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso.
- El Software trabajando es la principal medida de progreso.

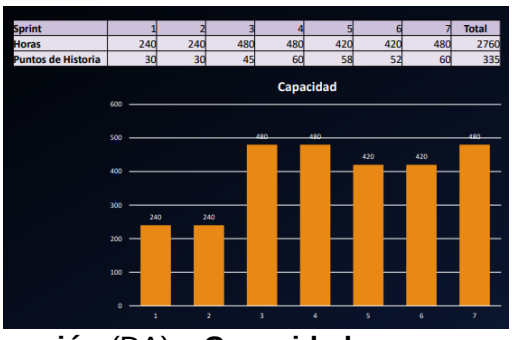
Running Tested Features (RTF). Mide la cantidad de features testeadas que están funcionando, es decir, cuantas piezas de producto (historias de usuarios, casos de usos, requerimientos) se terminaron y están en ejecución.
El problema de esta métrica es que no tiene en cuenta la complejidad de las piezas de producto que se implementan. Por ejemplo, si se toma como piezas de producto a historias de usuario, se mide cuántas historias de usuarios están funcionando, pero no se sabe de cuantos puntos de historias tiene cada una de ellas. Es una medición absoluta.



Si la curva es constante (llana) o tiene pendiente negativa, entonces indica la existencia de un problema, ya que las características no incrementan o disminuyen en el tiempo. Esta última situación se presenta cuando una funcionalidad del sistema deja de ser válida.

Capacidad. Es una métrica de proyecto que mide el compromiso de un equipo para un determinado sprint, en horas de trabajo ideales. Es por esto que, se utiliza para planificar, ya que permite definir cuántas historias de usuario se van a tomar del Product Backlog para implementar en el próximo sprint.
Se estima al principio de un sprint, por lo que también se la llama velocidad estimada. Se puede estimar en horas ideales o en Story Points.

Sprint	1	2	3	4	5	6	7	Total
Horas	240	240	480	480	420	420	480	2760
Puntos de Historia	30	30	45	60	58	52	60	335

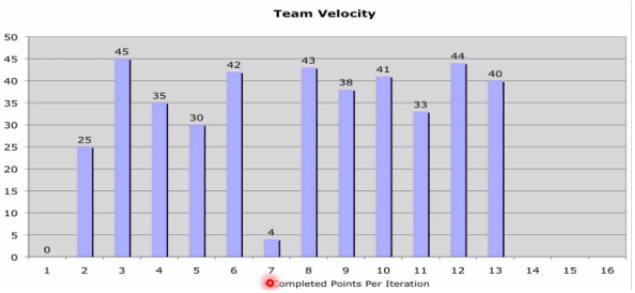


Cálculo de la Capacidad:

Horas de Trabajo Disponible por día (WH) x **Días Disponibles Iteración** (DA) = **Capacidad**
WH x DA = Capacidad

Algunas consideraciones:
- Individuos deben calcular capacidad realista
- Aplicar estimaciones honestas a sus tareas
- Considerar una capacidad máxima de 50% - 70%
- Comprender la capacidad a largo plazo con la velocidad y los puntos de historia
- Conocer promedio de finalización de un punto de historia para equipo/individuo

Velocidad (Velocity). Es la métrica más importante del enfoque (métrica de producto), y mide la cantidad de puntos de historia que se realizaron en un Sprint y fueron aceptados por el Product Owner. Hay que recordar que no se cuentan las historias parcialmente terminadas, sólo las que están completadas y aceptadas por el PO.



Esta métrica se calcula (no se estima) luego de que el PO. acepte la implementación de una US.

Esta métrica suele representarse con un gráfico de barras para medir la estabilidad del equipo. Estos gráficos son permanentes durante todo el proyecto y son visibles para todo el mundo.

Esta métrica, y sus valores a lo largo de un proyecto ágil, permiten analizar si el equipo posee una estabilidad a lo largo del mismo, lo cual también se relaciona con un principio del manifiesto ágil (desarrollo sostenible).